Paravirtualization For HPC Systems*
UCSB Computer Science Technical Report Number 2006-10

Lamia Youseff[α] Rich Wolski[α] Brent Gorda[β] Chandra Krintz[α]

[α] Department of Computer Science
University of California, Santa Barbara
{lyouseff, rich, ckrintz}@cs.ucsb.edu
[β] Lawrence Livermore National Lab (LLNL)
bgorda@llnl.gov

Abstract

Virtualization has become increasingly popular for enabling full system isolation, load balancing, and hardware multiplexing. This wide-spread use is the result of novel techniques such as paravirtualization that make virtualization systems practical and efficient. Paravirtualizing systems export an interface that is slightly different from the underlying hardware but that significantly streamlines and simplifies the virtualization process.

In this work, we investigate the efficacy of using paravirtualizing software for performance-critical HPC kernels and applications. Such systems are not currently employed in HPC environments due to their perceived overhead. However, virtualization systems offer tremendous potential for benefitting HPC systems by facilitating application isolation, portability, operating system customization, and program migration.

We present a comprehensive performance evaluation of Xen, a low-overhead, Linux-based, virtual machine monitor (VMM), for paravirtualization of HPC cluster systems at Lawrence Livermore National Lab (LLNL). We consider four categories of micro-benchmarks from the HPC Challenge (HPCC) and LLNL ASCI Purple suites to evaluate a wide range of subsystem-specific behaviors. In addition, we employ macro-benchmarks and HPC application to evaluate overall performance in a real setting. We also employ statistically sound methods to compare the performance of a paravirtualized kernel against three popular Linux operating systems: RedHat Enterprise 4 (RHEL4) for build versions 2.6.9 and 2.6.12 and the LLNL CHAOS kernel, a specialized version of RHEL4. Our results indicate that Xen is very efficient and practical for HPC systems.

*This work is sponsored in part by grant from the National Science Foundation (ST-HEC-0444412).

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 [36, 34, 27, 33, 4, 17], 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. VMMs offer a wide range of benefits including 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 came 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 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 paravirtualiza-

tion in some form to reduce the overhead of virtualization including Denali [1], IBM rHype [46], Xen [31, 45, 11], and VMWare [21, 37, 42]. Moreover, hardware vendors now employ new ways of enabling efficient virtualization in the next-generation processors [41, 32] which have the potential for improving VMM-based execution performance further.

Despite the potential benefits, performance advances, and recent research indicating its potential [24, 48, 16, 20], 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 virtual machine monitor for the Linux operating system which reports low-overhead and efficient execution of Linux [45]. 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 [31, 47] 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. 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. We employ four categories of micro-benchmarks that evaluate distinct performance characteristics of machine subsystems including MPI-based network bandwidth and latency, CPU processing, memory and disk I/O. Our experiments using the macro-benchmarks and HPC applications assess full system performance.

We find that the Xen paravirtualizing system, in general, does not introduce significant overhead over the other OS configurations that we study – including the specialized CHAOS kernel – for almost all of the test cases. The two exceptions are for random access disk I/O (where Xen's performance degradation is significant) and bidirectional MPI network bandwidth where the performance impact is only for a small number of message sizes and is generally small. In each such case, we attempt an analysis of the cause of the overhead. 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 the flexibility and potential for enhanced security that Xen offers makes it useful in a commodity HPC context.

In the sections that follow, we first present background and motivation for the use of paravirtualized systems in HPC environments. In Section 3, we overview our experimental methodology, platform, operating systems, VMM configuration, and applications. We then present results from a large number of experiments that show the performance impact of using Xen for HPC systems and programs as compared to extant non-virtualized Linux systems. We analyze the performance characteristics of the micro- and macro-benchmarks as well as of the HPC applications. We present related work in Section 6, and our conclusions and future work in Section 7.

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) without introducing a serious performance degradation. For example, Xen supports guest-OS suspend/resume and system image migration. If it does not impose a substantial performance cost, it is possible to use this facility to implement automatic checkpoint/restart for cluster users without modifications to the Linux kernel.

OS migration is another added benefit to full-system virtualization that makes deployment and maintenance of VMM-based HPC clusters appealing. Several researchers have explored OS and process migration, such as Internet Suspend/Resume [23] and μDenali [43]. Recent studies on OS image migration [18, 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. Effective migration can be used for load balancing but also for proactive replacement of failing hardware. For example, if a hardware failure occurs, the application which was running on it has to be restarted from the last checkpoint. A proactive approach can avoid this re-execution overhead by migrating applications off of machines requiring maintenance or exhibiting behaviors indicative of potential failures (disk errors, fan speed inconsistency, etc.). Such an approach can potentially save HPC centers thousands of computational

hours and leading to higher hardware utilization rates.

