Paravirtualization For HPC Systems [⋆]

Lamia Youseff[1], Rich Wolski[1], Brent Gorda[2], and Chandra Krintz[1]

[1] Department of Computer Science
University of California, Santa Barbara
[2] Lawrence Livermore National Lab (LLNL)

Abstract. In this work, we investigate the efficacy of using paravirtualizing software for performance-critical HPC kernels and applications. We present a comprehensive performance evaluation of Xen, a low-overhead, Linux-based, virtual machine monitor, for paravirtualization of HPC cluster systems at LLNL. We investigate subsystem and overall performance using a wide range of benchmarks and applications. We employ statistically sound methods to compare the performance of a paravirtualized kernel against three Linux operating systems: RedHat Enterprise 4 for build versions 2.6.9 and 2.6.12 and the LLNL CHAOS kernel. Our results indicate that Xen is very efficient and practical for HPC systems.

1 Introduction

Virtualization is a widely used technique in which a software layer multiplexes lower-level resources among higher-level software programs and systems. Examples of virtualization systems include a vast body of work in the area of operating systems [32, 31, 25, 30, 4, 16], high-level language virtual machines such as those for Java and .Net, and, more recently, virtual machine monitors (VMMs). VMMs virtualize entire software stacks including the operating system (OS) and application, via a software layer between the hardware and the OS of the machine. VMM systems enable application and full-system isolation (sand-boxing), OS-based migration, distributed load balancing, OS-level check-pointing and recovery, non-native (cross-system) application execution, and support for multiple or customized operating systems.

Virtualization historically has come at the cost of performance due to the additional level of indirection and software abstraction necessary to achieve system isolation. Recent advances in VMM technology however, address this issue with novel techniques that reduce this overhead. One such technique is paravirtualization [1] which is the process of strategically modifying a small segment of the interface that the VMM exports along with the OS that executes using it. Paravirtualization significantly simplifies the process of virtualization (at the cost of perfect hardware compatibility) by eliminating special hardware features and instructions in the OS that are difficult to virtualize efficiently. Paravirtualization systems thus, have the potential for improved scalability and performance over prior VMM implementations. A large number of popular VMMs employ paravirtualization in some form to reduce the overhead of virtualization including Denali [1], IBM rHype [41], Xen [28, 40, 11], and VMWare [20, 33, 38]. Moreover, hardware vendors now employ new ways of enabling efficient virtualization in the next-generation processors [37, 29] which have the potential to further improve the performance of VMM-based execution.

⋆ This work is sponsored in part by LLNL and NSF (CNS-0546737 and ST-HEC-0444412).

Despite the potential benefits, performance advances, and recent research indicating its potential [22, 44, 15, 19], virtualization is currently not used in high-performance computing (HPC) environments. One reason for this is the perception that the remaining overhead that VMMs introduce is unacceptable for performance-critical applications and systems. The goal of our work is to evaluate empirically and to quantify the degree to which this perception is true for Linux and Xen.

Xen is an open-source VMM for the Linux OS which reports low-overhead and efficient execution of Linux [40]. Linux, itself, is the current operating system of choice when building and deploying computational clusters composed of commodity components. In this work, we study the performance impact of Xen using current HPC commodity hardware at Lawrence Livermore National Laboratory (LLNL). Xen is an ideal candidate VMM for an HPC setting given its large-scale development efforts [28, 42] and its availability, performance-focus, and evolution for a wide range of platforms.

We objectively compare the performance of benchmarks and applications using a Xen-based Linux system against three Linux OS versions and configurations currently in use for HPC application execution at LLNL and other super-computing sites. The Linux versions include Red Hat Enterprise Linux 4 (RHEL4) for build versions 2.6.9 and 2.6.12 and the LLNL CHAOS kernel, a specialized version of RHEL4 version 2.6.9.

We collect performance data using micro- and macro-benchmarks from the HPC Challenge, LLNL ASCI Purple, and NAS parallel benchmark suites among others, as well as using a large-scale, HPC application for simulation of oceanographic and climatologic phenomena. Using micro-benchmarks, we evaluate machine memory and disk I/O performance while our experiments using the macro-benchmarks and HPC applications assess full system performance.

