A Performance Comparison of Linux and a Lightweight Kernel

Ron Brightwell, Rolf Riesen, Keith Underwood
Sandia National Laboratories*
PO Box 5800
Albuquerque, NM 87185-1110
{rbbrigh,rolf,kdunder}@sandia.gov

Trammell B. Hudson
Operating Systems Research, Inc.
Albuquerque, NM
hudson@osresearch.net

Patrick Bridges, Arthur B. Maccabe
Computer Science Department
University of New Mexico
Albuquerque, NM 87131
{bridges,maccabe}@cs.unm.edu

Abstract

In this paper, we compare running the Linux operating system on the compute nodes of ASCI Red hardware to running a specialized, highly-optimized lightweight kernel (LWK) operating system. We have ported Linux to the compute and service nodes of the ASCI Red supercomputer, and have run several benchmarks. We present performance and scalability results for Linux compared with the LWK environment. To our knowledge, this is the first direct comparison on identical hardware of Linux and an operating system designed specifically for large-scale supercomputers. In addition to presenting these results, we will discuss the limitations of both operating systems, in terms of the empirical evidence as well as other important factors.

1. Introduction

Linux is the preferred operating system of choice for commodity clusters. In addition to its well-documented use on small- and medium-sized clusters [18], Linux is making in-roads on large-scale machines. In particular, while previous large-scale systems such as ASCI Red [19] used a custom operating systems for compute nodes (e.g., Cougar [17]) and commodity UNIX kernels on service nodes, (e.g., OSF1[20]) many new large-scale machines are using Linux on both compute and service nodes (e.g., Sandia National Laboratories' Cplant[tm] cluster [4], Lawrence Livermore National Laboratory's MCR cluster [1], Los Alamos National Laboratory's Pink cluster [2], and SGI's Altix 3000 [14]).

The IBM Blue Gene/L [3] and ASCI Red Storm [5] machines are the notable exceptions. Both of these systems plan to use a specialized lightweight operating system on compute nodes instead of a commodity operating system. Red Storm is using a lightweight kernel based on Cougar, and IBM is using a new, internally developed, high-performance microkernel. However, both of these systems plan to use Linux on their service nodes.

Commodity hardware trends are helping Linux. In large-scale machines of the past, memory was an expensive and scarce resource. Today, memory is cheap and plentiful. The footprint of the operating system is much more important on a 16 MB node than it is on a 2 GB node. Processor cycles are much cheaper as well, especially for commodity clusters where the performance of the processor relative to the network is significantly greater. Cluster machines with significantly fast processors and relatively slower networks may not need to be highly optimized for delivering the most processor cycles. We have previously questioned the use of commodity operating systems in such an environment [15], primarily on the grounds that commodity operating systems do not offer the scalability and reproducible performance found in lightweight kernels. Our research is exploring the peformance impacts of Linux on non-cluster hardware platforms, but we are also conversely exploring the appropriateness of using lightweight kernels for commodity clusters. This paper presents our initial empirical results of comparing Linux with a lightweight kernel on a traditional supercomputer in terms of absolute performance, scalability to large problems, and deterministic performance.

* 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.

Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03)
0-7695-2066-9/03 $17.00 © 2003 IEEE

2. Related Work

We are aware of no empirical studies comparing the appropriateness of lightweight kernels to modern commodity kernels on current-day large-scale supercomputers. Recent work has compared Windows NT, Linux, and QNX in terms of the appropriate OS for a cluster [9]. This work contrasts the feature sets of the various kernels and presents some single-node OS micro-benchmarks. It does not, however, look at overall impact operating system structure on application performance on a well-balanced architecture, the primary goal of our work.

A direct predecessor to this work compares OSF1/AD [20] and SUNMOS [10] on the Intel Paragon using BLAS, FFT, and NPB [16]; however, there have been significant advances in commodity operating systems (spurred by the Open Source movement) as well as commodity microprocessors (e.g., significant increases in the number of TLB entries) since that time. This work revisits the comparison of lightweight kernels and full featured operating systems by analyzing Cougar and Linux on ASCI Red.

3. ASCI Red Hardware

ASCI Red [19] is large-scale supercomputer comprised of more than 4500 dual-processor nodes connected by a high-performance custom network fabric. Each compute node has two 333 MHz Pentium II Xeon processors each with a 16 KB level-1 cache and a 256 KB level-2 cache, along with 256 MB of main memory, for a total of more than 9000 processors and more than a terabyte of distributed main memory. ASCI Red does not in general support global shared memory; data is shared between nodes using explicit message-passing.

