2nd IEEE International Conference on Cloud Computing Technology and Science

Recommendations for Virtualization Technologies in High Performance Computing

Nathan Regola
Center for Research Computing
111 ITC Bldg, Univ. of Notre Dame
Notre Dame, IN 46556
Email: nregola@nd.edu

Jean-Christophe Ducom
Center for Research Computing
124 ITC Bldg, Univ. of Notre Dame
Notre Dame, IN 46556
Email: jducom@nd.edu

***Abstract*—The benefits of virtualization are typically considered to be server consolidation, (leading to the reduction of power and cooling costs) increased availability, isolation, ease of operating system deployment and simplified disaster recovery. High Performance Computing (HPC) environments pose one main challenge for virtualization: the need to maximize throughput with minimal loss of CPU and I/O efficiency. However, virtualization is usually evaluated in terms of enterprise workloads and assumes that servers are underutilized and can be consolidated. In this paper we evaluate the performance of several virtual machine technologies in the context of HPC. A fundamental requirement of current high performance workloads is that both CPU and I/O must be highly efficient for tasks such as MPI jobs. This work benchmarks two virtual machine monitors, OpenVZ and KVM, specifically focusing on I/O throughput since CPU efficiency has been extensively studied [1]. OpenVZ offers near native I/O performance. Amazon's EC2 "Cluster Compute Node" product is also considered for comparative purposes and performs quite well. The EC2 "Cluster Compute Node" product utilizes the Xen hypervisor in hvm mode and 10 Gbit/s Ethernet for high throughput communication. Therefore, we also briefly studied Xen on our hardware platform (in hvm mode) to determine if there are still areas of improvement in KVM that allow EC2 to outperform KVM (with InfiniBand host channel adapters operating at 20 Gbit/s) in MPI benchmarks. We conclude that KVM's I/O performance is suboptimal, potentially due to memory management problems in the hypervisor. Amazon's EC2 service is promising, although further investigation is necessary to understand the effects of network based storage on I/O throughput in compute nodes. Amazon's offering may be attractive for users searching for "InfiniBand-like" performance without the upfront investment required to build an InfiniBand cluster or users wishing to dynamically expand their cluster during periods of high demand.**

I. Introduction

Virtualization technology is not a novel concept [2]. It was introduced and implemented four decades ago on the IBM Mainframe Model 67. The virtual machine monitor (VMM), originally referred to as a supervisor, but now named a 'hypervisor', provides virtualized hardware interfaces to the virtual machine (VM). Each VM can then run an instance of its own operating system. With the growing economic pressures on data centers' operational costs as well as the environmental concerns in the past few years, virtualization is no longer confined to mainframe environments. Virtualization has become a mainstream solution to reduce the costs associated with power and cooling in data centers with commodity x86 hardware running enterprise workloads.

The most frequently cited benefits of virtualization for enterprise workloads are the reduction of power and cooling costs, increased availability, isolation, ease of deployment and simplified disaster recovery. Consolidating enterprise servers is possible because many servers were setup to achieve isolation (not as a result of high utilization on existing servers). However, a High Performance Computing (HPC) environment typically runs workloads that contain computationally intensive algorithms, large datasets, or both. Therefore, the goal of a HPC environment is to maximize the utilization and throughput of all available servers.

Consolidating HPC servers is usually not feasible because consolidation decreases the amount of work done and consumes more power. For example, consolidating two user jobs that each require 12GB of memory (from two machines with 13GB of RAM each) to one server with 13GB of RAM would not reduce power consumption or cooling. In fact, attempting this type of consolidation is likely to increase power consumption and application runtime as the two virtual machines cause excessive swapping and context switching when the server exhausts its memory. This problem can be eliminated by scheduling one virtual machine to run on each bare metal server.

Even with this concession, several advantages of virtualization for HPC remain: one of them is the possibility to live migrate a virtual machine when the underlying hardware is experiencing hardware failure. While minimizing downtime due to hardware errors is a worthy goal, perhaps the most compelling argument for virtualization from a HPC perspective is to enable each user to package their own customized environment with the specific libraries, compilers, and applications that they require.

Virtualization technology is traditionally thought to be a hindrance if the computational task runs for a long time. This is the case because even a fraction of one percent in performance penalty (remember that a VMM has to trap and process at least all privileged operations from the VMs) adds up very quickly over the course of weeks or months in datacenters containing thousands of servers. Numerous efforts at both the hardware [3] [4] and software [5] [6] levels

978-0-7695-4302-4/10 $26.00 © 2010 IEEE
DOI 10.1109/CloudCom.2010.71

have allowed a reduction in overhead to an acceptable level, especially for I/O operations, to the point where the benefits of virtualization outweigh its major drawback.

In this paper, we evaluated three well known open source hypervisors, KVM, OpenVZ, and Xen for suitable performance when running a HPC workload with OpenMP and MPI jobs: we will focus only on the overall performance of the VM and will not study the other two pillars of virtualization benchmarking which are performance isolation and scalability. In HPC clusters, the scheduler will maximize performance for tightly parallel applications by instantiating one virtual machine instance per individual compute node and by establishing a 1:1 mapping between virtual and physical cores. Sections II and III provide some background on KVM and OpenVZ, respectively, in a HPC context. We present our analysis of the performance overhead for disk and network I/O in Section IV and for a more complete HPC workload type in Section V. Related work is described in Section VI and we conclude with discussion in Section VII.

