A Comparison of Virtualization Technologies for HPC

Conference Paper · April 2008

DOI: 10.1109/AINA.2008.45 · Source: IEEE Xplore

Vishwamitra Chaudhary
Dr.M.C.Saxena coll. of engg. & tech.,lko
76 PUBLICATIONS **4,144** CITATIONS
SEE PROFILE

John Paul Walters
University of Southern California
48 PUBLICATIONS **928** CITATIONS
SEE PROFILE

Steven M Gallo
University at Buffalo, The State University of N…
39 PUBLICATIONS **1,005** CITATIONS
SEE PROFILE

A Comparison of Virtualization Technologies for HPC*

J. P. Walters, Vipin Chaudhary, and Minsuk Cha
University at Buffalo, SUNY
Computer Science and Engineering
{waltersj, vipin, mcha}@buffalo.edu

Salvatore Guercio Jr. and Steve Gallo
The Center for Computational Research
University at Buffalo, SUNY
{sguercio, smgallo}@ccr.buffalo.edu

Abstract

Virtualization is a common strategy for improving the utilization of existing computing resources, particularly within data centers. However, its use for high performance computing (HPC) applications is currently limited despite its potential for both improving resource utilization as well as providing resource guarantees to its users. This paper systematically evaluates various VMs for computationally intensive HPC applications using various standard benchmarks. Using VMWare Server, Xen, and OpenVZ we examine the suitability of full virtualization, paravirtualization, and operating system-level virtualization in terms of network utilization, SMP performance, file system performance, and MPI scalability. We show that the operating system-level virtualization provided by OpenVZ provides the best overall performance, particularly for MPI scalability.

1 Introduction

The use of virtualization in computing is a well-established idea dating back more than 30 years [1]. Traditionally, its use has meant accepting a sizable performance reduction in exchange for the convenience of the virtual machine. Now, however, the performance penalties have been reduced. Faster processors as well as more efficient virtualization solutions now allow even modest desktop computers to host virtual machines.

Soon large computational clusters will be leveraging the benefits of virtualization in order to enhance the utility of the cluster as well as to ease the burden of administering such large numbers of machines. Indeed, virtual machines allow administrators to more accurately control their resources while simultaneously protecting the host node from malfunctioning user-software. This allows administrators to provide "sandbox-like" environments with little performance reduction from the user's perspective.

In light of the benefits to virtualization, we believe that virtual machine/virtual server-based computational clusters will soon gain widespread adoption in the high performance computing (HPC) community. However, to date, a comprehensive examination of the various virtualization strategies and implementations has not been conducted, particularly with an eye towards its use in HPC environments. We fill this literature gap by examining multiple aspects of cluster performance with standard HPC benchmarks. In so doing we make the following two contributions:

1. **Single Server Evaluation:** In Section 3 we evaluate several virtualization solutions for single node performance and scalability. Rather than repeat existing research, we focus our tests on industry-standard scientific benchmarks including SMP (symmetric multiprocessor) tests through the use of OpenMP (open multiprocessing) implementations of the NAS Parallel Benchmarks (NPB) [2]. We examine file system and network performance (using IOZone [3] and Netperf [4]) in the absence of MPI (message passing interface) benchmarks in order to gain insight into the potential performance bottlenecks that may impact distributed computations.

2. **Cluster Evaluation:** We extend our evaluation to the cluster-level and benchmark the virtualization solutions using the MPI implementation of NPB. Drawing on the results from our single node tests, we consider the effectiveness of each virtualization strategy by examining the overall performance demonstrated through industry-standard scientific benchmarks.

2 Existing Virtualization Technologies

To accurately characterize the performance of different virtualization technologies we begin with an overview of the major virtualization strategies that are in common use

*This research was supported in part by NSF IGERT grant 9987598, the Institute for Scientific Computing at Wayne State University, MEDC/Michigan Life Science Corridor, and NYSTAR.

for production computing environments. In general, most virtualization strategies fall into one of four major categories:

1. **Full Virtualization:** Also sometimes called hardware emulation. In this case an unmodified operating system is run using a hypervisor to trap and safely translate/execute privileged instructions on-the-fly. Because trapping the privileged instructions can lead to significant performance penalties, novel strategies are used to aggregate multiple instructions and translate them together. Other enhancements, such as binary translation, can further improve performance by reducing the need to translate these instructions in the future [5, 6].

