A Case for High Performance Computing with Virtual Machines

WEI HUANG, JIUXING LIU, BULENT ABALI, AND DHABALESWAR K. PANDA

Technical Report
OSU-CISRC-4/06-TR38

A Case for High Performance Computing with Virtual Machines *

Wei Huang† Jiuxing Liu‡ Bulent Abali‡ Dhabaleswar K. Panda†

† Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{huanwei, panda}@cse.ohio-state.edu

‡ IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
{jl, abali}@us.ibm.com

Abstract

Virtual machine (VM) technologies are experiencing a resurgence in both industry and research communities. VMs offer many desirable features such as security, ease of management, OS customization, performance isolation, checkpointing, and migration, which can be very beneficial to the performance and the manageability of high performance computing (HPC) applications. However, very few HPC applications are currently running in a virtualized environment due to the performance overhead of virtualization. Further, using VMs for HPC also introduces additional challenges such as management and distribution of OS images.

In this paper we present a case for HPC with VMs by introducing a framework which addresses the performance and management overhead associated with VM-based computing. Two key ideas in our design are: Virtual Machine Monitor (VMM) bypass I/O and scalable VM image management. VMM-bypass I/O achieves high communication performance for VMs by exploiting the OS-bypass feature of modern high speed interconnects such as InfiniBand. Scalable VM image management significantly reduces the overhead of distributing and managing VMs in large scale clusters. Our current implementation is based on the Xen VM environment and InfiniBand, however, many of our ideas are readily applicable to other VM environments and high speed interconnects.

We carry out detailed analysis on the performance and management overhead of our VM-based HPC framework. Our evaluation shows that HPC applications can achieve almost the same performance as those running in a native, non-virtualized environment. Therefore, our approach holds promise to bring the benefits of VMs to HPC applications with very little degradation in performance.

*This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506; National Science Foundation grants #CNS-0403342 and #CCR-0509452; grants from Intel, Mellanox, Sun, Cisco, and Linux Networx; and equipment donations from Apple, AMD, IBM, Intel, Microway, Pathscale, Silverstorm and Sun.

1 Introduction

Virtual machine (VM) technologies were first introduced in the 1960s [8], but are experiencing a resurgence in both industry and research communities in recent years. A VM environment provides virtualized hardware interfaces to VMs through a Virtual Machine Monitor (VMM) (also called hypervisor). VM technologies allow running different guest VMs in a physical box, with each guest VM possibly running a different guest operating system. They can also provide secure and portable environments to meet the demanding requirements of computing resources in modern computing systems [5].

Recently, network interconnects such as InfiniBand [15], Myrinet [26] and Quadrics [37] are emerging, which provide very low latency (less than 5 μs) and very high bandwidth (multiple Gbps). Due to those characteristics, they are becoming strong players in the field of high performance computing (HPC). As evidenced by the Top 500 Supercomputer list [41], clusters, which are typically built from commodity PCs connected through high speed interconnects, have become the predominant architecture for HPC since the past decade.

Although originally more focused on resource sharing, current virtual machine technologies provide a wide range of benefits such as ease of management, system security, performance isolation, checkpoint/restart and live migration. Cluster-based HPC can take advantage of these desirable features of virtual machines, which is especially important when ultra-scale clusters are posing additional challenges on performance, scalability, system management, and administration of these systems [33, 16].

In spite of these advantages, VM technologies have not yet been widely adopted in the HPC area. This is due to the following challenges:

- Virtualization overhead: To ensure system integrity, the virtual machine monitor (VMM) has to trap and process privileged operations from the guest VMs. This overhead is especially visible for I/O virtualization, where

the VMM or a privileged host OS has to intervene every I/O operation. This added overhead is not favored by HPC applications where communication performance may be critical. Moreover, memory consumption for VM-based environment is also a concern because a physical box is usually hosting several guest virtual machines, with each guest VM running a separate OS.