Each compute node has a network interface, called a CNIC, that resides on the memory bus and allows for low-latency access to all of physical memory on a node. The CNIC interface connects each node to a 3-d mesh network that provides a 400 MB/s uni-directional wormhole-routed connection between the nodes; this results in a system bisection bandwidth that is greater than 50 GB/s. The CNIC interface is capable of sustaining the 400MB/s node-to-node transfer rate to and from main memory across the entire machine. This interface appears essentially as two FIFOs on the system bus; one for transmitting and one for receiving.

4. Cougar for ASCI Red

The compute nodes of ASCI Red run Cougar, a variant of the Puma lightweight kernel that was designed and developed by Sandia and the University of New Mexico for maximizing both message passing throughput and application resource availability [17]. Puma was ported by Intel from the Paragon to ASCI Red, at which time Intel productized it as Cougar. Sandia continued development of Puma as a research project while Intel continued to develop and maintain Cougar specifically for ASCI Red.

4.1. System Structure

Compared to most other operating systems, Cougar is structurally relatively simple because it was designed explicitly for high-performance message-passing systems. In Cougar, all of the resources on a compute node are managed by the *system processor*. This is the only processor that performs any significant processing in supervisor mode. The remaining processor runs application code and only rarely enters supervisor mode. This processor is called the *user processor*. This arrangement produces a slight asymmetry in the performance of the processors, but it greatly simplifies the structure of the Cougar kernel and maximizes the processor cycles available to the applications.

Another significant difference between Cougar and most general purpose operating systems is Cougar maps all available memory contiguously. This simplifies the implementation of high performance message passing systems as well as lowers TLB thrashing; contiguous mapping allows simple use of TLB superpages, allowing mappings for half of physical memory to reside in the TLB.

4.2. Networking Support

Cougar uses a simple network protocol built around the Portals message passing interface [17]. Portals are data structures in an application's address space that determine how the kernel should respond to message passing events. Portals allow the kernel to deliver messages directly from the network to the application's memory. Once a Portal has been set up, the kernel has all of the information necessary to deliver a message directly to the application. Messages for which there is no corresponding Portal description are simply discarded.

Portals enables low-latency, high-bandwidth message delivery in Cougar because the operating system easily can arrange for incoming message data to be delivered directly to application space once the kernel has pulled the message header from the CNIC FIFO. This allows Cougar to keep pace with the 400 MB/s CNIC network interface. It is important to note that Portals is relatively straightforward to implement on Cougar because of its simple memory management strategy; in general purpose operating systems with complex memory management systems and protocol stacks, implementation of Portals is significantly more challenging.

4.3. Processor Modes

Cougar is not a traditional symmetric multi-processing operating system. Instead, it supports four different modes that allow different distributions of application processes on the processors. The processor mode is determined at runtime for the processes in a parallel job when the job is launched. The following provides an overview of each of these processor modes. More details can be found in [11].

The simplest processor usage mode is to run both the kernel and application process on the system processor. This mode is commonly referred to as "heater mode" since the second processor is not used and only generates heat. In this mode, the kernel runs only when responding to network events or in response to a system call from the application process. This mode does not offer any significant performance advantages to the application process.

In the second mode, message co-processor mode, the kernel runs on the system processor and the application process runs on the user processor. When the processors are configured in this mode, the kernel runs continuously waiting to process events from external devices or service system call requests from the application process. Because the time to transition from user mode to supervisor mode and back can be significant, this mode offers the advantage of reduced network latency and faster system call response time. Because of the increased message passing performance, this mode favors applications that are latency bound.

In the third mode, compute co-processor mode, the system processor and user processor both run the kernel and an application process. However, the kernel code running on the application processor does not perform any resource management activities, it simplify notifies the system processor when a system call is performed. The advantage of this mode is that it provides more processor cycles for the application. However, the two processors are not symmetric since the part of the application running on the shared system processor will not progress as rapidly as the portion of the application running on the dedicated user processor. In order to use this mode, the application typically uses a non-standard library interface that executes a co-routine on the application processor; specialized libraries (e.g. BLAS, OpenMP) have been implemented that make efficient use of this processing mode. Because of the opportunity to utilize both processors, this mode favors applications that are compute-bound.