2. **Paravirtualization:** Like full virtualization, paravirtualization also uses a hypervisor, and also uses the term virtual machine to refer to its virtualized operating systems. However, unlike full virtualization, paravirtualization requires changes to the virtualized operating system. This allows the VM to coordinate with the hypervisor, reducing the use of the privileged instructions that are typically responsible for the major performance penalties in full virtualization. The advantage is that paravirtualized virtual machines typically outperform fully virtualized virtual machines. The disadvantage, however, is the need to modify the paravirtualized virtual machine/operating system to be hypervisor-aware. This has implications for operating systems without available source code.

3. **Operating System-level Virtualization:** The most intrusive form of virtualization is operating system-level virtualization. Unlike both paravirtualization and full virtualization, operating system-level virtualization does not rely on a hypervisor. Instead, the operating system is modified to securely isolate multiple *instances* of an operating system within a single host machine. The guest operating system instances are often referred to as virtual private servers (VPS). The advantage to operating system-level virtualization lies mainly in performance. No hypervisor/instruction trapping is necessary. This typically results in system performance of near-native speeds. The primary disadvantage is that all VPS instances share a single kernel. Thus, if the kernel crashes or is compromised, all VPS instances are compromised. However, the advantage to having a single kernel instance is that fewer resources are consumed due to the operating system overhead of multiple kernels.

4. **Native Virtualization:** Native virtualization leverages hardware support for virtualization within a processor itself to aid in the virtualization effort. It allows multiple unmodified operating systems to run alongside one another, provided that all operating systems are capable of running on the host processor directly. That is, native virtualization does not emulate a processor. This is unlike the full virtualization technique where it is possible to run an operating system on a fictional processor, though typically with poor performance. In x86_64 series processors, both Intel and AMD support virtualization through the Intel-VT and AMD-V virtualization extensions. x86_64 Processors with virtualization support are relatively recent, but are fast-becoming widespread.

For the remainder of this paper we use the word "guest" to refer to the virtualized operating system utilized within any of the above virtualization strategies. Therefore a guest can refer to a VPS (OS-level virtualization), or a VM (full virtualization, paravirtualization).

In order to evaluate the viability of the different virtualization technologies, we compare VMWare Server version 1.0.2[1], Xen version 3.0.4.1, and OpenVZ based on kernel version 2.6.16. These choices allow us to compare full virtualization, paravirtualization, and OS-level virtualization for their use in HPC scenarios. We do not include a comparison of native virtualization in our evaluation as the existing literature has already shown native virtualization to be comparable to VMWare's freely available VMWare Player in software mode [7].

2.1 Overview of Test Virtualization Implementations

Before our evaluation we first provide a brief overview of the three virtualization technologies that we will be testing: VMWare Server [8], Xen [9], and OpenVZ [10].

VMWare is currently the market leader in virtualization technology. We chose to evaluate the free VMWare Server product, which includes support for both full virtualization and native virtualization, as well as limited (2 CPU) virtual SMP support. Unlike VMWare ESX Server, VMWare Server (formerly GSX Server) operates on top of either the Linux or Windows operating systems. The advantage to this approach is a user's ability to use additional hardware that is supported by either Linux or Windows, but is not supported by the bare-metal ESX Server operating system (SATA hard disk support is notably missing from ESX Server). The disadvantage is the greater overhead from the base operating system, and consequently the potential for less efficient resource utilization.

[1]We had hoped to test VMWare ESX Server, but hardware incompatibilities prevented us from doing so.

VMWare Server supports three types of networking: bridged networking, NAT networking, and host-only networking. Bridged networking allows multiple virtual machines to act as if they are each distinct hosts, with each virtual machine being assigned its own IP address. NAT networking allows one or more virtual machines to communicate over the same IP address. Host-only networking can be used to allow the virtual machine to communicate directly with the host without the need for a true network interface. Bridged networking was used for all of our experiments to follow.