- Management Efficiency: Though it is possible to utilize VMs as static computing environments and run applications in pre-configured systems, as in non-virtualized environments, high performance computing cannot fully benefit from VM technologies unless there exists a management framework which helps to map VMs to physical machines, dynamically distribute VM OS images to physical machines, boot-up and shutdown VMs with low overhead in cluster environments.

In this paper, we take on these challenges and propose a VM-based framework for HPC which addresses various performance and management issues associated with virtualization. In this framework, we reduce the overhead of network I/O virtualization through VMM-bypass I/O [20]. The concept of VMM-bypass extends the idea of OS-bypass [47, 46, 31, 2, 35], which takes the shortest path for time critical operations through user-level communication. An example of VMM-bypass I/O was presented in Xen-IB [20], a prototype we developed to virtualize InfiniBand under Xen [9]. Bypassing the VMM for time critical communication operations, Xen-IB provides virtualized InfiniBand devices for Xen virtual machines with near-native performance. Our framework also provides the flexibility of using customized kernels/OSes for individual HPC applications. It also allows building very small VM images which can be managed very efficiently. With detailed performance evaluations, we demonstrate that high performance computing jobs can run as efficiently in our Xen-based cluster as in a native, non-virtualized InfiniBand cluster. Although our current prototype implementation is based on InfiniBand and Xen, we believe that our framework can be readily extended for other high-speed interconnects and other VMMs. To the best of our knowledge, this is the first study to adopt VM technologies for HPC in modern cluster environments equipped with high speed interconnects.

In summary, the main contributions of our work are:

- We propose a framework which allows high performance computing applications to benefit from the desirable features of virtual machines. To demonstrate the framework, we have developed a prototype system using Xen virtual machines on an InfiniBand cluster.

- We describe how the disadvantages of virtual machines, such as virtualization overhead, memory consumption, management issues, etc., can be addressed using current technologies with our framework.

- We carry out detailed performance evaluations on the overhead of using virtual machines for high performance computing. This evaluation shows that our virtualized InfiniBand cluster is able to deliver almost the same performance for HPC applications as those in a non-virtualized InfiniBand cluster.

The rest of the paper is organized as follows: To further justify our motivation, we start with more discussion on the benefits of VM for HPC in Section 2. In Section 3, we provide the background knowledge for this work. We identify the key challenges for VM-based computing in Section 4. Then, we present our framework for VM-based high performance computing in Section 5 and carry out detailed performance analysis in Section 6. In Section 7, we discuss several issues in our current implementation and how they can be addressed in future. Last, we discuss the related work in Section 8 and conclude the paper in Section 9.

2 Benefits of Virtual Machines for HPC Applications

With the deployment of large scale clusters for HPC applications, management and scalability issues on these clusters are becoming increasingly important. Virtual machines can greatly benefit cluster computing in such systems, especially from the following aspects:

- **Ease of management:** A system administrator can view a virtual machine based cluster as consisting of a set of virtual machine templates and a pool of physical resources [39]. To create the runtime environment for certain applications, the administrator needs only pick the correct template and instantiate the virtual machines on physical nodes. VMs can be shutdown and brought up much easier than real physical machines, which eases the task of system reconfiguration. VMs also provide clean solutions for live migration and checkpoint/restart, which are helpful when dealing with hardware problems like hardware upgrades and failure, which happen frequently in large-scale systems.

- **Customized OS:** Currently most clusters are utilizing general purpose operating systems, such as Linux, to meet a wide range of requirements from various user applications. Although researchers have been suggesting that light-weight OSes customized for each type of application can potentially gain performance benefits [44], this has not yet been widely adopted because of management difficulties. However, with VMs, it is possible to highly customize the OS and the run-time environment for each application. For example, kernel configuration, system software parameters, loaded modules,

as well as memory and disk space can be changed conveniently. VMs can also run special OSes developed exclusively for HPC instead of a general purpose OS.