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 check-pointing), 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 [24, 48, 16, 20].

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 overhead, to facilitate virtualization. This overhead however, has been the focus of much optimization effort recently. In particular, extant, performance-aware, VMMs such as Xen [31], 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 [38].

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 the one 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 used ANL implementation of message passing interface (MPI) protocol; i.e. MPICH v1.2.7p1 for establishing communications between the distributed processes on different nodes in the cluster.

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. More information on the t-test and the computation we use can be found in [25, 8].

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 in this paper 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)[1] is RHEL4 v2.6.12 with the Xen 3.0.1 patch. Above Xen, i.e. 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.

[1]The Xen host OS is commonly referred to as dom0 and the guest OS which sits above dom0 is commonly referred to as domU (U for unprivileged). We also refer to the two kernels as the host OS and the guest OS, respectively

For RHEL2.6.9, RHEL2.6.12, and CHAOS, we execute the applications without VMM (Xen) support. Only Xen employs VMM support. Figure 1 depicts these two cases, respectively.

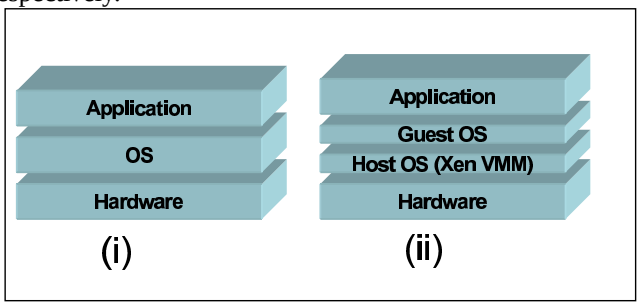

Figure 1. Software stack for our different experiments. (i) shows the traditional OS deployment running directly on the hardware; we employ this setup for CHAOS, RHEL2.6.9 and RHEL2.6.12 experiments. (ii) shows the virtualized system on which Xen executes on the hardware and RHEL2.6.12 Linux (the guest OS) runs on Xen. We refer to experiments that employ this setup simply as Xen.

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 [26] and LLNL ASCI Purple [3]. The programs are specifically designed to evaluate distinct performance characteristics of machine subsystems including MPI-based network bandwidth and latency, CPU processing, memory and disk I/O. The ASCI Purple Presta suite evaluates inter-process network latency and bandwidth for MPI message passing operations. The benchmark is written in C. We employ two of the benchmark codes to evaluate latency (Laten) and bandwidth (Com). Presta uses MPI_wtime to report the time measurements of the codes, therefore we configure the code to perform one thousand operations between calls to MPI_wtime to obtain accurate resolution.

To evaluate computational overhead, we employ the freely available Linpack benchmark [30]. Linpack is a benchmark that solves dense systems of linear equations. It is available in different languages, and parallel and serial versions. We employ serial Fortran implementation as our micro-benchmark for evaluating computational performance in isolation.

Stream is a standard memory benchmark that is part of both HPCC and LLNL ASCI Purple [40]. It reports the sustainable memory bandwidth in MB/s for four different memory operations:

$$\begin{aligned} Copy &: A(i) = B(i) \\ Scale &: A(i) = q * B(i) \\ Sum &: A(i) = B(i) + C(i) \\ Triad &: A(i) = B(i) + q * C(i) \end{aligned}$$

The first operation: *copy* reads a large array from memory, and writes it back to a different location. The three other operations 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 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, communicational and data movement characteristics of large scale computational fluid dynamics (CFD) applications.

As an example of large-scale HPC applications, we employ an application from the popular scientific simulations class of programs: the General Circulation Model (GCM) from the Massachusetts Institute of Technology. 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 Massachusetts Institute of Technology (MIT) Climate Modeling Initiative (CMI) team [29]. Researchers commonly integrate this implementation into oceanographic simulations. The MIT CMI team supports a publicly available version [2, 28], which we employ and refer to in this paper *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