Xen is the most popular paravirtualization implementation in use today. Because of the paravirtualization, guests exist as independent operating systems. The guests typically exhibit minimal performance overhead, approximating near-native performance. Resource management exists primarily in the form of memory allocation, and CPU allocation. Xen file storage can exist as either a single file on the host file system (file backed storage), or in the form of partitions or logical volumes.

Xen networking is completely virtualized (excepting the Infiniband work done by Liu, et al. [11]). A series of virtual ethernet devices are created on the host system which ultimately function as the endpoints of network interfaces in the guests. Upon instantiating a guest, one of the virtual ethernet devices is used as the endpoint to a newly created "connected virtual ethernet interface" with one end residing on the host and another in the guest. The guest sees its endpoint(s) as standard ethernet devices (e.g. "eth0"). Each virtual ethernet devices is also given a MAC address. Bridging is used on the host to allow all guests to appear as individual servers.

OpenVZ is the open source version of SWsoft's Linux Virtuozzo product. It uses operating system-level virtualization to achieve near native performance for operating system guests. Because of its integration with the Linux kernel, OpenVZ is able to achieve a level of granularity in resource control that full virtualization and paravirtualization cannot. Indeed OpenVZ is able to limit the size of an individual guest's communication buffer sizes (e.g. TCP send and receive buffers) as well as kernel memory, memory pages, and disk space down to the inode level. Adjustments can only be made by the host system, meaning an administrator of a guest operating system cannot change his resource constraints.

OpenVZ fully virtualizes its network subsystem and allows users to choose between using a virtual network device, or a virtual ethernet device. The default virtual network device is the fastest, but does not allow a guest administrator to manipulate the network configuration. The virtual ethernet device is configurable by a guest administrator and acts like a standard ethernet device. Using the virtual network device, all guests are securely isolated (in terms of network traffic). Our tests were performed using the default virtual network device.

3 Performance Results

We now present the results of our performance analysis. We benchmarked each system (Xen, OpenVZ, and VMWare Server) against a base x86 Fedora Core 5 install. All tests were performed on a cluster of 64 dedicated Dell PowerEdge SC1425 servers consisting of 2x3.2 GHz Intel Xeon processors, gigabit ethernet, 2 GB RAM, and an 80 GB 7200 RPM SATA hard disk.

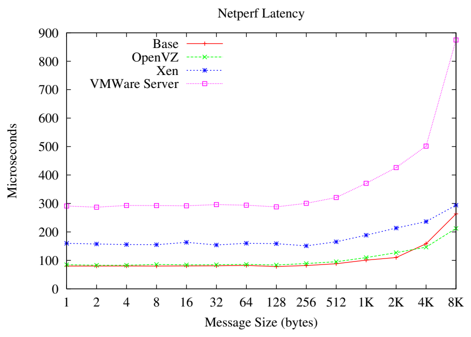
In addition, nodes are connected through a pair of Force10 E1200 switches. The E1200 switches are fully non-blocking gigabit ethernet using 48port copper line cards. To maintain consistency, each guest consisted of a minimal install of Fedora Core 5 with full access to both CPUs. The base system and VMWare Server installs used a 2.6.15 series RedHat kernel. Both OpenVZ and Xen used the most recent 2.6.16 series kernels. All guest operating systems were allotted 1650 MB RAM, leaving 350 MB for the host operating system. This allowed all benchmarks to run comfortably within the guest without any swapping, while leaving adequate resources for the host operating system as well. All unnecessary services were disabled in order to maximize the guest's resources.

Each system was tested for network performance using Netperf [4], and file system read and write performance using IOZone [3]. These tests served as microbenchmarks, and proved useful (particularly the network benchmarks) in analyzing the scalability and performance of the distributed benchmarks. Our primary computational benchmarks were the NAS Parallel Benchmark suite [2]. We tested both serial, parallel (OpenMP), and MPI versions of the NPB kernels. All guests were instantiated with a standard install, and all performance measurements were obtained with "out-of-the-box" installations.