II. KVM IN HPC

The KVM code, which is rather small (about 10,000 lines), turns a Linux kernel into a hypervisor simply by loading a kernel module [7]. Instead of writing a hypervisor and the necessary components, such as a scheduler, memory manager, I/O stack, and device drivers, KVM leverages the ongoing development of the Linux kernel. The kernel module exports a device called /dev/kvm, which enables a guest mode of the kernel (in addition to the traditional kernel and user modes). With /dev/kvm, a virtual machine has a unique address space. Devices in the device tree (/dev) are common to all user-space processes. But /dev/kvm is different because each process sees a different device map in order to support isolation of the virtual machines. KVM takes advantage of hardware-based virtualization extensions [3] [4] to run an unmodified OS. Despite a simplified I/O call procedure, the I/O performance, as well as the lack of support for low latency drivers, plagued the acceptance of KVM as a viable VMM for HPC environments [8]. Para-virtualized I/O is the traditional way to accelerate I/O in virtual machines [5]. A major drawback of para-virtualization is that it requires modifications to the virtualized operating system. The recent development of the virtio driver, a para-virtualized I/O driver framework for KVM, and the support for PCI passthrough [9] [10] made KVM a strong candidate for HPC evaluation.

III. OPENVZ IN HPC

OpenVZ is a specially modified Linux kernel that enables Linux based operating systems to run as Linux processes [11]. The guest operating systems run unmodified binaries. However, there is only one kernel running on the host. This is an important distinction from KVM in terms of isolation and performance. Operating system virtualization does not provide the isolation of full virtualization (and thus the virtual machine must also be a Linux based distribution), but for HPC environments this is not a problem–almost all Top 500 clusters are Linux. Each virtual machine in KVM runs its own kernel. OpenVZ's approach is to use one kernel on the host to support both the host and any virtual operating systems. A crash (or security compromise) of the kernel by any guest would also crash or compromise the other operating systems. Aside from this limitation, and the fact that the guest must also be a Linux based distribution (it can be any distribution), the guest operating system is isolated (although not strictly isolated if we follow the defininition of isolation from Goldberg, et. al. [2]) from the host and can have its own IP address(es), user accounts, applications, and disk volume(s).

The first performance study of OpenVZ [1] compared the performance of Xen and OpenVZ with enterprise workloads. More specifically, they focused on using server consolidation to benchmark multi-tiered applications through RUBIS, an auction benchmark. They tested virtualized Apache, PHP and MySQL servers in various configurations and concluded that while Xen provided excellent isolation by allowing the user to run other x86 operating systems, this was at the expense of many L2 cache misses and extensive context switching. OpenVZ had lower CPU overhead and allowed faster response times to clients accessing servers because each operating system is just a process. Our paper focuses on the I/O overheads (disk and network) of OpenVZ and KVM for an MPI workload in a high performance computing environment, while [1] focused primarily on CPU overhead and the application response times of OpenVZ and Xen. While disk throughput and network latency were certainly included in the "end to end" response time calculations, they did not present I/O performance benchmark results or discuss the implications of I/O performance for various workloads.

We agree with the conclusions of Padala, et. al. [1] in regard to the tradeoffs between isolation and overhead. For HPC applications, the strict isolation of Xen is not needed and therefore operating system virtualization is a suitable candidate for virtualized I/O benchmarking.

IV. I/O EVALUATION

There are countless performance studies of virtualization in the HPC context with special attention given to disk and network I/O. However most of these studies focus on the more mature VMM Xen [12] [13] [14] [15] [6] [16]. In the first part, we will present our detailed analysis of the disk I/O performance for KVM and OpenVZ. We did not benchmark the disk I/O performance of Amazon's EC2 product. Elastic Block Storage (EBS), a network based block storage service, is used as a local disk for the "Cluster Compute Node". Therefore running IOZone on an EBS volume in our EC2 instances and comparing these results to a Xen image stored on a directly attached SAS disk would not allow us to draw rigorous and valid conclusions. In the second part we report the network overhead of each virtualization implementation in terms of latency and bandwidth for Gigabit Ethernet and an InfiniBand host channel adapter (HCA).

The experimental setup consists of four identical Dell R610 machines, each with two 2.27 GHz Intel Xeon E5520

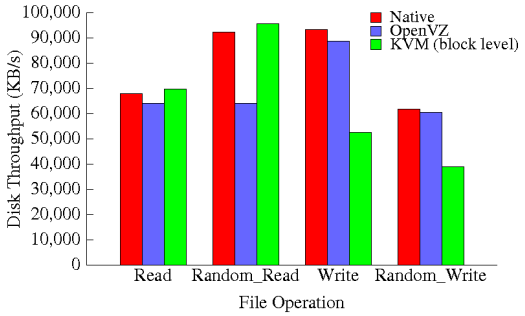

Fig. 1: Disk throughput, in KB/s, of read, random read, write, and random write operations from IOZone.