	Benchmark Category	Code Name	What it measures
Micro	Communication	Presta Com	Bandwidth and OpTime versus message size
		Presta Laten	Max latency versus number of processes
	Computational	Linpack 3000d	Total Mflops and different execution time
	Memory	Stream	Rate of memory read/write in MB/s
	Disk I/O	Bonnie	sequential & Random disk input/output in MB/s
Macro	Parallel Benchmarks	NAS Parallel Benchmark; class C *Multigrid (MG) input* 512^3 *LU Solver (LU) input* 162^3 *Integer Sort (IS) input* 2^{27} *Embarrassingly parallel (EP) input* 2^{32} *Conjugate gradient (CG) input 150000*	Total time and millions of operations per second (Mops)
App	Scientific Simulations	MIT GCM exp2	Total execution time

Table 1. Benchmark Overview

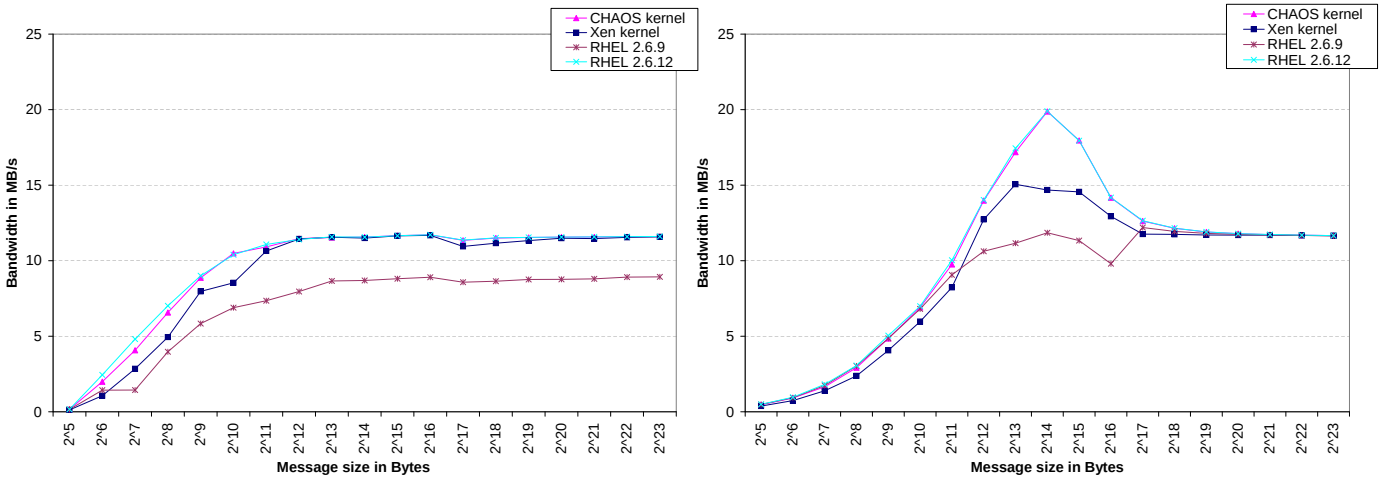

Figure 2. Com benchmark results for average network bandwidth (MB/s) for the MPI unidirectional (left graph) and the MPI bidirectional test (right graph)

circulation at a one-second resolution.

4 Micro-Benchmarks

In this section, we evaluate the performance impact of using virtualization for specific subsystems of our cluster system. We employ micro-benchmarks for network communication, computation, memory access, and disk I/O. We present and analyze the results for each of these micro-benchmarks in the following subsections.

4.1 Network Performance

We first evaluate the impact that using Xen has on network communication performance. We focus on the Message Passing Interface (MPI) for this investigation since applications commonly employ MPI to facilitate and coordinate distributed execution of the program across cluster resources. Although, applications differ in the type and amount of communication they perform [49], MPI micro-benchmark performance gives us insight into the performance overhead introduced by virtualized communication.

Our MPI micro-benchmarks are part of the LLNL ASCI Purple Presta Stress Benchmark v1.2 [35]. To investigate unidirectional and bidirectional bandwidth, we employ the `Com` benchmark. Com calculates bisectional bandwidth for unidirectional and bidirectional MPI process communication. Com outputs both bandwidth and the average time calculated for the longest operation per test. We refer to the latter as operation time (OpTime) and report these values in microseconds. Each test consists of 1000 operations and we consider 1 pair of MPI processes. We vary the message size from 2^5 to 2^{23} bytes. Our cluster system currently implements cluster connectivity via 100Mb (12.5MB/s) Ethernet.

Figure 2 shows the bandwidth attained by the different kernels for unidirectional (left graph) and bidirectional messages (right graph). The y-axis in each graph is the attained bandwidth in MB/s as a function of the message size shown along the x-axis (higher is better).