- **System Security:** Some applications, such as system level profiling [30], may require additional kernel services to run. In a traditional HPC system, this requires either the service to run for all applications, or users to be trusted with privileges for loading additional kernel modules, which may lead to compromised system integrity. With VM environments, it is possible to allow normal users to execute such privileged operations on VMs. Because the resource integrity for the physical machine is controlled by the VMM instead of the guest operating system, faulty or malicious code in guest OS will in the worst case crash a virtual machine, which can be easily recovered.

3 Background

In this section, we provide the background information for our work. Our implementation of VM-based computing is based on the Xen VM environment and InfiniBand. Therefor, we start with introducing Xen in Section 3.1 and the InfiniBand architecture in Section 3.2. Then in Section 3.3, we introduce Xenoprof, a profiling toolkit for Xen we used in this paper.

3.1 Overview of the Xen Virtual Machine Monitor

Xen is a popular high performance VMM originally developed at the University of Cambridge. It uses paravirtualization [51], which requires that the host operating systems be explicitly ported to the Xen architecture, but brings higher performance. However, Xen does not require changes to the application binary interface (ABI), so existing user applications can run without any modification.

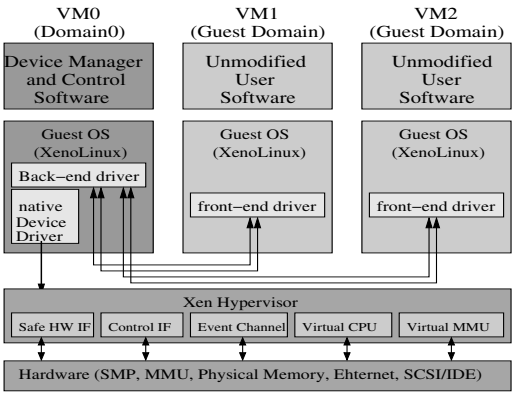

Figure 1. The structure of the Xen hypervisor, hosting three xenoLinux operating systems (courtesy [34])

Figure 1 illustrates the structure of a physical machine running Xen. The Xen hypervisor (the VMM) is at the lowest level and has direct access to the hardware. The hypervisor is running in the most privileged processor-level. Above the hypervisor are the Xen domains (VMs). There can be many domains running simultaneously. Guest OSes running in guest domains are prevented from directly executing privileged processor instructions. A special domain called *Domain0 (or Dom0)*, which is created at boot time, is allowed to access the control interface provided by the hypervisor. The guest OS in Dom0 hosts the application-level management software and performs the tasks to create, terminate or migrate other guest domains *(User Domain or DomU)* through the control interfaces.

In Xen, domains communicate with each other through shared pages and *event channels*, which provide an asynchronous notification mechanism between domains. A "send" operation on one side of the event channel will cause an event to be received by the destination domain, which may in turn cause an interrupt. Event channels are only intended for sending notifications between domains. If a domain wants to send data to another, the typical scheme is for a source domain to grant access to local memory pages to the destination domain. Then, these shared pages are used to transfer data.

Most existing device drivers assume they have complete control of the device, so there cannot be multiple instantiations of such drivers in different domains for a single device. To ensure manageability and safe access, device virtualization in Xen follows a split device driver model [12]. Each device driver is expected to run in an *Isolated Device Domain* (or *IDD*, typically the same as Dom0), which also hosts a *backend* driver, running as a daemon and serving the access requests from DomUs. The guest OS in DomU uses a *frontend* driver to communicate with the backend. The split driver organization provides security: misbehaving code in one DomU will not result in failure of other domains.

3.2 InfiniBand

InfiniBand [15] is a high speed interconnect offering high performance as well as features such as OS-bypass. InfiniBand host channel adapters (HCAs) are the equivalent of network interface cards (NICs) in traditional networks. A queue-based model is presented to the customers. A *Queue Pair (QP)* consists of a send queue and a receive queue. The send queue holds instructions to transmit data and the receive queue holds instructions that describe where received data is to be placed. Communication instructions are described in *Work Queue Requests (WQR)*, or descriptors, and are submitted to the work queue. Submitted WQRs are executed by Channel Adapters and the completions are reported through a *Completion Queue (CQ)* as *Completion Queue Entries (CQE)*.