	Read	Random Read	Write	Random Write
native	66.27	89.98	91.06	60.31
OpenVZ	62.45	85.58	86.54	59.01

TABLE I: Disk throughput, in MB/s, of read, random read, write, and random write operations from IOZone in the OpenVZ container operating system.

processors (each processor has four cores), for a total of eight cores per server, 24GB of RAM, 73GB SAS hard drive, one Gigabit Ethernet card and one DDR Qlogic 7240 InfiniBand card. The network switches are Extreme Networks Summit 450 and the Qlogic Silverstorm 9040, supporting Gigabit Ethernet and InfiniBand, respectively. Fedora Core 12 was installed on bare metal hardware as it was supporting PCI passthrough [9], essential to test InfiniBand HCA within the virtual machines. All additional software was installed from the Fedora repository. However we had to run a KVM enabled kernel (2.6.32 based) on the host to run KVM 0.11.0-13 based virtual machines, another kernel (2.6.18 based) to run OpenVZ 3.0.24-1 virtualized operating systems and a Xen enabled kernel (2.6.32.16-1.2.108) to run Xen 3.4.3-2 virtual machines. The results of all three configurations are presented for completeness: the KVM kernel listed as 'KVM Host to KVM Host', the OpenVZ enabled kernel as 'OpenVZ Host to OpenVZ Host' and the Xen kernel as 'Xen Host to Xen Host'. We did not intentionally choose different kernel versions, but rather utilized the most current stable releases available at the time of testing. This was done because using the same kernel version for KVM, OpenVZ, and Xen would be impossible unless we resorted to backporting kernel code to untested versions of the kernel for each configuration. Also, updated kernel versions often include enhancements to virtual machine I/O performance and we thought that the only fair comparison was to use the latest "bleeding edge" code (and the matching kernel version) from each project for comparison. Finally, the virtual machine operating system (or guest operating system) was RedHat 5.4 in order to support fully commercial applications.

	Read	Rnd Read	Write	Rnd Write
native	66.27	89.98	91.06	60.31
block level	68.08	93.41	51.21	37.93
qcow2 none	12.44	16.46	11.40	8.20
qcow2 writeback	44.38	298.20	71.39	32.63
qcow2 writethrough	213.80	87.82	19.14	14.27
raw none	27.56	36.77	10.93	10.89
raw writeback	50.63	96.71	52.25	44.62
raw writethrough	112.13	229.65	45.96	31.67
vmdk none	52.03	63.72	5.39	5.41
vmdk writeback	87.37	367.55	33.28	31.84
vmdk writethrough	118.62	285.13	5.51	5.18

TABLE II: Disk throughput, in MB/s, of read, random read, write, and random write operations from IOZone across several KVM storage formats.

A. *Disk*

The Fedora Core 12 host machines were formatted identically with four partitions. All of the virtual machines used the ext3 file system with default settings and no logical volume manager. The partition containing the virtual machine image file was formatted with the ext3 file system (without logical volume manager) and contained only the virtual machine image file. The only exception to this was the KVM direct block level virtual machine. In this special situation, we assigned the /dev/sda4 partition to KVM and allowed the virtual machine to "own" the host's /dev/sda4 partition and format it for its exclusive use. In the direct access block level virtual machine, only one ext3 file system existed between the disk and the virtual machine (the file system that the virtual machine controlled). In all of the other situations, each virtual machine had its own ext3 file system encapsulated in an image file, which resided on the host's ext3 file system (/dev/sda4). The effect of the cache mechanism (no cache, writeback and writethrough) used on the host operating system for different image formats is shown in Table II. Allocating the /dev/sda4 partition on the host to the virtual machine instead of a KVM image file offered the highest performance. Even though this method is not very portable, we used it in the benchmarks in Section V to compare the highest performing I/O setup of each virtualization platform. The microbenchmarks suggest that while improvements have been made in regard to I/O performance with the assistance of para-virtualized drivers, these drivers are still only able to achieve 62% of native performance on random write file operations, while OpenVZ is able to achieve 97% of native performance (see Table I). IOZone demonstrates that even KVM's most efficient, but non-portable, block level access to a dedicated partition is inconsistent. While KVM demonstrates high read and random read performance, KVM has much lower performance than OpenVZ for write and random write operations (see Figure 1). Closer inspection reveals that OpenVZ can closely match native performance for read, write, and random write operations while KVM is only able to match native performance for read and random read operations. The reason for OpenVZ's low performance on random read is curious and warrants further investigation.

B. Network

Network latency is important for MPI jobs, as nodes frequently send small messages to other nodes. If the message size is small, the loss of efficiency due to latency overhead can be significant. We benchmarked the latency between two hosts capable of running the virtualized operating system (with the virtualized operating systems powered off) and between two virtualized operating systems (each virtualized operating system ran on its own host) to determine the performance characteristics of various configurations. Figure 2a demonstrates the overhead present in the KVM virtIO network drivers and in OpenVZ. These differences are significant with respect to MPI jobs, because each message must incur the overhead if KVM is used to virtualize MPI clusters.

The network throughput of various Ethernet configurations is described in Figure 2b. Amazon EC2, with 10Gbit/s Ethernet clearly outperforms 1Gbit/s Ethernet. KVM virtual machines, with both the para-virtualized virtio drivers and standard drivers, have far lower throughput than OpenVZ and Xen virtual machines on the same hardware. The KVM host machine kernel eventually performs as well as OpenVZ and Xen, for message sizes larger than 256 bytes. With message sizes over 512 bytes, the Xen, KVM, and OpenVZ host machine kernels are able to achieve over 930Mbit/s. Therefore, we can conclude that KVM needs further work in regard to network performance in virtual machines, as all three host kernels exhibit acceptable and consistent performance above 512 byte message sizes, while the virtualized operating systems are not consistent.