The MPI bandwidth saturates at approximately 12 MB/s equally for all kernels (except RHEL2.6.9 unidirectional MPI bandwidth) for both unidirectional and bidirectional MPI messages. RHEL2.6.9 performs significantly worse than the other three kernels for the MPI unidirectional test. This is due to a known implementation error in the TCP segmentation offload (TSO) of RHEL Linux in versions. The bug causes the driver to limit the buffer size to the maximum transmission unit (MTU) of the fabric and thus, to drop packets prematurely which results in the decreased bandwidth. This bug is fixed in the CHAOS, Xen, and RHEL v2.6.12 kernels, and thus they are not impacted by it.

Xen bandwidth for small buffer sizes is less than that achieved by CHAOS or RHEL2.6.12. This is due to the im-

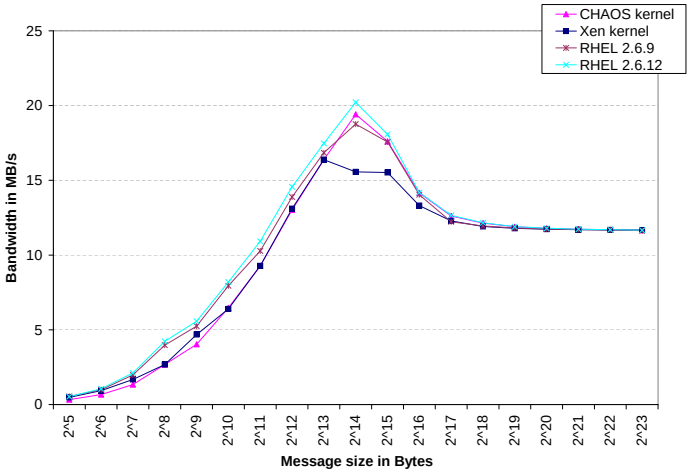

Figure 3. Com benchmark results for the maximum bandwidth (MB/s) attained by MPI bidirectional messages.

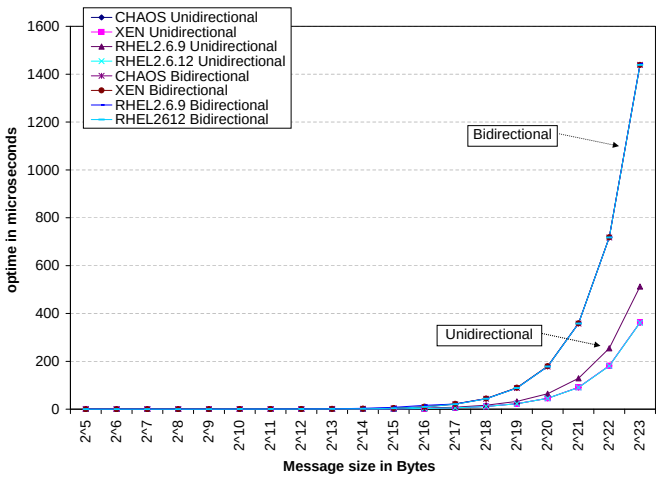

Figure 4. Com benchmark results for OpTime for both MPI unidirectional and MPI bidirectional tests. OpTime is the average time calculated for the longest bandwidth operation per test.

plementation of the network layer in Xen. Xen provides two I/O rings of buffer descriptors for each domain for network activity, one for transmit and the other for receive. To send a packet, the guest OS produces a buffer descriptor and adds it to the I/O ring. The host OS consumes the requests using a simple round-robin packet scheduler. The guest OS however, must however exchange a page frame with host OS for each received packet in order to ensure efficient packet reception. This process degrades the bandwidth achieved for small packet sends since there are a large number of guest-host interactions and heavy use of the I/O rings of buffer descriptors. Xen is able to amortize this overhead as the buffer size increases. Similarly, for the bidirectional experiments, this difference is insignificant for small packet sizes.

For the bidirectional experiments (right graph in Figure 2), CHAOS, Xen, and RHEL2.6.12 achieve hypersaturation for message sizes between 2^{12} and 2^{16}. This is due to the buffering that the kernels perform that enables overlap of communication and message processing. Xen and RHEL2.6.9 do not achieve the same benefits as CHAOS and RHEL2.6.12 on average. Figure 3 shows the maximum bandwidth achieved across tests for different message sizes. These results show that RHEL2.6.9 behaves similarly to CHAOS and RHEL2.6.12. Thus, the apparent loss of performance in the average for RHEL2.6.9 is due to greater variation rather than an absolute loss.

