Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems[*]

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 for high-end server systems. Virtualizing software has the potential to benefit HPC systems similarly by facilitating efficient cluster management, application isolation, full-system customization, and process migration. However, virtualizing software is not currently employed in HPC environments due to its perceived overhead.

In this work, we investigate the overhead imposed by the popular, open-source, Xen virtualization system, on performance-critical HPC kernels and applications. We empirically evaluate the impact of Xen on both communication and computation and compare its use to that of a customized kernel using HPC cluster resources at Lawrence Livermore National Lab (LLNL). 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.

1 Introduction

Virtualization is a widely used technique in which a software layer multiplexes lower-level resources among higher-level applications and system software. Examples of virtualization systems include a vast body of work in the area of operating systems [27, 25, 21, 24, 3, 13], 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, and support for multiple or customized operating systems.

This added flexibility however, can potentially introduce significant execution overhead due to the extra level of indirection and interference between the hardware and application. Recent VMM optimizations and advances however, attempt to reduce this overhead to zero. 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. Paravirtualized 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 [1, 32, 23, 16].

Despite the flexibility benefits, performance advances, and recent research indicating its potential [18, 35, 12, 15], 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 – even highly optimized VMMs – is still 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, specifically for the Linux operating system and the Xen [23] paravirtualizing VMM.

Xen is an open-source virtual machine monitor for the Linux operating system which reports low-overhead and efficient execution of Linux [31]. 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 [23, 33] 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 have collected performance data using micro- and macro-benchmarks from the HPC Challenge, LLNL ASC Purple, and NAS parallel benchmark suites among others, as well as using a large-scale, HPC application for simulation of oceanographic and climatologic phenomena (c.f. [34] for complete details). However, due to space limitations, in this paper we focus only on the performance evaluation of the paravirtualized computational and communications subsystems, including MPI-based network bandwidth and latency, and CPU processing.

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 one exception is for the bidirectional MPI network bandwidth where the performance impact is only for a small number of message sizes and is generally small. 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.

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 our results (Section 4), the related work (Section 5), and our conclusions and future work (Section 6).

2 Background and Motivation

Our investigation into the performance implications of paravirtualization for high performance computing (HPC) systems stems from the need to improve the flexibility of large-scale HPC clusters at Lawrence Livermore National Laboratory (LLNL) without introducing serious performance degradations. For example, Xen currently supports guest-OS suspend/resume and system image migration. If it does not impose a substantial performance cost, we believe it is possible to use this facility to implement automatic checkpoint/restart for cluster users without modifications to the Linux kernel. The resulting system, then, implements an important functionality with no impact on user programming effort and without the maintenance difficulties associated with the use of non-standard operating system kernels.

OS migration is an added benefit to full-system virtualization that makes deployment and maintenance of VMM-based HPC clusters appealing. Not only can effective migration be used for load balancing, but also it can be used for proactive replacement of failing hardware. For example, in the current system deployment at HPC centers, if a hardware failure occurs, the application which was running on it normally 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. 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 customized operating systems [18, 35, 12, 15] that are optimized for, and tailored to the needs of individual applications.

Though many of the benefits of virtualization are well known, the perceived cost of virtualization is what makes it of questionable use to the HPC community, where performance is critical. This overhead however, has been the focus of much optimization effort recently. In particular, extant, performance-aware, VMMs such as Xen [23], 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. No application code

must be changed to execute using a paravirtualizing system such as Xen. A more detailed overview of system-level virtual machines, and paravirtualization can be found in [28].

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. We investigate the effects of paravirtualization on the two most critical application performance components – interprocess communication and per-processor execution speed – as a way of gauging whether a more full-scale engineering effort to deploy paravirtualization at LLNL should be mounted.

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's 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 formulation we use can be found in [19, 5].

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 [9, 6] 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. For RHEL2.6.9, RHEL2.6.12, and CHAOS, we execute the applications without VMM (Xen) support. Only Xen employs VMM support.