We find that Xen paravirtualization system, in general, does not introduce significant overhead over other OS configurations that we study – including one specialized for the HPC cluster we investigate. There is one case for which Xen overhead is significant: random disk I/O. Curiously, in a small number of other cases, Xen improves subsystem or full system performance over various other kernels due to its implementation for efficient interaction between the guest and host OS. Overall, we find that Xen does not impose an onerous performance penalty for a wide range of HPC program behaviors and applications. As a result we believe the flexibility and potential for enhanced security that Xen offers makes it useful in a commodity HPC context.

2 Background and Motivation

Our investigation into the performance implications of coupling modern virtualization technologies with high performance computing (HPC) systems stems from our goal to improve the flexibility of large-scale HPC clusters at Lawrence Livermore National Laboratory (LLNL). If virtualization does not impose a substantial performance degradation, we believe it will be useful in supporting important system-level functionalities such as automatic checkpoint/restart and load balancing.

For example, several researchers have explored OS and process migration, such as Internet Suspend/Resume [21] and μDenali [39]. Recent studies on OS image migration [17, 12] illustrate that migrating an entire OS instance with live interactive services is achievable with very little down time (e.g. 60ms) using a VMM. To be effective in a production HPC environment, however, the operating system and language systems must all be commonly available and standardized to ease the system administration burden. Thus it is the virtualization of Linux (the basis for a large proportion of cluster installations) that is of primary interest.

In addition, it is possible for one cluster to run different Linux images which aids software maintenance (by providing an upgrade path that does not require a single OS "upgrade" event) and allows both legacy codes and new functionality to co-exist. This is important for legacy codes that execute using a particular version of the OS and/or obsolete language-level libraries that depend on a specific OS kernel release level. VMMs also enable very fast OS installation (even more when coupled with effective checkpointing), and thus, their use can result significant reductions in system down time for reboot. Finally, VMMs offer the potential for facilitating the use of application-specific and customized operating systems [22, 44, 15, 19].

Though many of the benefits of virtualization are well known, the perceived cost of virtualization is not acceptable to the HPC community, where performance is critical. VMMs by design introduce an additional software layer, and thus an overhead, in order to facilitate virtualization. This overhead however, has been the focus of much optimization effort recently. In particular, extant, performance-aware, VMMs such as Xen [28], employ *paravirtualization* to reduce virtualization overhead. Paravirtualization is the process of simplifying the interface exported by the hardware in a way that eliminates hardware features that are difficult to virtualize. Examples of such features are *sensitive* instructions that perform differently depending on whether they are executed in user or kernel mode but that do not trap when executed in user mode; such instructions must be intercepted and interpreted by the virtualization layer, introducing significant overhead. There are a small number of these instructions that the OS uses that must be replaced to enable execution of the OS over the VMM. No application code must be changed to execute using a paravirtualizing system such as Xen. A more detailed overview of system-level virtual machines, sensitive instructions, and paravirtualization can be found in [34].

To investigate the performance implications of using paravirtualization for HPC systems, we have performed a rigorous empirical evaluation of HPC systems with and without virtualization using a wide range of HPC benchmarks, kernels, and applications, using LLNL HPC hardware. Moreover, we compare VMM-based execution with a number of non-VMM-based Linux systems, including CHAOS (a Linux distribution and kernel based on Red Hat Enterprise Release 4) that is currently employed by, and specialized for LLNL users and HPC clusters.

3 Methodology and Hardware Platform

Our experimental hardware platform consists of a four-node cluster of Intel **E**xtended **M**emory 64 **T**echnology (EM64T) machines. Each node consists of four Intel Xeon 3.40 GHz processors, each with a 16KB L1 data cache and a 1024KB L2 cache. Each node has 4GB of RAM and a 120 GB SCSI hard disk with DMA enabled. The nodes are interconnected with an Intel PRO/1000, 1Gigabit Ethernet network fabric using the ch_p4 interface with TCP/IP.

We perform our experiments by repeatedly executing the benchmarks and collecting the performance data. We perform 50 runs per benchmark code per kernel and compute the average across runs. We perform a *t-test* at the $\alpha \geq 0.95$ significance level to compare the means of two sets of experiments (e.g. those from two different kernels). The t-test tells us whether the difference between the observed means is statistically significant (see [23] and [8] for clear and readable treatments of the procedure we employ).

3.1 HPC Linux Operating System Comparison

We empirically compare four different HPC Linux operating systems. The first two are current releases of the RedHat Enterprise Linux 4 (RHEL4) system. We employ builds v2.6.9 and v2.6.12 and refer to them, respectively, as *RHEL2.6.9* and *RHEL2.6.12*.