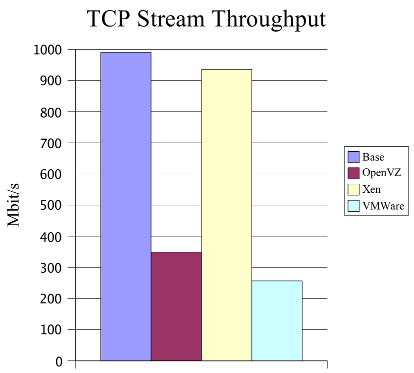
3.1 Network Performance

Using the Netperf [4] network benchmark tool, we tested both the network throughput and latency of each virtualization strategy and compared it against the native results. All tests were performed multiple times and their results were averaged. We measured latency using Netperf's TCP Request/Response test with increasing message sizes. The latency shown is the half-roundtrip latency.

Examining Figure 1(b) we can see that Xen clearly outperforms both OpenVZ and VMWare Server in network bandwidth and is able to utilize 94.5% of the base system's network bandwidth. OpenVZ and VMWare Server, however, are able to achieve only 35.3% and 25.9%, respectively, of the native bandwidth.



(a) Latency



(b) Throughput

Figure 1. Comparison of network performance.

Examining Figure 1(a), however, tells a different story. While Xen was able to achieve near-native performance in bulk data transfer, it demonstrates exceptionally high latency. OpenVZ, however, closely matches the base latency with an average 1-byte one-way latency of 84.62 μs compared to the base latency of 80.0 μs. This represents a difference of only 5.8%. Xen, however, exhibits a 1-byte one-way latency of 159.89 μs, approximately twice that of the base measurement. This tells us that, while Xen may perform exceptionally well in applications that move large amounts of bulk data, it is unlikely to outperform OpenVZ on applications that require low-latency network traffic. Again VMWare demonstrates the greatest virtualization overhead with latencies of more than three times those of the base system.

3.2 File System Performance

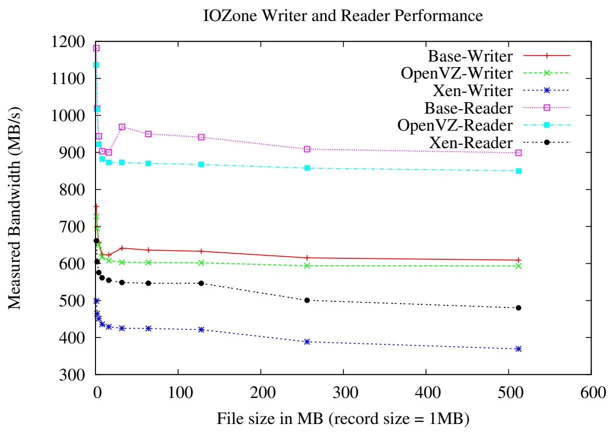


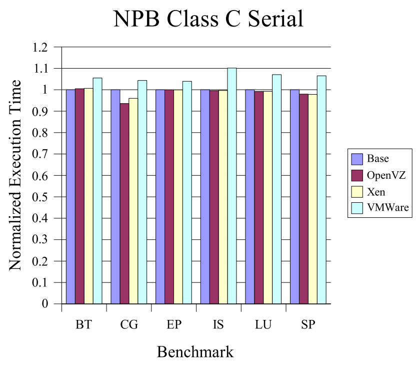
Figure 2. IOZone file system performance.

We tested each guest's file system using the IOZone [3] file system benchmark. We fixed the record size at 1 MB and measured the bandwidth for varying file sizes between 1 MB and 512 MB. While, in each case, the full complement of IOZone tests were run, space does not allow us to show all results. We chose the reader and writer tests as our representative benchmarks as they accurately demonstrated the types of overhead found within each virtualization technology. The reader test measures the performance of reading an existing file and the writer test measures the performance of writing a new file. We omit the results of the VMWare Server IOZone tests as incompatibilities with the serial ATA disk controller required the use of file-backed virtual disks rather than LVM-backed or partition-backed virtual disks. It is well known that file-backed virtual disks suffer from exceptionally poor performance.