However, Xen bidirectional performance for message sizes 2^{14} and 2^{15} does not achieve the same maximum even in the best case, i.e., there is a true systemic difference in absolute best-case performance for these message sizes. We believe that this effect is due to the management of the dual ring buffer descriptors which reduces the effective buffer size and thus, efficacy, of kernel buffering thereby reducing the amount of overlap that Xen is able to achieve. All kernels saturate the network at the same level for message sizes greater than 2^{16}. We plan to investigate optimizations for the I/O rings and descriptor management in Xen as part of future work.

We present the OpTime for both unidirectional and bidirectional messages in Figure 4. The y-axis is the average time in microseconds for the longest operation in a test as a function of the message size on the x-axis (lower is better). The data indicates that there is no significant difference in OpTime between Xen and Chaos and RHEL2.6.12. The RHEL2.6.9 data for the unidirectional test shows a statistically significant performance degradation in OpTime for large message sizes. This is a side-effect of the lack of the TSO bug in the Ethernet driver as we described previously.

We next evaluate network latency using the Presta Laten benchmark from the ASCI Purple suite. Laten calculates the maximum latency for a test (1000 operations) between pairs of MPI processes as the elapsed time in a ping-pong communication. In this benchmark we vary the number of simultaneously communicating processes.

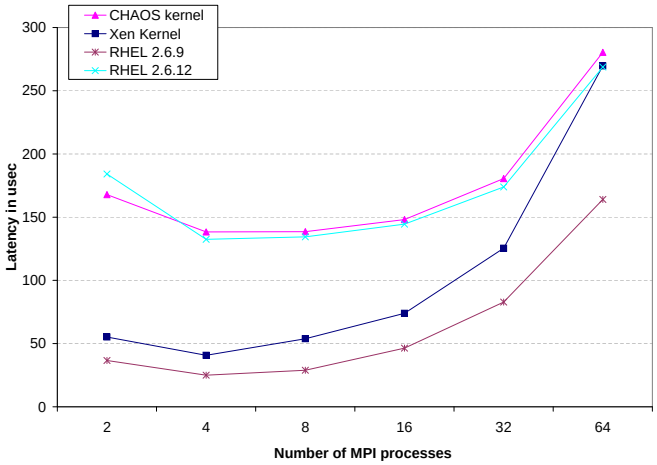

Figure 5. Laten MPI bidirectional results in microseconds.

Figure 5 shows the results for the four kernels. The y-axis is the average of maximum latency in microseconds between per test as a function of the number of processes shown on the x-axis (lower is better).

Although it is counter-intuitive, Xen has lower latency for up to 32 MPI communicating processes than CHAOS and RHEL2.6.12. This is a result of the use of *page-flipping* in Xen that optimizes data transfer by avoiding copying between the guest OS and the host OS. However, as the number of processes increases, the overhead of Xen's I/O rings of buffer descriptors has a larger impact that the optimization cannot amortize to the same degree.

RHEL2.6.9 enables the lowest latency. This behavior depicts an interesting effect of the TSO bug described earlier. The bug causes RHEL2.6.9 to achieve lower bandwidth than the other kernels but also to introduce less overhead for individual sends that do not require buffering. Therefore, kernels prior to v2.6.11 impose lower latency.

4.2 Computational Performance

HPC are performance-critical systems. The computational performance is undoubtedly one of the most important factors -if not the most important- in characterizing the efficiency of the HPC system. Therefore, we also evaluate the computational performance of the paravirtualized system in comparison with the non-virtualized kernels.

We use Linpack [14] LU decomposition for this study. The Linpack LU decomposition process consists of two phases: factoring and back-solve. The benchmark reports the time taken in each phase and the rate of floating point operations in mflops. We also measure the total time using the Linux time utility. Our input to Linpack is a matrix with 3000x3000 in double-precision values.

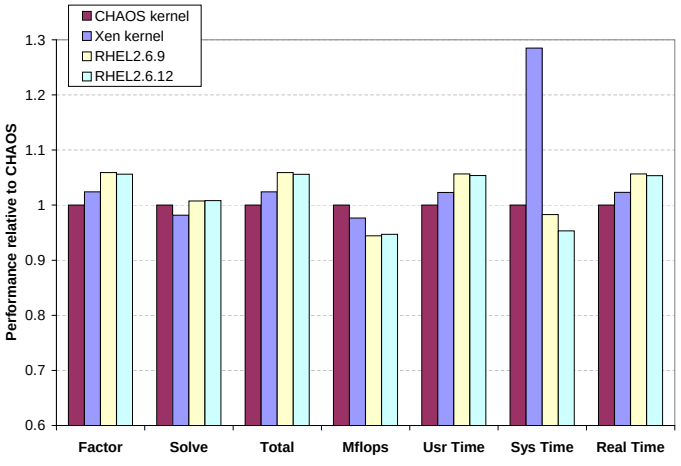

Figure 6. Linpack LU decomposition 3000d performance relative to CHAOS. Lower is better for metrics factor, solve, total, user, system, and real time. Higher is better for Mflops.