In conclusion, with respect to Gigabit Ethernet latency, KVM's para-virtualized drivers are capable of 8.3% of native performance, while OpenVZ is able to achieve 89.2% of native performance for a 2 byte message. Comparing bandwidth suggests that once again, operating system virtualization is better suited to HPC workloads. KVM achieves approximately 55% of native TCP throughput performance over Gigabit Ethernet connections (approximately 520Mbps), while OpenVZ is capable of 99.99% of native performance (approximately 941Mbps). Prior work [1] did not study OpenVZ network performance with benchmarks, but instead focused on CPU performance and end-to-end application overheads for enterprise multi-tier applications in a server consolidation context.

The Top500 list from June 2010 [17] demonstrates a 37% growth of InfiniBand [18] interconnects in clusters over one year: InfiniBand is now deployed on 208 systems ranked in the list. As the technology is becoming more accessible, the capability of accessing a HCA is crucial for any virtualization implementation. PCI passthrough allows the host to export the HCA to KVM [10] and Xen [19] virtual machines. Figure 3 shows the latency and the bandwidth of the HCA accessed from a VM measured with OMB-3.1.1 suite [20]. Figure 3 shows an artifact in both KVM bandwidth and latency results for a large window of message sizes ranging from 8192 bytes to 1048576 bytes. We did not have the time to investigate its origin: it could be caused by the version of the OpenMPI implementation used as well as a possible buffer limitation implemented in KVM.

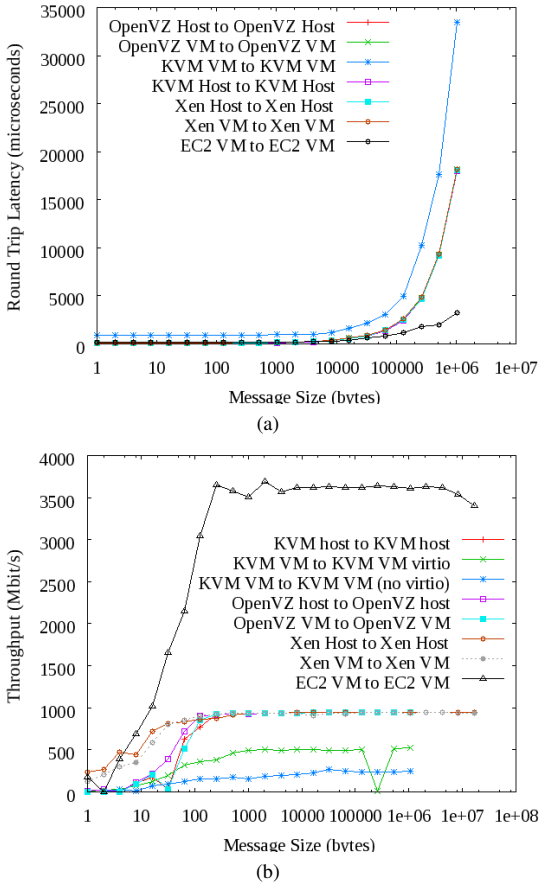

Fig. 2: Latency (2a) and throughput (2b) results for Ethernet based networking in KVM, OpenVZ, and Xen virtualized operating systems. The throughput (Figure 2b) at low message sizes is important for MPI applications. Xen outperforms OpenVZ and KVM until message sizes of 256bytes are sent. At around 256bytes, OpenVZ and Xen follow a similar climb to maximum throughput at 941Mbps. KVM is not able to achieve this throughput and achieves maximum throughput at around 1024bytes. This suggests that Xen will outperform OpenVZ and KVM in MPI applications for small message sizes (under 256bytes). For MPI message sizes over 256bytes, OpenVZ and Xen will offer similar networking performance.

V. NAS Parallel Benchmarks

The NAS Parallel benchmark NPB [21] [22] is a suite of benchmarks that emphasizes a particular type of numerical computation and reproduces the CPU, cache, memory, and I/O system workload of a wide range of "real world" applications. The NPB suite consists of five kernels (EP, MG, CG, FT, IS) and three simulated complex fluid dynamics (CFD) applications (BT, SP, LU). The problem size Class C was chosen for our tests due to its large memory footprint that will stress