3.2 Benchmarks

We have done an extensive performance evaluation for the different subsystems using a wide range of standard benchmarks for the communications, computations, Disk I/O and memory performance. Furthermore, we have used macro-benchmarks and real HPC applications to evaluate the overall performance of the paravirtualized system. However, we opt to include only the communication and computational subsystems performance evaluations in this workshop paper, due to space limitations. The comprehensive set of results are included in a thorough technical report [34].

Our micro-benchmark set includes programs from the HPC Challenge [20] and LLNL ASC Purple [2]. The programs are specifically designed to evaluate distinct performance characteristics of machine subsystems including MPI-based network bandwidth and latency, and CPU processing. The ASC Purple Presta suite evaluates interprocess 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.

---

[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
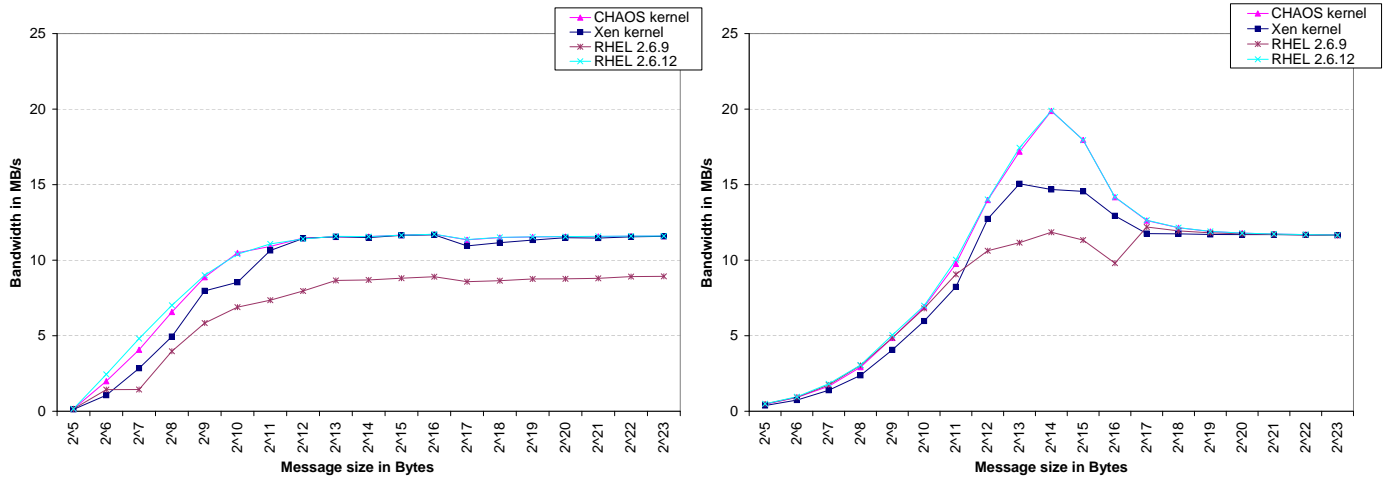
Figure 1. Com benchmark results for average network bandwidth (MB/s) for the MPI unidirectional (left graph) and the MPI bidirectional test (right graph)

To evaluate computational overhead, we employ the freely available Linpack benchmark [22]. Linpack is a library that solves dense systems of linear equations. A benchmark based on the library, which is available in both C and Fortran, and parallel and serial versions is deployed in this study. We employ the serial Fortran implementation as our micro-benchmark for evaluating computational performance in isolation of other factors.

4 Benchmarks Results

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 and computation. We present and analyze the results for each of these micro-benchmarks in the following subsections.

4.1 Network Communication Performance

We first evaluate the impact 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 [36], MPI micro-benchmark performance gives us insight into the performance overhead introduced by virtualized communication.

Our MPI micro-benchmarks are part of the LLNL ASC Purple Presta Stress Benchmark v1.2 [26]. 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 1000Mb (12.5MB/s) Ethernet.