Finally, in the fourth mode, known as virtual node mode, the system processor runs both the kernel and an application process, while the second processor also runs the kernel and a full separate application process. This mode essentially allows a compute node to be viewed by the runtime system as two independent compute nodes. The asymmetry of compute co-processor mode also exists in this mode, so the application process running on the user processor is likely to receive slightly more processor cycles than the application process running with the kernel on the system processor. This mode allows applications to avail of the user processor more easily, since the application does not need to be modified to use the non-standard co-routine interface.

5. Linux for ASCI Red

5.1. System Structure

Unlike Cougar, Linux is a general-purpose, interrupt-driven kernel that supports symmetric multiprocessing. This makes it relatively easy to port applications to Linux clusters, but makes it somewhat challenging to use in a dedicated high-performance environment. As previously mentioned, a variety of projects have supplemented Linux with support for medium- and large-scale clusters.

As part of this study, we ported the Linux 2.4 kernel to the specialized compute node hardware as well as the more general-purpose service node hardware. The diskless compute nodes receive their kernel from the OSF service nodes via a bootmesh protocol. The Linux bootloader and startup code were adapted to work with this protocol. All of the nodes in the machine are connected to a management network that provides serial console access.

On boot, the Linux service node receives its kernel via bootmesh, then continues to boot RedHat Linux from a direct attached SCSI disk. This provided the entire service environment for the parallel machine; users could compile their test codes with gcc and a full UNIX environment. Additionally, the service node exports an NFS root filesystem over the mesh for the diskless compute nodes.

The compute nodes receive their kernels via bootmesh and then mount their root filesystems over the mesh from the service node. The runtime environment on the compute nodes is very sparse – sshd for remote access and enough libraries for MPI jobs to run. No other daemons or processes were running on the nodes.

5.2. Networking Support

As part of our port, we implemented a Linux driver for the CNIC interface based on an example network driver (`isa-skeleton.c`), which implements a ring buffer, interrupt driven network interface. With this driver, Linux can use TCP/IP to communicate over the mesh—the Linux software layer provides all of the higher-level functionality, including MPICH 1.2.5.

The IP MTU for the CNIC device was varied between 4 KB (one page) and 16 KB (four pages) with no noticeable effect on bandwidth. Overall, the network was CPU limited due to the depth of the TCP/IP stack. For 333 MHz nodes

it was 45 MB/s and only 32 MB/s on the older 200 MHz nodes. With the faster nodes, this is only 11% of the available bandwidth and scales almost directly with the clock rate of the CPU.

To verify that the bandwidth was limited by the IP stack, a custom raw device was implemented. This achieved 310 MB/s of bandwidth, demonstrating that Linux was capable of making full use of the mesh even with 4 KB pages. It is possible that even better performance would be possible with the 4 MB "superpages" as described in [13].

It is possible to port the Portals message passing layer to work with the Linux CNIC driver, but this would require much more time and effort. The memory handling semantics of Linux are much more complex than that of Cougar. The generality of Linux comes at a cost. Nevertheless, we are pursuing an implementation or Portals in Linux for the CNIC.

5.3. Processor Modes

The CNIC driver was modified to support the same computational modes as Cougar (heater, message co-processor and user co-processor). However, little difference was seen between the different modes on Linux because the kernel must handle interrupts from the CNIC driver and perform TCP processing regardless of the processing mode. Virtual node mode is essentially "free" since the Linux kernel is an SMP capable multitasking kernel.

6. Evaluation Codes

6.1. Latency Test

The MPI ping-pong latency test measures the half round trip time between two nodes with pre-posted received for various message lengths. Both operating systems have a clock timer function that provides microsecond accuracy.

6.2. NPB 2.4

The NAS Parallel Benchmarks (NPB) are a collection of MPI applications that are distilled from real computational fluid dynamics applications[6]. They all exhibit particular message passing and computation patterns that stress different parts of the system. The three that we have tested are Conjugate Gradient (CG), Multi-grid (MG) and Integer Sort (IS).

The first of these, CG, solves an unstructured sparse linear system by the conjugate gradient method. It uses the inverse power method to find an estimate of the largest eigenvalue of a symmetric positive definite sparse matrix with a random pattern of non-zeros. CG requires all-to-all communication, so performance depends highly on the performance of the network.