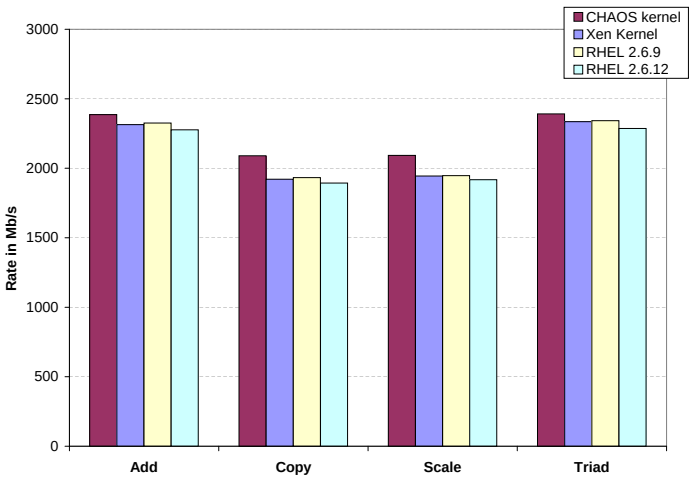

Figure 7. Stream memory bandwidth performance in Mb/s.

Figure 6 illustrates a Linpack performance comparison between the four kernels. The y-axis is the performance of the different kernels relative to the CHAOS kernel with respect to the different metrics on the x-axis. The smaller the time ratio, the better but the higher the Mflops ratio is the better.

The comparison indicates that Xen is slower than CHAOS kernel for the factoring phase and in terms of the total time. However, Xen is faster than the other kernels. Furthermore, the t-values for these differences show statistical significance for these differences even at the 0.999 confidence level.

Most of the difference occurs during the factoring phase of LU. The Xen kernel does appear to have a shorter backsolve time than three other kernels, but the t-values do not indicate statistical significance at the 0.95 confidence level. On the other hand, Xen achieves a total Mflops rate that is approximately 2% lower than CHAOS kernels and 3% higher than the RHEL kernels. The reason behind better Mflops performance for Xen is due to its CPU scheduling process: a very efficient implementation of the borrowed virtual time (BVT) scheduler [15]. BVT and the overhead of scheduling in general positively impacts the Mflops rate of Xen-based Linpack. CHAOS scheduler optimizations enable additional performance improvements. As a result, a Xen-based CHAOS implementation (that we are building as part of future work) should be able to achieve benefits similar to those for CHAOS reported here.

For these reasons, Xen improves user time (and thus total time) by approximately 3% over the RHEL kernels and achieve user/total time that is lower by 2% than CHAOS. The system time bar in the graph for Xen is an anomaly that is due to a known bug in they way Xen computes system time – this value is invalid but we report it to enable others to validate our exact results using this version of Xen. The bug will be fixed in the next release of Xen.

4.3 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 [40], which we configure with the default array size of 2 million elements.

Figure 7 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.

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 behave 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.