InfiniBand requires all buffers involved in communica-

tion be registered before they can be used in data transfers. In Mellanox HCAs, the purpose of registration is two-fold. First, an HCA needs to keep an entry in the Translation and Protection Table (TPT) so that it can perform virtual-to-physical translation and protection checks during data transfer. Second, the memory buffer needs to be pinned in memory so that HCA can DMA directly into the target buffer Upon the success of registration, a local key and a remote key are returned. They will be used later for local and remote (RDMA) accesses.

Initiating data transfer (posting work requests) and completion of work requests notification (poll for completion) are time-critical tasks that need to be performed by the application in a OS-bypass manner. In the Mellanox [23] approach, which represents a typical implementation of InfiniBand specification, these operations are done by ringing a doorbell. Doorbells are rung by writing to the registers that form the *User Access Region (UAR)*. UAR is memory-mapped directly from a physical address space that is provided by HCA. It allows access to HCA resources from privileged as well as unprivileged mode. Posting a work request includes putting the descriptors (WQR) to a QP buffer and writing the doorbell to the UAR, which is completed without the involvement of the operating system. CQ buffers, where the CQEs are located, can also be directly accessed from the process virtual address space. These OS-bypass features make it possible for InfiniBand to provide very low communication latency.

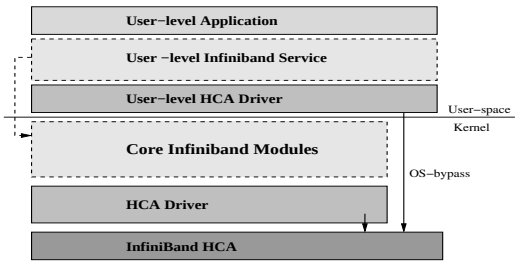

Figure 2. Architectural overview of OpenIB Gen2 stack

There are two popular stacks for InfiniBand drivers. VAPI [24] is the Mellanox implementation and OpenIB Gen2 [29] recently comes out as a new generation of IB stack provided by the OpenIB community. Xen-IB presented in this paper is based on the OpenIB-Gen2 driver. Figure 2 illustrates the architecture of the Gen2 stack.

3.3 Xenoprof: Profiling in Xen VM Environments

Xenoprof [25] is an open source system-wide statistical profiling toolkit developed by HP for the Xen VM environment. The toolkit enables coordinated profiling of multiple VMs in a system to obtain the distribution of hardware events such as clock cycles, cache and TLB misses, etc.

Xenoprof is an extension to Oprofile [30], the statistical profiling toolkit for Linux system. For example, Oprofile can collect the PC value whenever the clock cycle counter reaches a specific count to generate the distribution of time spent in various routines at Xen domain level. And through multi-domain coordination, Xenoprof is able to figure out the distribution of execution time spent in various Xen domains and the Xen hypervisor (VMM).

In this paper, we use Xenoprof as an important profiling tool to identify the bottleneck of virtualization overhead for HPC application running in Xen environments.

4 Challenges for VM-based Computing

In spite of the advantages of virtual machines, their usage in cluster computing has hardly been adopted. Performance overhead and management issues are the major bottlenecks for VM-based computing. In this section, we identify the key challenges to reduce the virtualization overhead and enhance the management efficiency for VM-based computing.

4.1 Performance Overhead

The performance overhead of virtualization includes three main aspects: CPU virtualization, memory virtualization and I/O virtualization.

Recently high performance VM environments such as Xen and VMware are able to achieve low cost CPU and memory virtualization [9, 36, 48]. Most of the instructions can be executed natively in the guest VMs except a few privileged operations, which are trapped by the VMM. This overhead is not a big issue because applications in the HPC area seldom call these privileged operations.