The MG benchmark uses a multi-grid method to approximate the solution to a scalar Poisson problem on a discrete 3D mesh. This solution is found by solving approximating coarse approximations of the problem followed by refining that solution. It performs a large amount of computation at the local level, but at coarse approximations may communicate with distant nodes. Global reductions are also performed after each time step to maintain boundary conditions.

IS generates local random numbers then performs a bucket sort of the local data. After all nodes have sorted their data, they all use `MPI_Alltoallv` to exchange region requests, followed by `MPI_Rsend`'s to exchange the actual data. The bucket sort algorithm does not make very good use of locality and stresses the TLB, while the all-to-all communication stresses the communication layer.

6.3. CTH

The CTH [7] family of codes developed at Sandia models complex multi-dimensional, multi-material problems characterized by large deformations and/or strong shocks. It uses a two-step, second-order accurate finite-difference Eulerian solution algorithm to solve the mass, momentum, and energy conservation equations. CTH has material models for equations of state, strength, fracture, porosity, and high explosives. The production CTH software family runs on a variety of computing platforms, from low-end PC's to high-end massively parallel supercomputers. It is used extensively within both the Department of Defense and the Department of Energy laboratory complexes for studying armor/anti-armor interactions, warhead design, high explosive initiation physics, and weapons safety issues.

To assess the performance and scalability of CTH, a three-dimensional simulated problem was designed. This problem has moving materials filling the entire computational mesh throughout the simulation time, so load balancing problems are not evident. A series of calculations with different problem sizes was performed on 1, 32, and 64 nodes. For each calculation, the problem size was scaled such that the total number of computational cells allocated on each node and the resolution of the problem remained the same, independent of the number of nodes. Each calculation was repeated a few times so that the average performance index was as objective as possible.

7. Performance

The performance results we present are from a 144-node development system that contains the same hardware as the

production ASCI Red machine.

The performance comparison of Linux and Cougar yielded interesting results. In the latency test, shown in Figure 1, Cougar clearly outperforms the Linux implementation. This is primarily a difference in the underlying message transport. Linux uses MPICH over TCP while Cougar uses an MPICH derivative over the Portals transport layer. Just as interesting as the performance is the shape of the curves. Latency measurements on Cougar form an almost perfectly straight line scaling with message size. The same measurements on Linux have significantly more jitter.

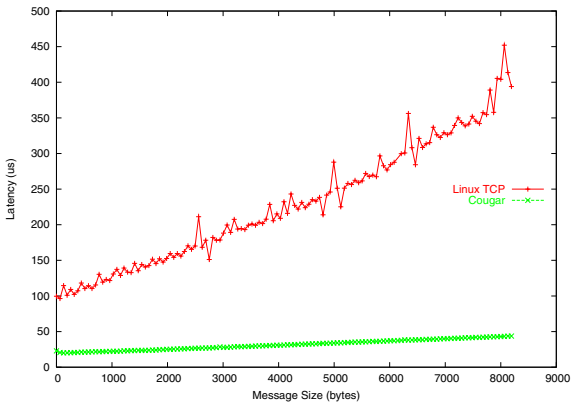

Figure 1. MPI latency performance.

The three NPB codes illustrate three interesting points in the comparison space. For each benchmark, the results are presented in total MOPS (millions of operations per second). For CG (Figure 2), Linux slightly outperforms Cougar on a small number of nodes where communication is a less significant factor. This result was a bit of a mystery at first, since there seemed to be little reason for a lightweight kernel to perform more poorly than Linux; however, we quickly realized that the application was compiled with a much newer compiler on Linux than on Cougar. Up to date tools is an issue for the LWK approach discussed further in Section 8. As the number of processors increase, however, scalability issues override the difference in single node performance and Cougar demonstrates significantly better performance.

Figure 3 shows that Cougar and Linux performance is almost identical for MG with small numbers of processors. In this case, differences in the compiler are outweighed by advantages of the LWK approach. Although communications issues do not seem to limit MG on Linux at 16, 32, or 64 processors, Cougar begins to have a significant advantage at 128 processors.

Finally, Figure 4 illustrates that Cougar significantly outperforms Linux at small numbers of nodes when running IS. This application highlights two of the advantages of Cougar: physically contiguous virtual memory and advanced collective operations. With contiguous memory, there is drastically lower pressure on the processor's TLB. Under Linux, the performance effects of TLB thrashing are clear. This gap only widens as the number of processors increase because the collective operations (specifically `MPI_Alltoall`) implemented for ASCI Red are significantly better than the default collectives in MPICH 1.2.5.