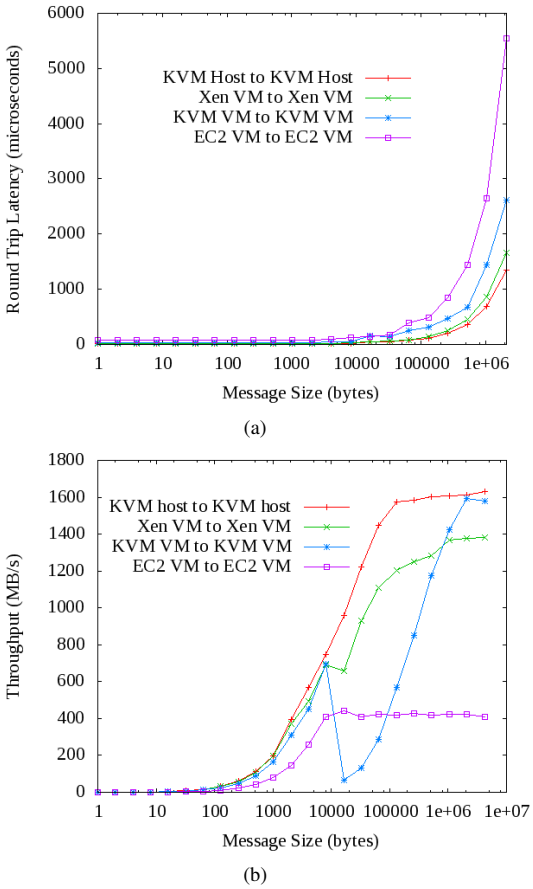

Fig. 3: Latency (3a) and throughput (3b) results for passthrough HCA InfiniBand based networking in KVM, Xen and 10 Gigabit Ethernet based networking in EC2 for comparison.

the virtualized system. The bulk of the computation is integer arithmetic in IS whereas all the other benchmarks are floating-point intensive. Finally, the NPB benchmarks exhibit a large variety of communication loads on the system, ranging from nearest-neighbor point-to-point exchange to coarse-grained all-to-all communication patterns.

Resource contention could hinder the scalability performance on native hardware [23] [24]. However, our interest is to quantify the overhead of virtualization technologies for I/O intensive applications using NPB benchmarks. The criterion to select a virtualization technology as a potential candidate in a HPC environment are performance as well as the percentage of degradation between a native and virtualized environment. More specifically, the evaluation of I/O performance is important because it allows a characterization of the ideal workload for a given hypervisor. For example, a CPU intensive application that only occasionally utilizes network or disk access may have identical performance on fully virtualized hypervisors, para-virtualized hypervisors and operating system level virtualization. However, if an application requires low

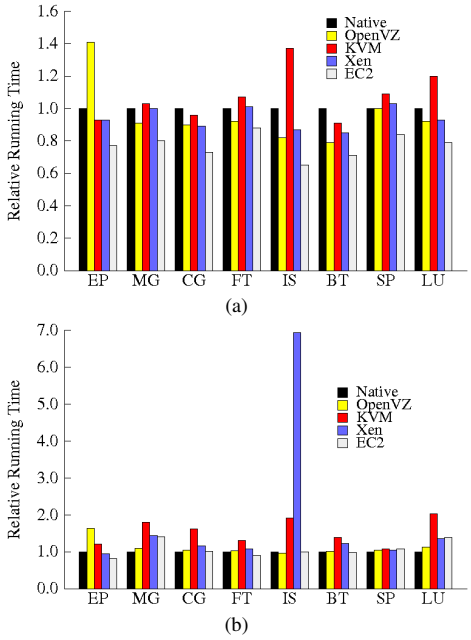

Fig. 4: Relative running time for NPB benchmarks Class C with OpenMP for one (4a) and eight (4b) cores on a single node.

latency communication for MPI jobs, the degradation of I/O performance could result in significant penalties for application performance. Therefore, it is important to determine the suitable workload for a given type of virtualization and identify the fundamental limitations of various virtualization strategies.

A. *OpenMP benchmark results on a single node*

NPB3.3-OMP was compiled on five configurations (native, OpenVZ, KVM, Xen, EC2) with gfortran based on gcc 4.1.2 in order to evaluate the baseline performance of the virtualization implementations on a single core. Due to the large size of the data set, the compiler flag -mcmodel=medium was used so that data structures larger than 2GB can be adressed; no other compiler optimization was performed. The Intel hyperthreading technology was disabled except on Amazon's EC2 service as their API does not allow user control of Hyperthreading yet. The results are shown in Figure 4a. The virtualization CPU overhead for KVM and OpenVZ is insignificant. In most of the cases the OpenVZ virtualization system performs even better than the native one: the difference in the kernel versions, discussed in Section IV, could be at the origin of the performance boost as also observed in [25]. To test the scalability of OpenVZ and KVM with a shared memory model, the number of threads was gradually increased up to the number of cores on a single node. In both environments, CPU affinity was used in order to avoid any over-subscription of a physical CPU with several virtual CPUs that could result in performance variability due to an increased number of context switches, a load imbalance, or contention in cache and memory accesses. In the one core case, no trend

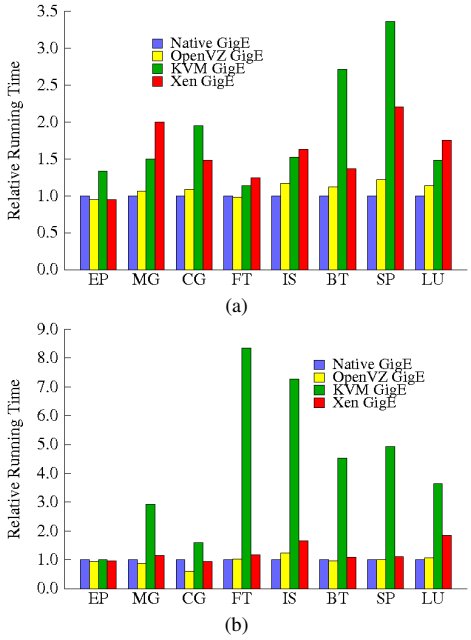

Fig. 5: Relative running time for NPB benchmarks Class C with MPI for eight (5a) and thirty two cores (5b) using Gigabit Ethernet.