Figure 1 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.

Using MPI, a user code saturates the available bandwidth 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 prior to 2.6.11. 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 in the unidirectional MPI test. This is due to the implementation 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
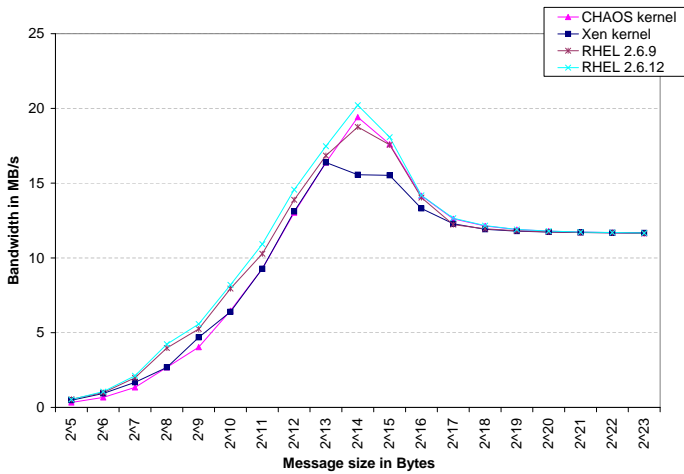
Figure 2. Com benchmark results for the maximum bandwidth (MB/s) attained by MPI bidirectional messages.
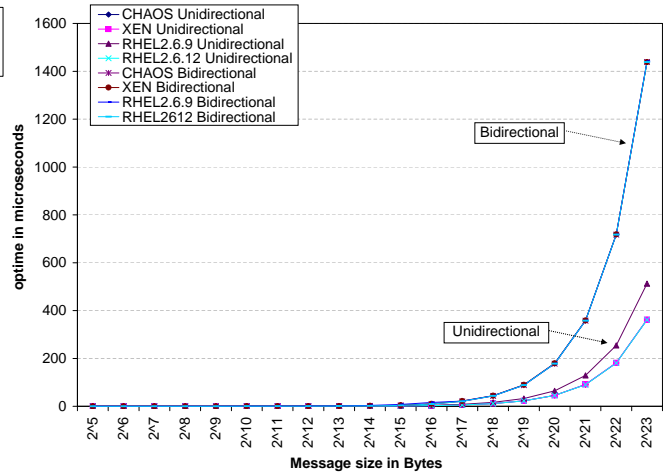

Figure 3. Com benchmark results for OpTime for both MPI unidirectional and MPI bidirectional tests. OpTime is the average time calculated for the longest bandwidth operation per test.

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 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 1), CHAOS, and RHEL2.6.12 achieve hypersaturation for message sizes between 2^{12} and 2^{16}. This is due to the buffering which 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 2 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 which we see in figure 1 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 3. 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 presence of the TSO bug in the Ethernet driver as we described previously.