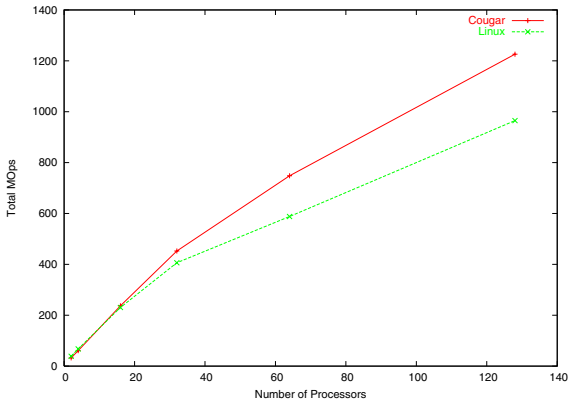

Figure 2. NPB 2.4 CG performance

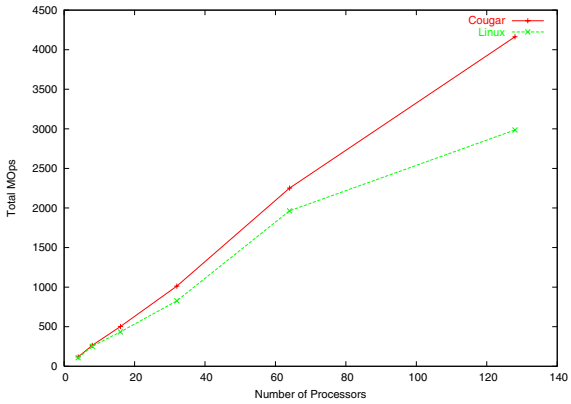

Figure 3. NPB 2.4 MG performance

For each of the NAS benchmarks, Cougar demonstrates drastically better scalability than Linux. This is unsurprising since Cougar achieves much better MPI bandwidth and latency than Linux; however, this should not be considered a completely unfair comparison. For these tests, Linux and Cougar run their native messaging layers. Additional work is underway to add a lighter-weight transport layer to Linux and port a run-time environment to it.

Figure 5 shows the compute performance of CTH on the different platforms. The results are presented in total runtime. This does not include CTH startup overhead, but does

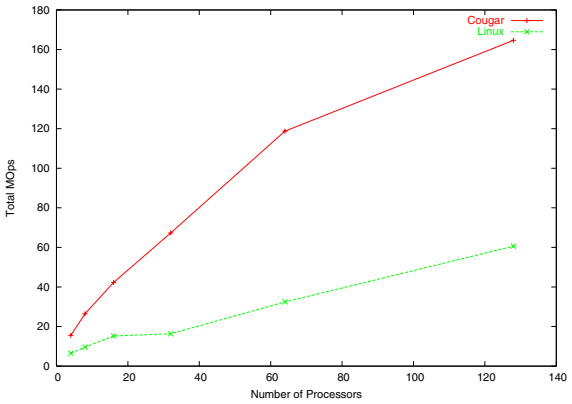

Figure 4. NPB 2.4 IS performance

include communications time on multiprocessor runs. For this real application, Cougar has a significant performance advantage at one node and maintains that advantage out to several processors. Unlike some of the NAS benchmarks, the scalability difference is not dramatic because CTH is highly compute bound, spending 90% of the time computing.

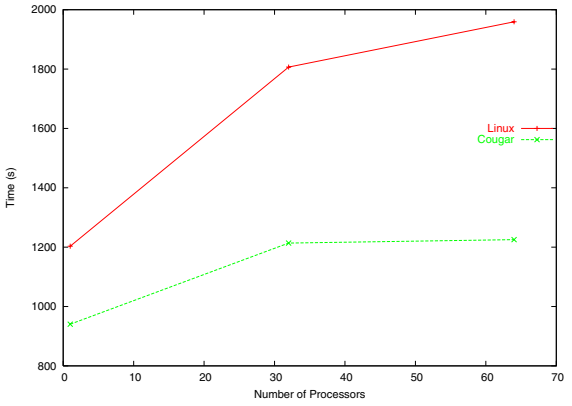

Figure 5. CTH performance

8. Other Issues

Performance impacts of the operating system may be a significant issue, but it is not the only issue in choosing a lightweight kernel approach to the compute node operating system. For example, the compiler technology and run-time environment available for an OS can have a far greater impact on application performance than the operating system. Another important issue is the level of determinism and reproducibility that the operating system offers. And, potentially most significant, is the level of effort involved in creating and maintaining an OS for the system.