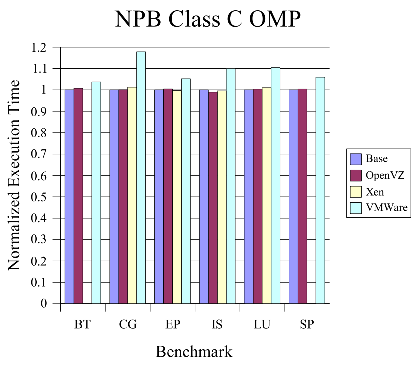
In Figure 2 we show the results of both the reader and writer tests for the base system, OpenVZ, and Xen. The results show that OpenVZ achieves near native performance of file system operations when compared to the base system, with a maximum overhead of approximately 10% in the worst case (32 MB file sizes in the reader test). The OpenVZ guests reside within the file system of the host as a directory. Xen, which uses Logical Volume Management (LVM) for guest storage, exhibits lower performance than either OpenVZ or the base system with overheads of up to 46% and 40% in the reader and writer tests, respectively. This is due to the overhead of Xen's virtualized IO subsystem.

3.3 Single Node Benchmarks

While our primary objective is to test the performance and scalability of VMWare Server, Xen, and OpenVZ for distributed HPC applications we first show the baseline performance of NAS Parallel Benchmarks [2] on a single node using both the serial and OpenMP benchmarks from NPB 3.2. Some of the benchmarks (namely MG and FT) were



(a)



(b)

Figure 3. Relative execution time of NPB serial and parallel (OpenMP) benchmarks.

excluded due to their memory requirements.

The results of the serial and parallel NPB tests are shown in Figure 3. We normalize the result of each test to a fraction of the native performance in order to maintain a consistent scale between benchmarks with differing run times. In Figure 3(a) we see that the class C serial results nearly match the baseline native performance. Even the fully-virtualized VMWare Server demonstrates performance that is consistently within 10% of the normalized native run time.

The most problematic benchmark for VMWare Server, as shown in Figures 3(a), is the IS (integer sort) kernel. Indeed the IS kernel is the only benchmark that exhibits a relative execution time that is more than 10% slower than the native time. Because of the normalized execution times shown in Figure 3 the actual time component of the benchmark is removed. However, IS exhibits an exceptionally short run time for the class C problem size. Thus, the small amount of overhead is magnified due to the short run times of the IS kernel.

Most encouraging, however, is that we see no meaningful performance penalty in using either Xen or OpenVZ. Even the IS kernel exhibits near-native performance. This suggests that the CPU-bound overhead of both paravirtualization and operating system-level virtualization is quite insignificant. Indeed, in several cases we see a slight performance boost over the native execution time. These slight performance differences have previously been shown to occur [12], and may be due to the slight difference in kernel versions between the base system and Xen/OpenVZ.

In Figure 3(b) we show the relative execution time of the OpenMP implementations of NPB. We again show the class C results. This time, however, we found that Xen was unable to execute both the BT and SP benchmarks. As a consequence, we omit Xen's results for non-working benchmarks.

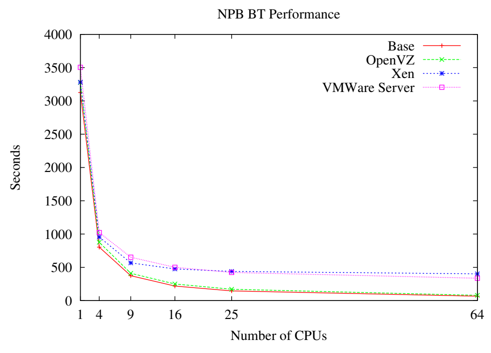
In general we see from Figure 3(b) that the relative performance of SMP processing via OpenMP is on-par with that of the native SMP performance. Similar to Figure 3(a) we see that both OpenVZ and Xen perform at native speeds, further suggesting that the overhead of both paravirtualization and operating system-level virtualization remains low even for parallel tasks. Indeed, for both OpenVZ and Xen, no benchmarks exhibit a relative execution time that is more than 1% slower than the native execution time.

VMWare Server, however, exhibits much greater SMP overhead than the serial benchmarks. Further, the number of benchmarks with runtimes of over 10% greater than the base time has also increased. Whereas the serial benchmarks see only IS exhibiting such a decrease in performance, three benchmarks (IS, LU, and CG) exhibit a decrease in performance of 10% or greater in the OpenMP results.