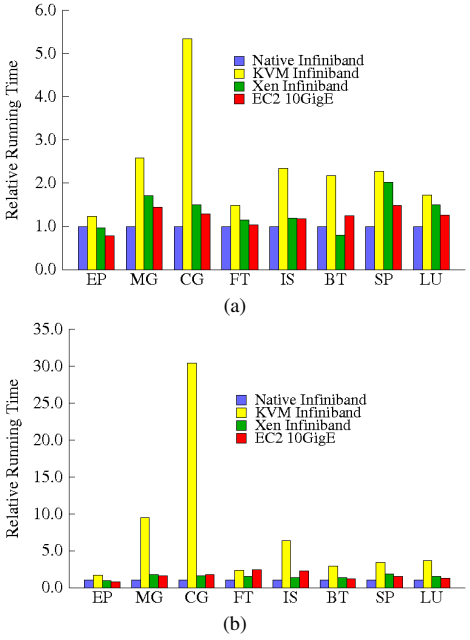

Fig. 6: Relative running time for NPB benchmarks Class C with MPI for eight (6a) and thirty two (6b) cores using InfiniBand and 10 Gigabit Ethernet.

in the virtualization overhead is clearly apparent. Actually, after averaging the relative running across all benchmarks for each configuration, Xen and OpenVZ show a 4% and 6% performance improvement, respectively, while KVM's average running time is within 5% of the native running time. The configuration used at EC2 shows an impressive 21% running time improvement: based on the Xen implementation, the performance can be solely attributed to the CPU used in the Amazon EC2 compute nodes, which exhibits significantly higher memory bandwidth and computational power than the CPU in our test nodes. The benchmark results for the number of threads, increased from one to eight, are represented in Figure 4b. While the parallel speedup is on average equal to 6.35 for the native configuration (a linear speedup would be 8 in our case), the parallel speedup for OpenVZ, KVM, Xen, and EC2 is respectively 5.38, 4.33, 4.43, and 4.74. The virtualization penalty is the most significant in benchmarks that require large amounts of communication or memory access, namely SP, MG, CG, and LU. OpenVZ overhead is still minimal, allowing near-native performance across all benchmarks. The performance significantly decreased for KVM, Xen, and EC2. In particular, EC2 configuration shows a 8% performance degradation in average compared to the native configuration. In the previous benchmark with one core, the same configuration was 21% faster in average. We believe that hyperthreading is the culprit.

B. MPI benchmark results

The benchmark suite NPB3.3-MPI was compiled with OpenMPI-1.4.2 based on gcc 4.1.2 on two sets of configurations: the first set is based on a Gigabit Ethernet connection between nodes, while the second set utilizes higher performance and lower latency network connections, such as InfiniBand and 10 Gigabit Ethernet. All CFD mini applications as well as most of the kernels except the purely computational EP benchmark create resource contention such as L2 cache, memory bandwidth and network bus in our multicore environment [26]: the average speedup for the native configuration is 1.01 in the first set. It is important to note that "average speedup is 1.01" does not mean that scalability is futile. For example, the EP speedup was 3.17 for the native OS using Gigabit Ethernet when going from 8 to 32 cores. While the other benchmarks create I/O contention, there is still reason to investigate the scalability (because scalability is possible, just not with every CPU and I/O workload combination). Due to the higher latency and even lower bandwidth for KVM and Xen, both demonstrate even worse scalability while OpenVZ offers near-native performance for all benchmarks. Lowering the latency and increasing the bandwidth are strategies to hide the effect of network contention. With the help of PCI passthrough, KVM and Xen could acccess the InfiniBand HCA that was installed on the host system. Amazon EC2 uses a 10 Gigabit Ethernet interconnect card by default. The InfiniBand interconnect tripled the native average speedup up to 3.05 (a linear speedup would be equal to 4), the biggest boost being visible for the CG, BT, and SP benchmarks. The virtualization overhead of KVM appears to be even more important than it was in the previous case studies. Due to the problem refernced in Section IV-B, KVM demonstrates the

worst performance throughout the entire suite of benchmarks. While Xen's performance degradations are less pronounced than KVM, the average overhead reaches 49% for 32cores, which is double the overhead measured previously with the Gigabit Ethernet environment. However the scalability is greatly improved leading to a 2.8 speedup factor. The same scalability observations result from the EC2 environment–the average overhead is 60%, leading to a 2.6 speedup factor. Due to the higher latency and lower bandwidth of the 10 Gigabit Ethernet demonstrated in Figure 3, the performance of EC2 does not scale as well as Xen hvm with InfiniBand support: while EC2 was outperforming Xen InfiniBand in all-but-one benchmark (Figure 6a) on 8core system, Figure 5b shows our Xen InfiniBand implementation is closing the performance gap despite a higher performing CPU in the EC2 configuration. The average virtualization overhead for EC2 increased from 21% with 8cores to 60% with 32cores.