We also evaluate the CHAOS kernel. CHAOS is the **C**lustered, **H**igh-**A**vailability, **O**perating **S**ystem [13, 10] from LLNL. CHAOS is a Linux distribution based on RHEL4 v2.6.9 that LLNL computer scientists have customized for the LLNL HPC cluster hardware and for the specific needs of current users. In addition, CHAOS extends the original distribution with new administrator tools, support for very large Linux clusters, and HPC application development. Examples of these extensions include utilities for cluster monitoring, system installation, power/console management, and parallel job launch, among others. We employ the latest release of CHAOS as of this writing which is v2.6.9-22; we refer to this system as CHAOS kernel in our results.

Our Xen-based Linux kernel (host OS) is RHEL4 v2.6.12 with a Xen 3.0.1 patch. Above Xen, the guest kernel is a paravirtualized Linux RHEL4 v2.6.12, which we configure with 4 virtual CPUs and 2GB of virtual memory. We refer to this overall configuration as *Xen* in our results. Xen v3 is not available for Linux v2.6.9, the latest version for which the CHAOS extensions are available. We thus, include both v2.6.9 and v2.6.12 (non-CHAOS and non-XEN) in our study to identify and isolate any performance differences between these versions.

	Benchmark Category	**Code Name**	**What it measures**
Micro	Memory	Stream	Mem read/write rate (MB/s)
	Disk I/O	Bonnie	Seq & Rand disk I/O (MB/s)
Macro	Parallel Benchmarks	NAS Parallel Benchmark; class C *Multigrid (MG) in: 512^3* *LU Solver (LU) in: 162^3* *Integer Sort (IS) in: 2^{27}* *Embarrassingly parallel (EP) in: 2^{32}* *Conjugate gradient (CG) in: 150000*	Total time (s) and millions of operations per second (Mops)
App	Scientific Simulations	MIT GCM exp2	Total time (s)

Table 1. Benchmark Overview

3.2 Benchmarks

We overview the benchmarks that we use in this empirical investigation in Table 1. The benchmarks set consists of micro-benchmarks, macro-benchmarks, and real HPC applications. We employ the same benchmark binaries for all operating system configurations.

Our micro-benchmark set includes programs from the HPC Challenge [24] and LLNL ASCI Purple Benchmark suite [3]. The programs are specifically designed to evaluate distinct performance characteristics of machine subsystems. In this work, we focus on memory and disk I/O benchmarks since the other macro-benchmarks and codes we use for evaluation test other performance characteristics more directly.

We use the HPCC/LLNL ASCI Purple benchmark Stream [36] to evaluate memory access performance. Stream reports the sustainable memory bandwidth in MB/s for four different memory operations: Copy (read/write of a large array), and three operations (Scale, Sum, and Triad) that combine computation with memory access to measure the corresponding computational rate for simple vector operations.

For evaluation of disk performance, we employ Bonnie [9]. Bonnie is a disk stress-test that uses popular UNIX file system operations. Bonnie measures the system I/O throughput for six different patterns of reads, writes, and seeks. We employ three different file sizes: 100MB, 500MB and 1GB for our experiments to eliminate any cache impact on measured performance.

To evaluate the full system and computational performance, we employ several popular macro-benchmarks from the NAS Parallel benchmark suite [6, 5]. The former set is from the NASA Advanced Supercomputing (NAS) facility at the NASA Ames Research Center. The suite evaluates the efficiency of highly parallel HPC computing systems in handling critical operations that are part of simulation of the future space missions. The benchmarks mimic the computational, communication, and data movement characteristics of large-scale, computational fluid dynamics (CFD) applications.

We also include an HPC application in our study, the General Circulation Model (GCM) from the Massachusetts Institute of Technology (MIT). GCM is a popular numerical model used by application scientists to study oceanographic and climatologic phenomena. GCM simulates ocean and wind currents and their circulation in the earth's atmosphere thousands of years in advance. A widely used implementation of GCM is made available by MIT Climate Modeling Initiative (CMI) team [27]. Researchers commonly integrate this implementation into oceanographic simulations. The MIT CMI team supports a publicly available version [2, 26], which we employ and refer to in this paper as *MIT GCM*. The MIT GCM package has been carefully optimized by its developers to ensure low overhead and high resource utilization.

The package includes a number of inputs. We use the sequential version of `exp2` for this study. Exp2 simulates the planetary ocean circulation at a 4 degree resolution. The simulation uses twenty layers on the vertical grid, ranging in thickness between 50m at the surface to 815m at depth. We configure the experiment to simulate 1 year of ocean circulation at a one-second resolution.