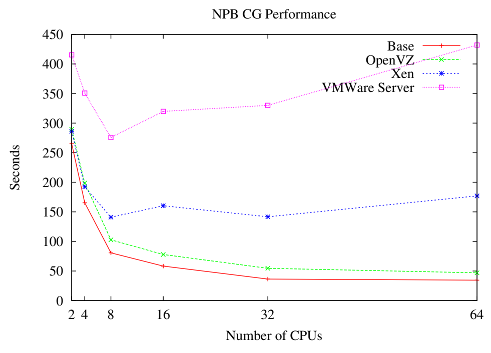
3.4 MPI Benchmarks

In Figure 4 we present the results of our MPI benchmark analysis, again using the Class C problem sizes of the NPB. We test each benchmark with up to 64 nodes (using 1 process per node). Unlike the serial and parallel/OpenMP results, it is clear from the outset that both VMWare Server and Xen suffer from a serious performance bottleneck, particularly in terms of scalability. Indeed, both VMWare Server and Xen exhibited exceptionally poor processor utilization as the number of nodes increased. In general, however, both Xen and VMWare Server were able to utilize, to some extent, the available processors to improve the overall run time with three notable exceptions: BT, CG, and SP. We only show results for EP, BT, CG, and SP in this paper as the remaining benchmarks exhibited similar results to those shown in Figure 4

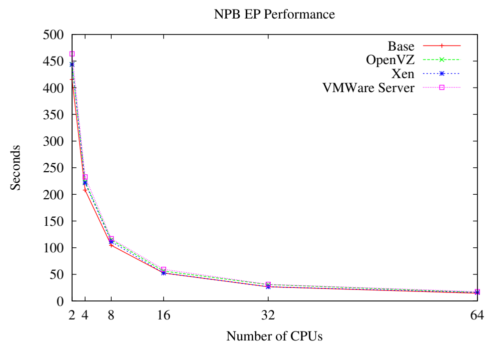
The results from Figure 4 suggest that the greatest overhead experienced by the guest operating systems is related to network utilization. For example, in Figure 4(c) we show the results of the "embarrassingly parallel" kernel, EP. EP



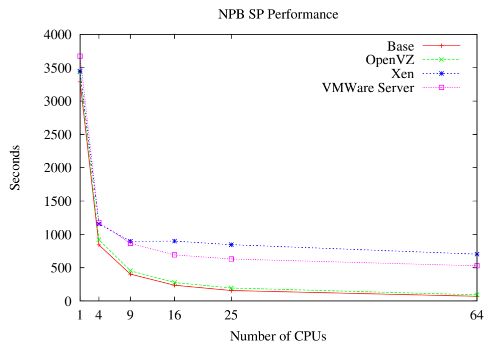
(a) MPI BT



(b) MPI CG



(c) MPI EP



(d) MPI SP

Figure 4. NPB MPI performance.

requires a minimum of network interaction, and as a consequence we see near-native performance for all virtualization technologies, including VMWare Server.

For benchmarks that make use of the network, however, the results are quite different. In fact, rather than closely following the results of OpenVZ and the base system, Xen now more accurately groups with VMWare Server. OpenVZ, however, largely follows the performance of the base system, particularly for the longer running computational benchmarks. Even the more network-heavy benchmarks, such as BT and SP, achieve near-native performance despite the lackluster bandwidth results shown in Figure 1(b). Unlike Xen, however, OpenVZ demonstrated native latencies (Figure 1(a)) which we believe is the primary reason for OpenVZ's scalability.

Both BT and SP are considered "mini applications" within the NAS Parallel Benchmark suite. They are both CFD applications with similar structure. While they are not considered network-bound, they are responsible for generating the greatest amount of network traffic (SP followed by BT) as shown by Wong, et al. [13]. We believe the primary reason for the poor performance of these benchmarks is the exceptionally high latencies exhibited by Xen and VMWare Server (see Figure 1(a)). Unlike CG, however, the modest performance improvement demonstrated with BT and SP is likely due to a small amount of overlap in communication and computation that is able to mask the high latencies to a small extent.