We next evaluate network latency using the Presta Laten benchmark from the ASC 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 4 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 on the performance
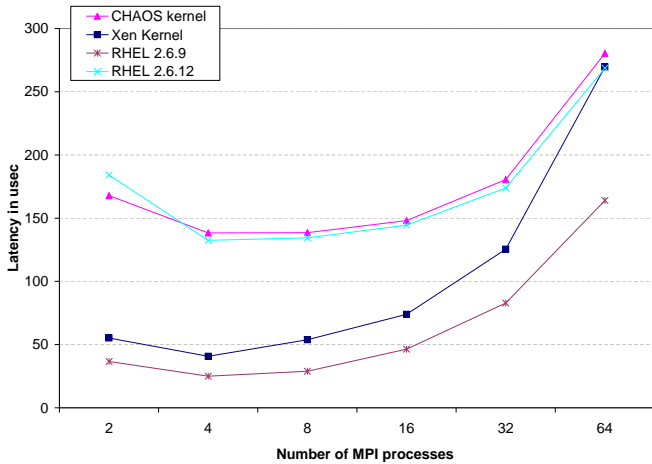
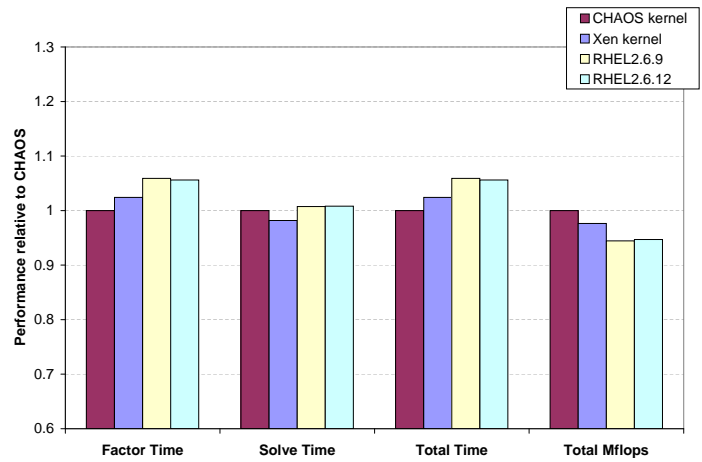**Figure 4. Laten MPI bidirectional results in microseconds.**

Figure 5. Linpack LU decomposition 3000d performance relative to CHAOS. Lower is better for metrics factor, solve and total times. Higher is better for Mflops.

which 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.

4.2   Computational Performance

HPC systems 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 [10] 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. Our input to Linpack is a matrix with 3000x3000 in double-precision values.

Figure 5 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 two other RHEL kernels in the factoring time and the total time. Furthermore, the t-values for these differences show statistical significance for these differences even at the 0.999 confidence level, but not between Xen kernel and CHAOS kernel .

Most of the difference occurs during the factoring phase

of LU. In addition, the Xen kernel does appear to have a shorter back-solve time than the three other kernels, but the t-values do not any 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 [11]. 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.

5   Related Research

The work related to that which we pursue in this paper, included performance studies of virtualization-based systems. In this paper, we investigated the communications and computational performance of HPC benchmarks. We consider both subsystems' performance for a number of important HPC components when using paravirtualizing systems for HPC cluster resources (IA64, SMP machines).

Other work has investigated the performance of Xen and other similar technologies in a non-HPC setting. The most popular performance evaluation of Xen is described in [23]. A similar, yet independent but concurrent, study is described in [7]. Both papers show the efficacy and low overhead of paravirtualizing systems. The benchmarks that both papers employ are general-purpose operating sys-

tems 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 as of this writing (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 [14]. 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 [4] describes a port of Xen to IA64 but provides only a minimal evaluation.

On the other hand, another study [29] 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 [8]. They show that live migration can be done with no performance cost, and with down times as low as 60 mseconds. In addition, related tools have been developed to deploy guest OS on Xen VMM, as in `Jisha` [17] and `Xen-Get` [30]. These systems do not rigorously investigate the performance overheads of doing so in an HPC setting.

6 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 view into a empirical evaluation of using Xen paravirtualization for two HPC-specific subsystems, and HPC kernels 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 and other sites of 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-benchmarks from LLNL ASC Purple and HPC Challenge benchmark suites. As a result, we are able to rigorously evaluate the performance of Xen-based HPC systems relative to non-virtualized system for two subsystems: computational and communications.

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 one instance. 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 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 [17] and for static and dynamic specialization of OS images in a way that is application-specific [18, 35].

7 Acknowledgements

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

[3] 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.

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

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

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

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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/`.

[13] 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.

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

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

[16] 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.

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

[18] 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.

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

[20] 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`.

[21] 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.

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

[23] 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/`.

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

[25] 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.

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

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

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

[29] 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.

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

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

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

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

[34] 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.

[35] 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.

[36] 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.