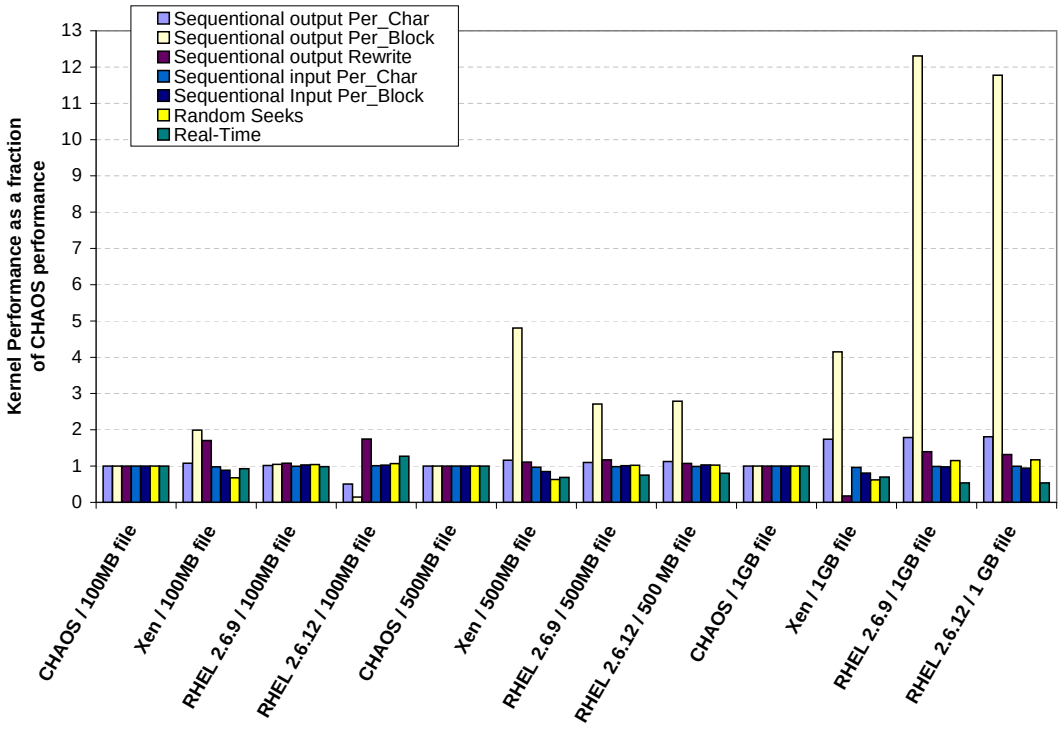

Figure 8. Bonnie Disk I/O bandwidth rate and Real-time relative to CHAOS performance

4.4 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 8 shows the performance comparison for the four kernels relative to the performance of CHAOS. The x-axis is the performance of the different disk I/O metrics relative to the performance of CHAOS, 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 (Real-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 [31]. 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.

The sequential input per character among all kernels is not significantly different. However, for sequential input per block, Xen disk I/O speed lags behind the other three kernels by about 11-17%. We suspect this is a configuration problem and we will work to verify this conjecture for the final version of this paper, should it be accepted.

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 of 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 9 shows the performance of the NPB codes(x-axis) for our different kernels relative to CHAOS (y-axis). 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 RHEL2.6.12 for both total time and Mops. The reason for this is similar to that for the Linpack results in the previous section due to overhead this kernel places on computation (c.f. Section 4.2). CHAOS optimizes this overhead away and RHEL2.6.9 makes up for this loss due to its low overhead on MPI-based network latency (c.f. Section 4.1). A combination of the scheduling policy and network performance enabled by Xen enables the Xen system to avoid the overhead also.

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,

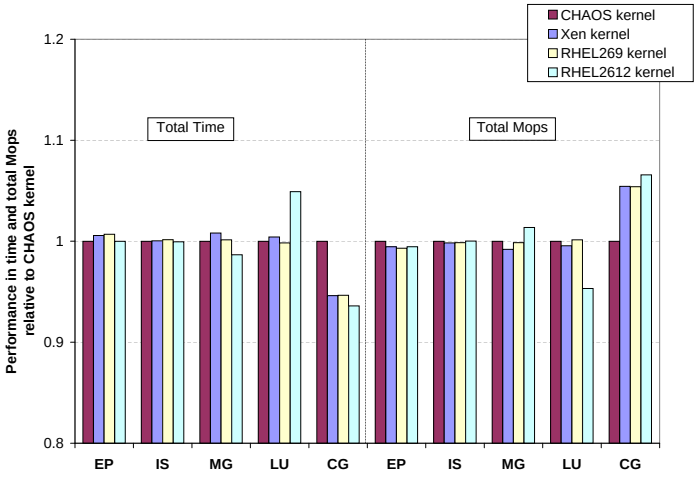

Figure 9. NAS Parallel Benchmark performance relative to CHAOS. The left half (first benchmark set) is for total time (lower is better); the right half is for Mops (higher is better).

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 10. The y-axis is the time in seconds for the kernels shown on the x-axis.