I/O virtualization, however, poses a more difficult problem. Because I/O devices are usually shared among all VMs in a physical machine, the VMM has to verify that accesses to them are legal. Currently, this requires the VMM or a privileged domain to intervene on every I/O access from guest VMs [42, 48, 12]. The intervention leads to longer I/O latency and higher CPU overhead due to context switches between the guest VMs and the VMM.

Figure 3 shows the performance of the NAS [27] Parallel Benchmark suite on 4 processes with MPICH [17] over TCP/IP. The processes are on 4 Xen DomUs, which are hosted on 4 physical machines. We compare the relative execution time with the same test on 4 nodes with native environment. For communication intensive benchmarks such as IS and CG, applications running in virtualized environments perform 12% and 17% worse, respectively. The EP benchmark, however, which is computation intensive with only a few barrier synchronization, maintains the native level of performance.

Table 1 illustrates the distribution of execution time collected by Xenoprof. We find that for CG and IS, around 30%

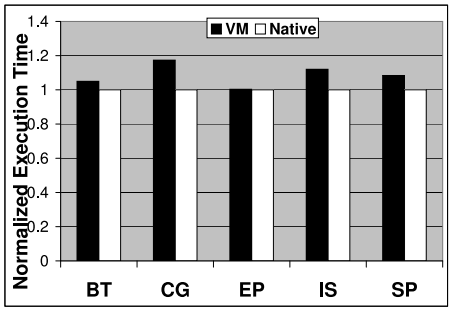

Figure 3. NAS Parallel Benchmarks

	Dom0	Xen	DomU
CG	16.6%	10.7%	72.7%
IS	18.1%	13.1%	68.8%
EP	00.6%	00.3%	99.0%
BT	06.1%	04.0%	89.9%
SP	09.7%	06.5%	83.8%

Table 1. Distribution of execution time for NAS

of the execution time is consumed by the isolated device domain (Dom0 in our case) and the Xen VMM for processing the network I/O operations. On the other hand, for EP benchmark, 99% of the execution time is spent natively in the guest domain (DomU) due to a very low volume of communication. It should be noted that in this example, each physical node hosts only one DomU with one processor, even though the node is equipped with dual CPUs. We notice that if we run two DomUs per node to utilize both the CPUs, Dom0 starts competing for CPU resources with user domains when processing I/O requests, which leads to even larger performance degradation compared to the native case.

This example identifies I/O virtualization as the main bottleneck for virtualization, which leads to the observation that I/O virtualization with near-native performance would allow us to achieve application performance in VMs that rivals native environments.

4.2 Management Efficiency

VM technologies decouple the computing environments from the physical resources. As mentioned in Section 2, this different view of computing resources bring numerous benefits such as ease of management, ability to use customized OSes, and flexible security rules. Although with VM environments, many of the administration tasks can be completed without restarting the physical systems, it is required that VM images be dynamically distributed to the physical computing nodes and VMs be instantiated each time the administrator reconfigure the cluster. Since VM images are often housed on storage nodes which are separate from the computing nodes, this poses additional challenges on VM image management and distribution. Especially for large scale clusters, the benefits of VMs cannot be achieved unless VM images can be distributed and instantiated efficiently.

5 A Framework for High Performance Computing with Virtual Machines

In this section, we propose a framework for VM-based cluster computing for HPC applications. We start with a high-level view of the design, explaining the key techniques used to address the challenges of VM-based computing. Then we introduce our framework and its major components. Last, we present a prototype implementation of the framework – an InfiniBand cluster based on Xen virtual machines.

5.1 Design Overview

As mentioned in the last section, reducing the I/O virtualization overhead and enhancing the management efficiency are the key issues for VM-based computing.