4 Micro-Benchmarks

In this section, we evaluate the impact of Xen on specific subsystems of our cluster system. We consider memory and disk I/O subsystems.

4.1 Memory Access Performance

Sustainable memory bandwidth is another important performance aspect for HPC systems, since long cache miss handling can hinder the computational power attainable by any machine. To study the impact of paravirtualization on sustainable memory bandwidth, we use Stream [36], which we configure with the default array size of 2 million elements.

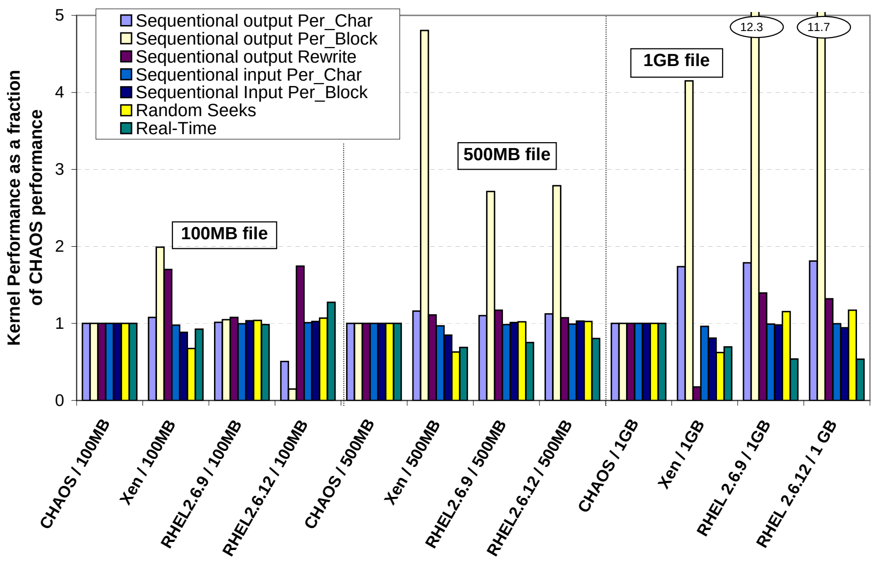

Fig. 2. Bonnie Disk I/O bandwidth rate and real-time relative to CHAOS performance

Figure 1 shows the results. CHAOS attains the highest memory bandwidth for all stream operations. This is the result of CHAOS optimizations by LLNL computer scientists for memory-intensive workloads. Surprisingly, Xen attains consistently higher memory bandwidth by approximately 1-2% for every operation over RHEL2.6.12. The t-value for the difference ranges between 12-14, indicating that the differences between Xen and RHEL2.6.12 measurements is statistically significant.

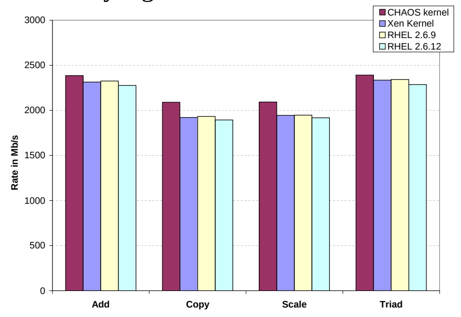

Fig. 1. Stream memory performance (Mb/s)

Since Xen uses asynchronous I/O rings for data transfers between the guest OS and the host OS, it is able to reorder requests and amortize each for better memory performance. The Xen I/O ring algorithm was wise enough to arrange the requests produces by domU on behalf of the stream code, and exploited their sequential nature to gain performance and memory bandwidth. On the other hand, these gains are less apparent between the Xen and RHEL2.6.9 configurations.

4.2 Disk I/O Performance

Disk performance of virtualized systems is also a concern for applications that perform significant disk I/O such as those for scientific database applications. To measure this performance, we use the Bonnie I/O benchmark. For the Xen kernel, we configure an LVM-backed virtual block device (VBD).

Bonnie reads and writes sequential character input and output in 1K blocks using the standard C library calls `putc()`, `getc()`, `read()`, and `write()`. For the Bonnie `rewrite` test, Bonnie reads, dirties, and writes back each block after performing an `lseek()`. The Bonnie random I/O test performs an `lseek()` to random locations in the file, then then `read()` to reads a block from that location. For these events, Bonnie rewrites 10% of the blocks.