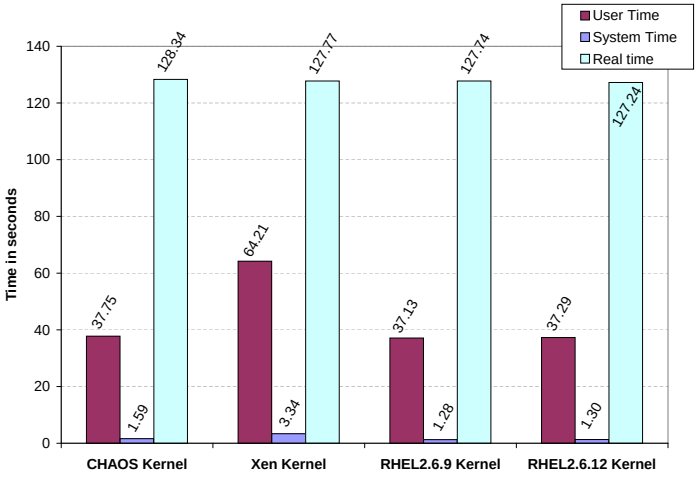

Figure 10. MIT GCM performance using input Exp2 in seconds (lower is better).

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 as mentioned previously). 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, include performance studies of virtualization-based systems. To our knowledge, our study is the first to investigate the performance implications of using paravirtualization technologies in an HPC setting. 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 has investigated the performance of Xen and other similar technologies in an non-HPC setting. The most popular performance evaluation of Xen is described in [31]. A similar, yet independent but concurrent, study is described in [11]. Both papers show 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, 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.

Students as part of an unpublished, class project at the Norwegian University of Science and Technology (NTNU) have investigated Xen performance for clusters [19]. This study investigates the network communication performance in Xen versus a native kernel using low-level and application-level network communication benchmarks. The resulting Master's Thesis [7] describes a port of Xen to IA64 but provides only a minimal evaluation. On the other hand, another study [39] done at Wayne State University investigated the communication performance for different network switch fabric on Linux clusters. They evaluated performance of Fast Ethernet using ch_p4 interface, Gigabit Ethernet using ch_p4 interface, Myrinet using ch_p4 interface, and Myrinet using ch_gm interface. Based on that study results, we anticipate that Xen would perform on Fast Ethernet and Myrinet using ch_p4 similar to how it did perform on Gigabit Ethernet in our study. However, It would be interesting to see how Xen page-flipping algorithm, described earlier interact with Myrinet's OS-bypass features.

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 msecconds. Related tools have been developed to Xen VMM, as in `Jisha` [22] and `Xen-Get` [44]. These systems do not rigorously investigate the performance overheads of doing so in an HPC setting.

7 Conclusions and Future Work

Paravirtualizing systems expose unique and exciting opportunities to the HPC community in the form of flexible system maintenance, management, and customization. Such systems however, are currently not considered for HPC environments since they are perceived to impose overhead that is unacceptable for performance-critical applications and systems. In this paper, we present a rigorous empirical evaluation of using Xen paravirtualization for HPC

applications, kernels, and systems that shows that such concern is unwarranted.

We compare three different Linux configurations with 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 – except in 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. The latter is very high-performance and successful networking technology for HPC applications. LLNL's cluster implementation will soon be extended to make use of this technology and our goal is to optimize Xen communication for the Infiniband hardware. We plan to port Xen to the latest version of CHAOS v2.6.12 when this version becomes available.

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 OS installation over Xen [22] and for static and dynamic specialization of OS images in a way that is application-specific [24, 48].

8 Acknowledgements

The authors owe great thanks to Makia Munich at LLNL and Graziano Obertelli at UCSB for their continuous work and help with the CHAOS system and the benchmarks. We would also like to extend our thanks to Xen-community, for their prompt response to our questions through the Xen mailing lists.

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 asci 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] J. Dongarra. The linpack benchmark: An explanation. In *Proceedings of the 1st International Conference on Supercomputing*, pages 456–474, London, UK, 1988. Springer-Verlag.

[15] 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.

[16] 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/.

[17] 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.

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

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

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

[21] 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.

[22] Jisha:Guest OS Deployment Tool for Xen. http://cs.ucsb.edu/~ahuda/jisha/.

[23] 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.

[24] 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.

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

[26] 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.pd%f.

[27] 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.

[28] 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.

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

[30] "Netlib Repository at UTK and ORNL". http://www.netlib.org/.

[31] 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/.

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

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

[34] 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.

[35] Presta stress benchmark code. http://www.llnl.gov/asci/purple/benchmarks/limited/presta/.

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

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

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

[39] 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.

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

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

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

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

[44] Xen-Get. http://www.xen-get.org/.

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

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

[47] XenSource. http://www.xensource.com/.

[48] 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.

[49] R. Zamani and A. Afsahi. Communication characteristics of message-passing scientific and engineering applications. In *17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2005), Phoenix, AZ, USA*, pages 644–649, "November" 2005.