VI. Related work

It is well known that compared to computationally intensive code, I/O intensive applications (especially network intensive applications) have far lower performance in virtualized environments. Prior benchmarking of OpenVZ [27] concluded that OpenVZ (and operating system virtualization in general) was the best choice for high performance computing due to lower overhead. At the time of Walters and Chaudhary [27], network bandwidth in OpenVZ was an abysmal 35.3% of native performance. OpenVZ is now able to achieve 100% of native performance. While this was a valid way of comparing overhead, we feel that a more realistic evaluation of SMP capability in virtual environments is warranted since the goal of high performance computing is to maximize the throughput of all available cores (to maximize cooling and power efficiency by utilizing all CPU cores in the host system, given that HPC servers run 24 hours per day). Our work used the NPB benchmark on all available cores (we assigned the guest eight cores) to determine if virtualization imposed any additional overhead when multiple virtual CPUs are used in compute intensive environments. For example, additional overhead from page table mapping strategies in the virtual machine monitor could surface as multiple virtual CPUs are assigned to a single guest. Our work extends the work of Walters and Chaudhary by benchmarking guests with multiple virtual CPUs and InfiniBand networking support. We also extend their work by benchmarking the various virtual hard drive file formats that are available. They used a maximum file size of 512MB, well within the buffer cache size of their host machine (their host machine had 2GB of RAM) and were unable to present throughput results for large files. We also used IOZone to benchmark the disk performance inside the guest, but focused on large files to determine the throughput characteristics of the virtual hard disk file formats, as well as the inherent differences between hypervisors: IOZone recommends a maximum file size larger than the memory size in order to determine the effects of the CPU cache, buffer cache, and actual disk spindle performance. We followed this recommendation and tested on file sizes of up to 32GB[1] since we used host machines with 24GB of RAM.

In another study, Naussbaum et al. [8] focused on the I/O performance of KVM and Xen, both with hardware assisted virtualization and para-virtualization solutions in a HPC context. They concluded that each implementation had their own strength in benchmarked areas such as latency and bandwidth for network and memory, disk throughput. Xen hvm was the only solution that was consistently performing the worst across all microbenchmarks. Our evaluation of a recent version of Xen (in hvm mode) and the EC2 offering (also based on hvm mode Xen) proved that the Xen implementation is on the contrary the most robust even if it does not carry the best performance for some HPC workloads. Moreover we extended Naussbaum et al. work by studying the use of OS virtualization and PCI passthrough for HPC workload. InfiniBand networking support is crucial for HPC environments because occasionally Gigabit Ethernet is a bottleneck for MPI jobs with a large amount of communication. Our benchmarks used PCI passthrough to determine how InfiniBand performs for virtualized MPI workloads. To our knowledge, this work presents the first benchmark results of PCI passthrough enabled InfiniBand for KVM and Xen.

Para-virtualization was introduced to reduce the performance penalty associated with emulated I/O access to virtual devices. TCP segmentation offloading, checksum offloading [14] or the optimization of the interrupt delivery route [16] increase the network I/O performance in virtualized environments. However, they do not increase the performance to the level of native performance. The most promising technique was introduced by Liu et al. [28] [29] for Xen VMM and Dinda et al. [30] for their home-brewed VMM, Palacios. They both attempt to reduce the network virtualization cost further by extending the OS-bypass mechanism inherent to low latency interconnects such as InfiniBand HCA to the VMM. In essence, the application, running in the virtual machine, is allowed to directly control the I/O device without the intervention of the VMM or the virtual machine OS. Since the device is directly controlled by the user application, with no involvment from the VMM layer, studies show that the cost of virtualization is considerably reduced for scientific kernel benchmarks [12] [13] [28]. Self-virtualized cards improve the throughput and latency performance significantly compared to the para-virtualized or emulated device interface. However, these cards are not widely available or financially practical for low cost compute nodes.

OpenVZ does not offer an InfiniBand driver that is the container equivalent of Xen-IB [29]. However, developers may include one in future releases of OpenVZ.

VII. Conclusion

High performance computing has traditionally avoided virtualization due to the inherent CPU and I/O overhead. CPU

[1]The largest file size we could accomodate in the virtual machine partition was 32GB due to the size of our hard disk. The host operating system also needed space for its files.

overhead has all but been eliminated, in operating system virtualization, para-virtualization and full virtualization. Careful examination of I/O overhead reveals that I/O performance has not yet matured to the level of CPU efficiency for low latency and high throughput applications, such as MPI or large scale data processing in para-virtualized and fully virtualized virtual machines. We showed that operating system virtualization (OpenVZ is one well known example) is currently the only solution that offers near native CPU and I/O performance. If the applications require higher network performance in terms of latency or bandwidth, PCI passthrough allows the virtual machine to directly access high end devices such as InfiniBand HCA. If PCI passthrough is utilized, NAS benchmarks results show the overall scalability greatly improves. However, the virtualization overhead increased significantly, orders of magnitude higher for KVM in some particular inter-node communication patterns. Further work is needed to investigate its origin. Finally, the newly released (July, 2010) Amazon EC2 product "Cluster Compute Node" exhibits performance comparable with our InfiniBand passthrough testbed and presents itself as an economically viable solution for HPC centers that cannot invest in high end network infrastructure or for organizations that want to dynamically expand their capacity during bursts of activity. HPC will only be able to take advantage of virtualization if the fundamental limitations of full virtualization are recognized (with respect to I/O) and resources are devoted to using operating system virtualization to achieve some of the benefits of virtualization. We believe that allowing users to package their own development environment for execution on a virtualized HPC environment will enable HPC computing to become more accessible to a wider range of scientists. This can only be achieved if the "high throughput" aspect of HPC is preserved. Thus far, operating system virtualization appears to be the only way to support the CPU and I/O requirements of HPC computing with minimal overhead.