8.1. Compilers and Run-time

Perhaps the single biggest factor in the time an application takes on a given serial node is the compiler technology used. This is most prominent when comparing serial node execution times on Linux and Cougar. In some cases, Cougar falls behind. We currently believe that the difference in the compiler and in the version of standard libraries used are the source of this difference. The compilers for Cougar simply have not been updated for several years while newer compilers for Linux are readily available.

Similarly, the version of other system software components has not been changed in many years. For example, Linux is using MPICH 1.2.5 while Cougar is still running a derivative of MPICH 1.0.12. This is yet another byproduct of differences in the support models for the two operating systems. Linux is widely used and supported by the community while ports to Cougar require effort internally.

8.2. Determinism

An interesting and desirable advantage of lightweight kernels, and Cougar in particular, is a high level of determinism and reproducibility. Simply put, there is virtually no variability in application execution times on Cougar. Whether on a serial node or a fixed group of nodes on the parallel machine, there is significantly less than 1% variability in the execution time of an application. This supports application tuning and debugging much better than comparable commodity operating systems. It also provides much better support for system software experimentation. If tests indicate that a system software change yields a 2% increase in application performance on Cougar, that difference is statistically significant. In general, the same cannot be said for commodity operating systems where application performance can easily range 5% or more across multiple runs.

8.3. Effort Level

The most controversial issue for comparison is the level of effort required to develop and maintain an operating system for a supercomputing platform. On one hand, the core of Linux is developed by the open source community. Many bugs and much experimentation is done to optimize the operating system at no cost to supercomputer owners. In contrast, a lightweight kernel, such as Cougar, is often developed completely in-house.

However, that is not the complete picture. The time it took to write the original version of the SUNMOS

lightweight kernel for the Intel Paragon is comparable to the the time it has taken to port Linux to ASCI Red hardware. The current code base of Cougar is only about 30 thousand lines of code versus 80 thousand lines of code in Linux (for equivalent core functionality). In fact, the heart Cougar is not much bigger than some Linux drivers.

There is also a significant amount of effort going into finding and fixing performance and scalability problems with commodity operating systems and runtime systems on large-scale machines. It is debatable whether the effort spent trying to work around inherent problems in commodity operating systems might be better spent developing and maintaining a custom operating system that does not have inherent limitations.

9. Conclusions

With the current data, there is no clear winner between Linux and Cougar. We do, however, present clear advantages and disadvantages to lightweight kernels on compute nodes. Preliminary data indicates that the Cougar operating system scales much better on well balanced hardware than Linux. Although some would argue that the comparison is unfair due to differences in the communication layer, it is important to note that the lightweight communication layer in Cougar was created by design as an integral part of the operating system. In that sense, the comparison is "apples to apples" since both platforms are benchmarked running their default configurations. In some cases, Cougar also offers improved performance with small numbers of nodes by reducing TLB thrashing through the use of physically contiguous virtual memory. This is despite the significant disadvantage in compiler technology. Finally, Cougar has a distinct advantage in providing deterministic performance. The adverse effects of operating systems studied on supercomputing platforms[12, 8] do not appear.

Linux, however, does show noticeable performance advantages over Cougar on some applications when a small number of nodes are used. Although data is still being gathered, this is probably due to the newer compilers and libraries available on Linux. The advantages of using a widely used and supported operating systems should not be underestimated. Unless compiler and library updates are continually funded for a given lightweight kernel, commodity operating systems such as Linux will always have newer, better development environments; however, contrary to popular belief, the use of Linux is not free. Modern supercomputing platforms are not commodity hardware and thus require a significant development effort just to make Linux run. Specialized hardware and system configurations require special drivers and boot mechanisms to be developed and maintained along with the kernel. Just as importantly, the focus of Linux is not on high performance scientific computing and so there is seldom a drive to include or maintain kernel features that are important to the supercomputing community.