Figure 2 shows the performance of Bonnie for the four kernels relative to the performance of CHAOS. The x-axis is the performance of the different disk I/O metrics, for different file sizes (y-axis). The first three bars in each group show the performance of the sequential output tests; the next two bars are for the sequential input test; the sixth bar is for the random test; and the last bar is total time.

Xen has a higher per-character output, per-block output, and rewrite rate for all file sizes relative to CHAOS. Xen performance is slower for sequential output rewrite for the 1GB file. CHAOS has not been optimized for disk I/O. The 1GB sequential output rewrite performance using Xen is the result of Xen's disk scheduling algorithm. As described previously, Xen used an I/O descriptor ring for each guest domain, to reduce the overhead of domain crossing upon each request. Each domain posts its request in the descriptor ring; the host OS consumes them as they are produced. This results in producer-consumer problem that the authors of Xen describe in [28]. The improvements from Xen I/O are the result of reordering of I/O requests by the host OS to enable highly efficient disk access. In the case of sequential output for 1GB files, the requests are very large in number and randomly generated across the file. This prevents Xen from making efficient use of the I/O rings and optimizing requests effectively. This effect is also apparent and significant in the results from the random seek tests. These results indicate that if random seeks to large files is a key operation in a particular HPC application, the Xen I/O implementation should be changed and specialized for this case. This scenario is fortunately not common in HPC applications.

The sequential character performance is not significantly different across kernels. However, for sequential input per block, Xen disk I/O speed lags behind the other three kernels by about 11-17%. This performance degradation is caused by Xen's implementation of I/O buffer rings. The default size of the buffer rings fail to optimize block input performance. However, tuning the I/O buffer rings in Xen Dom0 can improve on the performance for such workloads.

5 Macro-Benchmarks

Paravirtualization offers many opportunities to HPC applications and software systems, e.g., full system customization, check-pointing and migration, etc. As such, it is important to understand the performance implications that such systems impose for a wide range of programs and applications. We do so in this section for the popular NAS parallel benchmarks and the MIT GCM oceanographic and climatologic simulation system. This set of experiments shows the impact of using Xen for programs that exercise the complete machine (subsystems in an ensemble).

5.1 NAS Parallel Benchmarks (NPB)

For the first set of experiments we employ the NAS parallel benchmarks (NPB) as we describe in Section 3. The benchmarks mimic the computational, communicational and data movement characteristics of large scale computational fluid dynamics applications.

Figure 3 (a) shows the performance of the NPB codes(x-axis) for our different kernels relative to CHAOS (y-axis). *EP, IS* and *MG* are the Embarrassingly Parallel, Integer Sort, and Multi grid codes respectively, while *LU* is the Linear solver code and *CG* is the Conjugate gradient code. For all of the codes, we choose to run class C

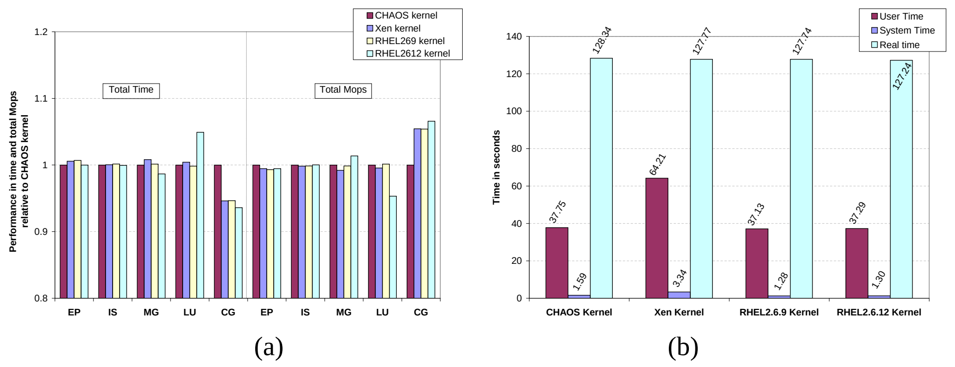

Fig. 3. Xen performance for NAS Benchmark and GCM Application. (a) shows the NAS Parallel Benchmark performance relative to CHAOS. The left half (first benchmark set) is for total time (lower is better); the right half is for Mops (higher is better). (b) is the MIT GCM performance in seconds (lower is better).

benchmark sizes to better asses the different performance implications of virtualization. We present two different metrics for each of the five benchmarks. The left five sets of bars reflect total execution time. The right five are for the total millions of operations per second (Mops) the benchmarks achieve.