REFERENCES

[1] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance evaluation of virtualization technologies for server consolidation," HP Labs, Tech. Rep., 2007.

[2] R. Goldberg, "Survey of virtual machine research," *Computer*, pp. 34–45, 1974.

[3] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert, "Intel virtualization technology for directed I/O," *Intel Technology Journal*, vol. 10, no. 03, pp. 179–192, August 2006.

[4] Advanced Micro Devices, "AMD I/O Virtualization Technology (IOMMU) Specification," February 2009, http://support.amd.com/us/Embedded_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf.

[5] R. Russell, "virtio: Towards a de-facto standard for virtual i/o devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 95–103, 2008.

[6] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for High Performance Computing with virtual machines," in *Processings of the 20th annual International Conference on Supercomputing*. ACM New York, NY, USA, 2006, pp. 125–134.

[7] Qumranet, "KVM: Kernel-based Virtualization Driver," Tech. Rep., 2006, http://www.qumranet.com/wp/kvm_wp.pdf.

[8] L. Nussbaum, F. Anhalt, O. Mornard, and J.-P. Gelas, "Linux-based virtualization for HPC clusters," in *Linux Symposium*, 2009, pp. 221–234.

[9] T. Jones, "Linux virtualization and PCI passthrough," October 2009, http://www.ibm.com/developerworks/linux/library/l-pci-passthrough/.

[10] Linux-KVM.org, "How to assign devices with VT-d in KVM," http://www.linux-kvm.org/page/How_to_assign_devices_with_VT-d_in_KVM.

[11] OpenVZ, http://openvz.org.

[12] L. Youseff, R. Wolski, B. Gorda, and C. Krintz, "Evaluating the performance impact of Xen on MPI and process execution in HPC systems," in *Virtualization Technology in Distributed Computing*. IEEE Computer Society Washington DC, USA, 2006.

[13] ——, "Paravirtualization HPC systems," in *Workshop on Xen in High Performance Cluster and Grid Computing*. Springer, 2006, pp. 474–486.

[14] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in xen," in *USENIX Annual Technical Conference*, May 2006, pp. 15–28.

[15] A. Menon, J. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the Xen virtual machine environment," in *International Conference on Virtual Execution Environment*, 2005.

[16] Z. Jian, L. Xiaoyong, and G. Haibing, "The optimization of Xen network virtualization," in *CSSE '08: Proceedings of the 2008 International Conference on Computer Science and Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 431–436.

[17] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon, "Top 500 supercomputer sites," http://www.top500.org, June 2010.

[18] InfiniBand Trade Association, "InfiniBand Architecture Specification," January 2008, release 1.2.1.

[19] Xen.org, "Xen PCI Passthrough," http://wiki.xensource.com/xenwiki/XenPCIpassthrough.

[20] OSU Microbenchmark, http://mvapich.cse.ohio-state.edu/benchmarks/.

[21] 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*, vol. 5, no. 3, pp. 63–73, Fall 1991.

[22] D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, A. Woo, and M. Yarrow, "The NAS parallel benchmarks 2.0," NASA Ames Research Center, Tech. Rep., 1995.

[23] H. Jin, R. Hood, J. Chang, J. Djomehri, D. Jespersen, and K. Taylor, "Characterizing application performance sensitivity to resource contention in multicore architectures," NASA Ames Research Center, Tech. Rep. NAS-09-002, 2009.

[24] S. Saini, A. Naraikin, R. Biswas, D. Barkai, and T. Sandstrom, "Early performance evaluation of a "nehalem" cluster using scientific and engineering applications," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1–12.

[25] J. P. Walters, V. Chaudhary, M. Cha, S. G. Jr., and S. Gallo, "A comparison of virtualization technologies for HPC," *Advanced Information Networking and Applications, International Conference on*, vol. 0, pp. 861–868, 2008.

[26] H. Jin, R. Hood, J. Chang, J. Djomehri, D. Jespersen, and K. Taylor, "Characterizing application performance sensitivity to resource contention in multicore architectures," NASA Ames Research Center, Tech. Rep. NAS-99-002, Novemeber 1999.

[27] J. P. Walters and V. Chaudhary, "A fault-tolerant strategy for virtualized HPC clusters," *J. Supercomput.*, vol. 50, no. 3, pp. 209–239, 2009.

[28] J. Liu, W. Huang, B. Abali, and D. K. Panda, "High performance VMM-bypass I/O in virtual machines," in *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2006, pp. 3–3.

[29] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *Processings of the 20th annual International Conference on Supercomputing*. ACM New York, NY, USA, 2006, pp. 125–134.

[30] J. A. F. Renato J. Figueiredo, Peter A. Dinda, "A case for grid computing on virtual machines," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*. IEEE Computer Society Washington, DC USA, 2003, p. 550.