While the BT and SP benchmarks demonstrated poor performance, the CG benchmark was unique in that it demonstrated decreasing performance on both Xen and VMWare Server. This is likely due to the CG benchmark requiring the use of blocking sends (matched with non-blocking receives). Because of the exceptionally high penalty that Xen and VMWare Server observe in latency, it comes as no surprise that their blocking behavior severely impacts their overall performance. As we described in Section 3.1, Xen exhibits nearly twice the latency of the base system, and VMWare exhibits over three times the base system's latency. Not surprisingly, this suggests that for the NPB kernels, latency has a greater impact on scalability and performance than bandwidth as we see a corresponding decrease in benchmark performance with the increase in latency.

4 Related Work

There have been many descriptions of virtualization strategies in the literature, including performance enhancements designed to reduce their overhead. In Huang, et al. and Abali et al. [14, 12] the idea of VMM bypass I/O and scalable VM image management is discussed. VMM bypass I/O is used to reduce the overhead of network communication with Infiniband network interfaces. Using VMM bypass I/O it was shown that Xen is capable of near-native bandwidth and latency, which resulted in exceptionally low overhead for both NPB and HPL benchmarks using the MVAPICH2 Infiniband-based MPI implementation.

Raj et al. describe network processor-based self virtualizing network interfaces that can be used to minimize the overhead of network communication [15]. Their work reduces the network latency by up to 50%, but requires specialized network interface cards. Menon et al. provide an analysis of Xen and introduce the Xenoprof [16] tool for Xen VM analysis. Using the Xenoprof tool, they demonstrated Xen's key weaknesses in the area of network overhead.

Emeneker, et al. compare both Xen and User-Mode Linux (UML) for cluster performance in [17]. Their goal was to compare paravirtualization (Xen) against an operating system-level virtualization package (UML). They showed that Xen clearly outperforms UML in terms of performance, reliability, and the impact of the virtualization technologies. However, many consider UML to be more correctly categorized as paravirtualization, rather than OS-level virtualization.

In Soltesz, et al. [18] a comparison of Xen and Linux-VServer is discussed with special attention paid to availability, security and resource management. They show that contain-based/operating system-based virtualization is particularly well suited for environments where resource guarantees along with minimal overhead are needed. However, none of the papers have adequately examined the varying virtualization strategies for their use in HPC environments, particularly with regard to scalability. Our evaluation and experimentation fills this gap and provides a better understanding of the use of virtualization for cluster computing.

There are many additional virtualization strategies that are similar in nature to VMWare Server, Xen, and OpenVZ. The combination of Xen and QEMU [19] have been particularly popular and have spawned several products such as Virtual Iron [20]. The kernel-based Virtual Machine (KVM) [21] is a native virtualization solution for Linux on x86 hardware supporting virtualization extensions (Intel VT or AMD-V). It has been added to the mainline Linux kernel. Container/operating system-level virtualization approaches are also common. Solaris Zones [22] and FreeBSD Jails [23] are common examples.

VMWare ESX Server has been studied in regards to the architecture, memory management and the overheads of I/O processing [24, 25, 26, 27]. Because VMWare infrastructure products implement a full virtualization technology, they can have detrimental impact on the performance even though they provide easy support for operating systems.

5 Conclusions

We have performed an analysis of the effect of virtualization on scientific benchmarks using VMWare Server, Xen, and OpenVZ. Our analysis shows that, while none match the performance of the base system perfectly, OpenVZ demonstrates low overhead and high performance in both file system performance and industry-standard scientific benchmarks. While Xen demonstrated excellent network bandwidth, its exceptionally high latency hindered its scalability. VMWare Server, while demonstrating reasonable CPU-bound performance, was similarly unable to cope with the MPI-based NPB benchmarks. While OpenVZ shows potential for virtualization, additional work must be performed to improve the network bandwidth performance while also minimizing CPU utilization. We had originally hoped to compare VMWare ESX Server against Xen and OpenVZ; however, the strict compatibility list prevented us from doing so. We hope to include this comparison in the future, as additional compatibility is added to the VMWare ESX Server product.

References

[1] R. P. Goldberg, "Survey of Virtual Machine Research," in *Computer*, June 1974.

[2] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The NAS Parallel Benchmarks," *International Journal of High Performance Computing Applications*, vol. 5, no. 3, pp. 63–73, 1991. [Online]. Available: http://hpc.sagepub.com/cgi/content/abstract/5/3/63

[3] W. Norcott and D. Capps, "The IOZone Filesystem Benchmark," http://www.iozone.org.