All of the kernels perform similarly for EP, IS, and MG. The differences between the bars, though visually different in some cases, are not statistically significant when we compare them using the t-test with 95% confidence. This is interesting since the benchmarks are very different in terms of their behavior: EP performs distributed computation with little communication overhead, IS performs a significant amount of communication using collective operations, and MG employs a large number of blocking send operations. In all cases, paravirtualization imposes no statistically significant overhead.

LU decomposition shows a performance degradation of approximately 5% for RHEL-2.6.12 for both total time and Mops. The reason for this is due to overhead this kernel places on computation. CHAOS optimizes this overhead away and RHEL2.6.9 makes up for this loss due to its low overhead on MPI-based network latency, which we studied in details in [43]. Xen implements a different CPU scheduling policy: a very efficient implementation of the borrowed virtual time (BVT) scheduler [14]. BVT and the overhead of scheduling in general positively impacts the Mflops rate of Xen-based sequential linear solvers. However, Xen network performance places a subtle performance penalty on MPI-based LU code performance. A combination of the scheduling policy and network performance enabled by Xen enables the Xen system to avoid the overhead which was endured by RHEL2.6.12. Xen's computational and communications performance was studied in more details in [43]

The *C*onjugate *G*radient (CG) code computes an approximation to the smallest eigenvalue of a large sparse matrix. It combines unstructured matrix system vector multiplication with irregular MPI communications. CG executes slower using CHAOS than using the other kernels by about 5%. The statistical difference however was not significant, which may mean that the differences was introduced due to noise in the readings. We support this claim using the standard deviation of the 50 measurements that we collected using this kernel: This value is 31 for an average measurement of 607s, in terms of Mops this value is 12 for an average of 237s. In summary, Xen performs consistently comparable to CHAOS and the two RHEL kernels and delivers performance similar to that of natively executed parallel applications.

5.2 MIT GCM

To evaluate the use of virtualization for real HPC applications, we employ the MIT *G*eneral *C*irculation *M*odel (GCM) implementation. MIT GCM is a simulation model for oceanographic and climatologic phenomena. The execution of the MIT GCM using the `exp2` input, involves reading several input files at the beginning of the run for initialization, processing a computationally intensive simulation, check-pointing the processed data to files periodically, and outputting the final results to several other files. The total amount of data that is read and written by the system during each run is approximately 33MB. The individual writes are on the order of 200B per call to write() and the total size of each file is approximately 1MB.

We use the Linux time utility to measure the performance of MIT GCM which reports the time spent executing user code (User Time), the time spent executing system code (System Time), and the total time (Real Time). We present the results in Figure 3 (b). The y-axis is the time in seconds for the kernels shown on the x-axis.

From the experiments, we found Xen execution time of MIT GCM to be slightly faster than that for CHAOS. The difference however, is not statistically significant given a 95% confidence level. Similarly, the difference in performance between Xen and RHEL kernels is negligible.

Our experience with the system indicates that the difference between Xen and CHAOS is primarily due to the disk I/O activity. We also observe that Xen User Time and CPU usage is consistently and uniformly different from that of the other kernels. This is due to the way Xen computes user and system time in the Linux time utility in error. This is a Xen implementation bug that will be fixed in the next version of Xen.

These results are extremely promising, despite the time utility bug. They show that Xen achieves performance equal to that of the RHEL kernels and slightly better than that of CHAOS. In addition, our results from the prior section on disk I/O indicate that Xen is able to mask I/O overhead for common disk activities. Our results show, that Xen can satisfy the performance requirements of real HPC applications such as GCM. We plan to investigate how other applications behave over Xen as part of future work.

6 Related Research

The work related to that which we pursue in this paper, includes performance studies of virtualization-based systems. We investigate a wide range of metrics for HPC benchmarks, applications, and systems. We consider both subsystem performance for a number of important HPC components as well as full-system performance when using paravirtualizing systems for HPC cluster resources (IA64, SMP machines).

Other work investigates the performance of Xen and other similar technologies in a non-HPC setting [28, 11]. This research shows the efficacy and low overhead of paravirtualizing systems. The benchmarks that both papers employ are general-purpose operating systems benchmarks. The systems that the authors evaluate are IA32 and stand-alone machines with a single processor. Furthermore, those papers investigate the performance of the first release of Xen, which has changed significantly. We employ the latest version of Xen (v3.0.1) that includes a wide range of optimization and features not present in the earlier versions, as well as running on SMP machines.