10. Future Work

To provide a full comparison of Linux and the lightweight kernel approach, it will be necessary to address the current limitations in each test platform. For Linux, it will be necessary to complete a Portals messaging layer and to port the appropriate run-time environment (MPI and application launch). This should dramatically increase the network bandwidth that Linux can achieve. This would be a partial commodity, partial custom approach to the compute node operating system. In addition, it would be interesting to experiment with support for larger pages appearing in the 2.5 development version of the Linux kernel. For Cougar, it is important to provide a more modern set of compilers, libraries, and run-time environment. These improvements should easily bring the performance of a single node into parity with Linux in all cases while retaining the significant wins provided on some applications.

11. Acknowledgments

We gratefully acknowledge the contributions of the other members of lightweight kernel research team at Sandia. Mike Levenhagen, Kurt Ferreira, Marcus Epperson, and Zhaofang Wen have provided valuable assistance in the collection of this data.

We would also like to express our sincere thanks to the ASCI Red maintenance team, who provided invaluable hardware support for our experiments with Linux on the ASCI Red development systems.

This work was supported by the Mathematical, Information, and Computational Sciences (MICS) program of the DOE Office of Science.

References

[1] `http://www.llnl.gov/linux/mcr/`.

[2] `http://www.lanl.gov/projects/pink/`.

[3] N. R. Adiga et al. An Overview of the Blue Gene/L Supercomputer. In *In Proceedings of the SC 2002 Conference on High Performance Networking and Computing*, Baltimore, MD, November 2002.

[4] R. Brightwell, L. A. Fisk, D. S. Greenberg, T. B. Hudson, M. J. Levenhagen, A. B. Maccabe, and R. E. Riesen. Massively Parallel Computing Using Commodity Components. *Parallel Computing*, 26(2-3):243–266, February 2000.

[5] W. J. Camp and J. L. Tomkins. Thor's Hammer: The First Version of the Red Storm MPP Architecture. In *In Proceedings of the SC 2002 Conference on High Performance Networking and Computing*, Baltimore, MD, November 2002.

[6] R. F. V. der Wijngaart. NAS Parallel Benchmarks Version 2.4. Technical report, October 2002.

[7] J. E.S. Hertel, R. Bell, M. Elrick, A. Farnsworth, G. Kerley, J. McGlaun, S. Petney, S. Silling, P. Taylor, and L. Yarrington. CTH: A Software Family for Multi-Dimensional Shock Physics Analysis. In *Proceedings of the 19th International Symposium on Shock Waves, held at Marseille, France*, pages 377–382, July 1993.

[8] A. Hoise, D. J. Kerbyson, S. Pakin, F. Petrini, H. J. Wasserman, and J. Fernandez-Peinador. Identifying and Eliminating the Performance Variability on the ASCI Q Machine. `http://wwwc3.lanl.gov/par_arch/pubs/QBperf010203.pdf`, January 2003.

[9] A. Kavas and D. G. Feitelson. Comparing Windows NT, Linux, and QNX as the Basis for Cluster Systems. *Concurrency and Computation: Practice and Experience*, 13:1303–1332, 2001.

[10] 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.

[11] A. B. Maccabe, R. Riesen, and D. W. van Dresser. Dynamic Processor Modes in Puma. *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS)*, 8(2):4–12, 1996.

[12] R. Mraz. Reducing the Variance of Point To Point Transfers in the IBM 9076 Parallel Computer. In *Proceedings of the 1994 conference on Supercomputing*, pages 620–629. IEEE Computer Society Press, 1994.

[13] J. Navarro, S. Iyer, P. Druschel, and A. Cox. Practical, Transparent Operating System Support for Superpages. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, December 2002.

[14] S. Neuner. Scaling Linux to New Heights: the SGI Altix 3000 System. *Linux Journal*, (106), February 2003.

[15] R. R. Ron Brightwell, Arthur B. Maccabe. On the Appropriateness of Commodity Operating Systems for Large-Scale, Balanced Computing Systems. In *International Parallel and Distributed Processing Symposium*, 2003.

[16] S. Saini and H. D. Simon. Applications performance under OSF/1 AD and SUNMOS on Intel Paragon XP/S-15. In *Proceedings of the 1994 ACM/IEEE conference on Supercomputing*, pages 580–589. ACM Press, 1994.

[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, pages I:11–14, Boca Raton, FL, August 1995. CRC Press.

[19] S. R. W. Timothy G. Mattson, David Scott. A TeraFLOPS Supercomputer in 1996: The ASCI TFLOP System. In *Proceedings of the 1996 International Parallel Processing Symposium*, 1996.

[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, January 1993.