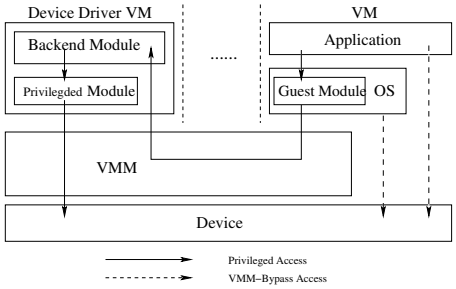

Figure 4. VMM-Bypass I/O

To reduce the I/O virtualization overhead we use a technique termed VMM-bypass I/O, which was proposed in our previous work [20]. VMM-bypass I/O extends the idea of OS-bypass I/O in the context of VM environments. The overall idea of VMM-bypass I/O is illustrated in Figure 4. In VMM-bypass I/O, a guest module in the VM manages all the privileged accesses, such as creating the virtual access points (i.e. UAR for InfiniBand) and mapping them into the address space of user processes. Guest modules in a guest VM cannot directly access the device hardware. Thus a *backend module* provides such accesses for guest modules. This backend module can either reside in the device domain (such as Xen) or in the VMM for other VM environments like VMware ESX server. The communication between guest modules and backend module is achieved through the inter-VM communication schemes provided by the VM environment. After the initial setup process, communication operations can be initiated directly from the user process. By removing the VMM from the critical path of communication, VMM-bypass I/O is able to achieve near-native I/O performance in VM environments.

To achieve the second goal, improving the VM image management efficiency, our approach has three aspects: customizing small kernels/OSes for HPC applications to reduce

the size of VM images that need to be distributed, developing fast and scalable distributing schemes for large-scale clusters and VM image caching on computing nodes. Reducing the VM image size is possible because the customized OS/kernel only needs to support a special type of HPC applications and thus needs very few services. We will discuss the detailed schemes in Section 5.3.

5.2 A Framework for VM-Based Computing

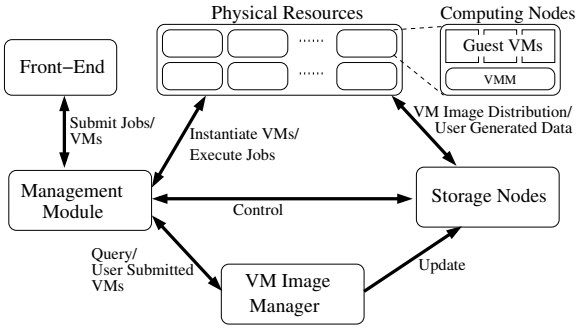

Figure 5. Framework for Cluster with Virtual Machine Environment

Figure 5 illustrates the framework that we propose for a cluster running in virtual machine environments. The framework takes a similar approach as batch scheduling, which is a common paradigm for modern production cluster computing environments. Users submit jobs and their runtime environment requirements through the front-end nodes. A management module decides if the job can be executed on the VM instances currently running on the physical resources. If not, it will query the VM image manager to find out the VM image which satisfies the user requirements, distribute the image onto physical nodes and instantiate the VMs to host the requested application. We have a highly modularized design so that we can focus on the overall framework and leave the detailed issues like resource matching, scheduling algorithms, etc., which may vary depending on different environments, to the implementations.

The framework consists of the following five major parts:

- **Front-end Nodes:** End users submit batch job requests to the management module. Besides the application binaries and the number of processes required, a typical job request also includes the runtime environment requirements for the application, such as required OS type, kernel version, system shared libraries etc. Users can execute privileged operations such as loading kernel modules in their jobs, which is a major advantage over normal cluster environment where users only have access to a restricted set of non-privilege operations. Users can be allowed to submit customized VMs containing special purpose kernels/OSes or to update the VM which they previously submitted. Since guest VMs are not trusted by the VMM which controls all the hardware resources, hosting user-customized VMs, just like running user-space applications, will not compromise the system integrity. The detailed rules to submit such user-customized VM images can be dependent on various cluster administration policies and we will not discuss them further in this paper.