[4] H. Packard, "Netperf," http://www.netperf.org.

[5] E. R. Altman, D. Kaeli, and Y. Sheffer, "Guest Editors' Introduction: Welcome to the Opportunities of Binary Translation," *Computer*, vol. 33, no. 3, pp. 40–45, 2000.

[6] S. Sridhar, J. S. Shapiro, E. Northup, and P. P. Bungale, “HDTrans: an open source, low-level dynamic instrumentation system,” in *VEE ’06: Proceedings of the second international conference on Virtual execution environments*. New York, NY, USA: ACM Press, 2006, pp. 175–185.

[7] K. Adams and O. Agesen, “A Comparison of Software and Hardware Techniques for x86 Virtualization,” in *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. New York, NY, USA: ACM Press, 2006, pp. 2–13.

[8] VMWare, “VMWare Server,” 2006, http://www.vmware.com.

[9] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, “Xen and the Art of Virtualization,” in *SOSP ’03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, 2003, pp. 164–177.

[10] SWSoft, “OpenVZ - Server Virtualization,” 2006, http://www.openvz.org/.

[11] J. Liu, W. Huang, B. Abali, and D. K. Panda, “High Performance VMM-Bypass I/O in Virtual Machines,” in *Proceedings of USENIX ’06*, 2006.

[12] W. Huang, J. Liu, B. Abali, and D. K. Panda, “A Case for High Performance Computing with Virtual Machines,” in *ICS ’06: Proceedings of the 20th annual international conference on Supercomputing*. New York, NY, USA: ACM Press, 2006, pp. 125–134.

[13] F. Wong, R. Martin, R. Arpaci-Dusseau, and D. Culler, “Architectural Requirements and Scalability of the NAS Parallel Benchmarks,” in *Supercomputing ’99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM Press, 1999, p. 41.

[14] C. Huang, G. Zheng, S. Kumar, and L. V. Kalé, “Performance Evaluation of Adaptive MPI,” in *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

[15] H. Raj and K. Schwan, “High Performance and Scalable I/O Virtualization via Self-Virtualized Devices,” in *the proceedings of HPDC 2007*, 2007.

[16] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, “Diagnosing Performance Overheads in the Xen Virtual Machine Environment,” in *VEE ’05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*. New York, NY, USA: ACM Press, 2005, pp. 13–23.

[17] W. Emeneker and D. Stanzione, “HPC Cluster Readiness of Xen and User Mode Linux,” in *2006 IEEE International Conference on Cluster Computing*, 2006.

[18] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors,” in *EuroSys 2007*, 2007.

[19] F. Bellard, “Qemu, a fast and portable dynamic translator,” in *Proceedings of the USENIX 2005 Annual Technical Conference, FREENIX Track*, 2005.

[20] V. I. Software, “Virtual iron,” http://www.virtualiron.com.

[21] Qumranet, “White Paper: KVM Kernel-based Virtualization Driver,” Qumranet, Tech. Rep., 2006. [Online]. Available: http://www.qumranet.com/wp/kvm_wp.pdf

[22] D. Price and A. Tucker, “Solaris Zones: Operating System Support for Consolidating Commercial Workloads,” in *LISA ’04: Proceedings of the 18th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 2004, pp. 241–254.

[23] P.-H. Kamp and R. N. M. Watson, “Jails: Confining the Omnipotent Root,” in *Proc. 2nd Int. SANE Conference*, 2000.

[24] I. Ahmad, J. M. Anderson, A. M. Holler, R. Kambo, and V. Makhija, “An Analysis of Disk Performance in VMware ESX Server Virtual Machines,” in *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on*, Oct. 2003, pp. 65–76.

[25] L. Cherkasova and R. Gardner, “Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor,” 2005, pp. 387–390. [Online]. Available: http://www.usenix.org/events/usenix05/tech/general/cherkasova.html

[26] C. A. Waldspurger, “Memory Resource Management in VMware ESX Server,” *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 181–194, 2002.

[27] J. E. Smith and R. Nair, “The Architecture of Virtual Machines,” *Computer*, vol. 38, no. 5, pp. 32–38, 2005.