Other work investigates the performance of Xen for clusters as part of an unpublished class project [18]. Researchers have also explored the impact of Xen on network communication [7, 35], the latter provides a minimal evaluation of Xen for an IA64 cluster. The authors of [35] investigate different network switch fabric on Linux clusters including Fast Ethernet, Gigabit Ethernet, and different Myrinet technologies. More recent studies evaluate other features of Xen such as the performance overhead of live migration of a guest OS [12]. They show that live migration can be done with no performance cost, and with down times as low as 60 mseconds. These systems do not rigorously investigate the performance overheads of doing so in an HPC setting.

7 Conclusions and Future Work

Paravirtualizing systems such as Xen, expose opportunities for improved maintenance and customization for HPC systems. In this paper, we evaluate the overhead of using Xen in an HPC environment. We compare three different Linux configurations against a Xen-based kernel. The three non-Xen kernels are those currently in use at LLNL for HPC clusters: RedHat Enterprise 4 (RHEL4) for build versions 2.6.9 and 2.6.12 and the LLNL CHAOS kernel, a specialized version of RHEL4 version 2.6.9. We perform experiments using micro- and macro-benchmarks from the HPC Challenge, LLNL ASCI Purple, and NAS parallel benchmark suites among others, as well as using a large-scale, HPC application for simulation of oceanographic and climatologic phenomena. As a result, we are able to rigorously evaluate the performance of Xen-based HPC systems relative to non-virtualized system for subsystems independently and in ensemble.

Our results indicate that, in general, the Xen paravirtualizing system poses no statistically significant overhead over other OS configurations currently in use at LLNL for HPC clusters – even one that is specialized for HPC clusters – in all but two instances. We find that this is the case for programs that exercise specific subsystems, a complete machine, or combined cluster resources. In the instances where a performance difference is measurable, we detail how Xen either introduces overhead or somewhat counter-intuitively produces superior performance over the other kernels.

As part of future work, we will empirically evaluate the Linux v2.6.12 CHAOS kernel as well as Infiniband network connectivity. In addition, we are currently investigating a number of research directions that make use of Xen-based HPC systems. In particular, we are investigating techniques for high-performance check-pointing and migration of full systems to facilitate load balancing, to isolate hardware error management, and to reduce down time for LLNL HPC clusters. We are also investigating techniques for automatic static and dynamic specialization of OS images in a way that is application-specific [22, 44].

References

[1] A. Whitaker and M. Shaw and S. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2002. `http://denali.cs.washington.edu/`.

[2] A. Adcroft, J. Campin, P. Heimbach, C. Hill, and J. Marshall. *MIT-GCM User Manual*. Earth, Atmospheric and Planetary Sciences, Massachusetts Institute of Technology, 2002.

[3] LLNL ASC Purple Benchmark Suite. `http://www.llnl.gov/asci/purple/benchmarks/`.

[4] J. D. Bagley, E. R. Floto, S. C. Hsieh, and V. Watson. Sharing data and services in a virtual machine system. In *SOSP '75: Proceedings of the fifth ACM symposium on Operating systems principles*, pages 82–88, New York, NY, USA, 1975. ACM Press.

[5] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The nas parallel benchmarks 2.0. *The International Journal of Supercomputer Applications*, 1995.

[6] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The nas parallel benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[7] H. Bjerke. HPC Virtualization with Xen on Itanium. Master's thesis, Norwegian University of Science and Technology (NTNU), July 2005.

[8] BMJ Publishing Group: Statistics at Square One: The t Tests, 2006. `http://bmj.bmjjournals.com/collections/statsbk/7.shtml`.

[9] Bonnie Disk I/O Benchmark. `http://www.textuality.com/bonnie/`.

[10] R. Braby, J. Garlick, and R. Goldstone. Achieving Order through CHAOS: the LLNL HPC Linux Cluster Experience, June 2003.

[11] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J. N. Matthews. Xen and the art of repeated research. In *USENIX Annual Technical Conference, FREENIX Track*, pages 135–144, 2004.

[12] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, USA, May 2005.

[13] Clustered High Availability Operating System (CHAOS) Overview. `http://www.llnl.gov/linux/chaos/`.

[14] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular. In *Symposium on Operating Systems Principles*, pages 261–276, 1999.