- **Physical Resources:** These are the cluster computing nodes connected via high speed interconnects. Each of the physical nodes hosts a VMM and perhaps a privileged host OS (such as Xen Dom0) to take charge of the start-up and shut-down of the guest VMs. To avoid unnecessary kernel noise and OS overhead, the host OS/VMM may only provide limited services which are necessary to host VM management software and different backend daemons to enable physical device accesses from guest VMs. For HPC applications, a physical node will typically host no more VM instances than the number of CPUs/cores. This is different from the requirements of other computing areas such as high-end data centers, where tens of guest VMs can be hosted in a single physical box.

- **Management Module:** The management module is the key part of our framework. It maintains the mapping between VMs and the physical resources. The management module receives user requests from the front-end node and determines if there are enough idle VM instances that match the user requirements. If there are not enough VM instances available, the management module will query the VM image manger to find the VM image matching the runtime requirements for the submitted user applications. This matching process can be similar to those found in previous research work on resource matching in Grid environments, such as *Matchmaker* in Condor [28]. The management module will then schedule physical resources, distribute the image, and instantiate the VMs on the allocated computing nodes. The scheduling algorithm can be as simple as random picking of the computing nodes which has enough resources or as complex as considering many system/user-centric merits, such as studied in [7]. A good scheduling methodology will not only consider those merits that have been studied in current cluster environment, but also take into account the cost of distributing and instantiating VMs, where reuse of already instantiated VMs is preferred.

- **VM Image Manager:** The VM image manager manages the pool of VM images, which are stored in the storage system. To accomplish this, VM image manager hosts a database containing all the VM image information, such as OS/kernel type, special user libraries

installed, etc. It provides the management module with information so correct VM images are selected to match the user requirements. The VM images can be created by system administrator, who understands the requirements for the most commonly used high performance computing applications, or VM images can also be submitted by users if their applications have special requirements.

- **Storage:** The performance of the storage system as well as the cluster file system will be critical in the framework. VM images, which are stored in the storage nodes, may need to be transferred to the computing nodes during runtime, utilizing the underlying file system. To reduce the management overhead, a high performance storage system and file system are desirable.

5.3 An InfiniBand Cluster with Xen VM Environment

To further demonstrate our ideas of VM-based computing, we present a case of an InfiniBand cluster with Xen virtual machine environments. Our prototype implementation does not address the problems of resource matching and node scheduling, which we believe have been well studied in literature. Instead, we focus on reducing the virtualization overhead and the VM image management cost within our framework.

5.3.1 Reducing Virtualization Overhead

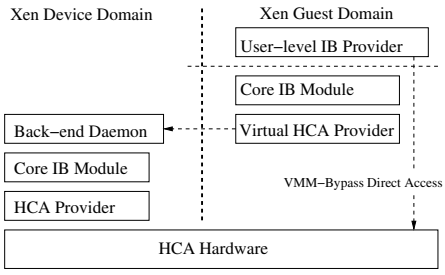

Figure 6. Xen-IB Architecture: Virtualizing InfiniBand with VMM-bypass I/O

As mentioned in Section 5.1, I/O virtualization overhead can be significantly reduced with VMM-bypass I/O techniques. Xen-IB is a VMM-bypass I/O prototype implementation we developed to virtualize the InfiniBand device for Xen. Figure 6 illustrates the general design of Xen-IB. Xen-IB involves the device domain for privileged InfiniBand operations such as initializing HCA, creating queue pair and completion queues, etc. However, Xen-IB is able to expose the *User Access Regions* to guest VMs and allow direct DMA operations. Thus, Xen-IB executes time critical operations such as posting work requests and polling for completion natively without sacrificing the system integrity. With the very low I/O virtualization overhead through Xen-IB, HPC applications have the potential to achieve near-native performance in VM-based computing environments.