[15] Eric Van Hensbergen. The Effect of Virtualization on OS Interference. In *Workshop on Operating System Interference in High Performance Applications, held in cooperation with The Fourteenth International Conference on Parallel Architectures and Compilation Techniques: PACT05* , Septmber 2005. `http://research.ihost.com/osihpa/`.

[16] S. W. Galley. PDP-10 virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 30–34, New York, NY, USA, 1973. ACM Press.

[17] J. Hansen and E. Jul". Self-migration of Operating Systems. In *ACM SIGOPS European Workshop (EW 2004)*, pages "126–130", "2004".

[18] H.Bjerke and R.Andresen. Virtualization in clusters, 2004. `http://haavard.dyndns.org/virtualization/clust_virt.pdf`.

[19] E. V. Hensbergen. PROSE : Partitioned Reliable Operating System Environment. In *IBM Research Technical Report RC23694*, 2005.

[20] J. Sugerman and G. Venkitachalam and B. Lim. Virtualizing I/O devices on VMware workstations hosted virtual machine monitor. In *USENIX Annual Technical Conference*, 2001.

[21] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *WMCSA '02: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications*, page 40, Washington, DC, USA, 2002. IEEE Computer Society.

[22] C. Krintz and R. Wolski. Using phase behavior in scientific application to guide linux operating system customization. In *Workshop on Next Generation Software at IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.

[23] R. J. Larsen and M. L. Marx. *An Introduction to Mathematical Statistics and Its Applications*. Prentice Hall, Third Edition, 2001.

[24] P. Luszczek, J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi. Introduction to the hpc challenge benchmark suite, March 2005. `http://icl.cs.utk.edu/projectsfiles/hpcc/pubs/hpcc-challenge-benchmark05.pdf`.

[25] S. E. Madnick and J. J. Donovan. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the workshop on virtual computer systems*, pages 210–224, New York, NY, USA, 1973. ACM Press.

[26] J. Marotzke and R. G. et al. Construction of the adjoint MIT ocean general circulation model and application to Atlantic heat transport sensitivity. *Journal of Geophysical Research*, 104(C12), 1999.

[27] MIT's Climate Modeling Initiative. `http://paoc.mit.edu/cmi/`.

[28] P. Barham and B. Dragovic and K. Fraser and S. Hand and T. Harris and A. Ho and R. Neugebauer. Virtual machine monitors: Xen and the art of virtualization. In *Symposium on Operating System Principles*, 2003. `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/`.

[29] AMD Virtualization Codenamed "Pacifica" Technology, Secure Virtual Machine Architecture Reference Manual, May 2005.

[30] G. J. Popek and R. P. Goldberg. Formal requirements for virtualizable third generation architectures. *Commun. ACM*, 17(7):412–421, 1974.

[31] G. J. Popek and C. S. Kline. The PDP-11 virtual machine architecture: A case study. In *SOSP '75: Proceedings of the fifth ACM symposium on Operating systems principles*, pages 97–105, New York, NY, USA, 1975. ACM Press.

[32] R.A. Meyer and L.H. Seawright. A Virtual Machine Time Sharing System. In *IBM Systems Journal*, pages 199–218, 1970.

[33] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005.

[34] J. E. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann/Elsevier, 2005.

[35] P. J. Sokolowski and D. Grosu. Performance considerations for network switch fabrics on linux clusters. In *Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, November 2004.

[36] The memory stress benchmark codes: stream. `http://www.llnl.gov/asci/purple/benchmarks/limited/memory/`.

[37] Enhanced Virtualization on Intel Architecture-based Servers, March 2005.

[38] C. A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, 2002.

[39] A. Whitaker, R. Cox, M. Shaw, and S. Gribble. Constructing services with interposable virtual hardware, 2004.

[40] Xen Virtual Machine Monitor Performance. `http://www.cl.cam.ac.uk/Research/SRG/netos/xen/performance.html`.

[41] J. Xenidis. rHype: IBM Research Hypervisor. In *IBM Research*, March 2005. `http://www.research.ibm.com/hypervisor/`.

[42] XenSource. `http://www.xensource.com/`.

[43] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. Paravirtualization for HPC Systems. Technical Report Technical Report Numer 2006-10, Computer Science Department University of California, Santa Barbara, Aug. 2006.

[44] L. Youseff, R. Wolski, and C. Krintz. Linux kernel specialization for scientific application performance. Technical Report UCSB Technical Report 2005-29, Univ. of California, Santa Barbara, Nov 2005.