Additional memory consumption is also a part of virtualization overhead. In the Xen virtual machine environment, the memory overhead mainly consists of the memory footprints of three components: the Xen VMM, the privileged domain, and the operating systems of the guest domains. The memory footprints of the virtual machine monitor is examined in detail in [9], which can be as small as 20KB of state per guest domain. The OS in Dom0 needs only support VM management software and the physical device accesses. And customized OSes in DomUs are providing the "minimum" services needed for HPC applications. Thus the memory footprints can be reduced to a small value. Simply by removing the unnecessary kernel options and OS services for HPC applications, we are able to reduce the memory usage of the guest OS in DomU to around 23MB when the OS is idle, more than two-thirds of reduction compared with running normal Linux AS4 OS. We believe it can be further reduced if more careful tuning is carried out here.

5.3.2 Reducing VM Image Management Cost

As previously mentioned, we take three approaches to minimize the VM image management overhead: making the virtual machine images as small as possible, developing scalable schemes to distribute the VM images to different computing nodes, and VM image caching.

Minimizing VM Images: In literature, there are lots of research efforts on customized and special OSes [40, 22, 19, 18], whose purpose is to develop infrastructures for parallel resource management and to enhance the OS for the ability to support systems with very large number of processors. In our Xen-based VM computing environment, our VM images include customized OS environments as thin as just one containing an MPI library, a very small set of system utilities, and some runtime environmental requirements such as ssh daemon. Such customized VM image can be very small and can be transferred through network and boot-up very efficiently.

Fast and Scalable Image Distribution: HPC applications will typically require multiple computing nodes. This requires VM images to be efficiently broadcasted to multiple physical nodes where the VMs will be instantiated. Even though a small VM image can be transferred very efficiently between the storage nodes and the computing nodes, we need a scalable VM image distribution scheme for large-scale clusters where jobs can include many nodes. For this purpose, we build a VM distribution module to broadcast the images to destination physical nodes through a binomial tree structure, which significantly speeds the VM image distribu-

tion.

VM Image Caching: We also use VM image caching on computing nodes to accelerate the time to launch a user request. The main idea is to cache the VM image on the local storage of the computing nodes, provided there is available space. In this case, we do not need to transfer the VM image to a computing node if the VM has not been updated since the last time it was instantiated on that computing node. We will continue to work on more advanced schemes to transfer only the updated portion of the VM image so the transfer will only include the difference, likely much smaller than the whole VM image. All these caching schemes will require additional coordination between the management module and the VM image manager to maintain consistency between different versions of VM images.

6 Evaluation

In this section, we evaluate the performance overhead and the VM management efficiency of an InfiniBand cluster with Xen VM environments. We first provide MPI-level performance results at both the micro-benchmark and application levels. Then, we present different efficiency metrics for VM image management, including image distribution time, additional memory consumption for the guest VMs, and time needed for startup/shutdown of a VM.

6.1 Experimental Setup

We conducted our performance evaluations on an eight-node InfiniBand cluster. Each node in the cluster is equipped with dual Intel Xeon 3.0GHz CPUs, 2 GB of memory and a Mellanox MT23108 PCI-X InfiniBand HCA. The systems are connected with an InfiniScale InfiniBand switch. The same cluster is used to obtain performance results for both the VM-based environment and the native, non-virtualized environment. In VM-based environments, we use Xen 3.0. The Xen domain0 hosts RedHat AS4 with kernel 2.6.12 with 256MB memory. All user domains are running with a single virtual CPU and 896 MB memory, which allows two DomUs per physical machine. Each guest OS in DomUs uses the 2.6.12 kernel with all unnecessary services removed. The OS is derived from ttylinux [45], with minimum changes to host MPI applications. In the native environment, we also use RedHat AS4 but with SMP mode.

Our performance results are collected using MVAPICH, a popular MPI implementation over InfiniBand [49, 21].

6.2 Micro-benchmark Evaluations

In this section, we compare MPI-level micro-benchmark performance between the Xen-based cluster and the native InfiniBand cluster. Tests were conducted between two user domains running on two different physical machines with Xen or between two different nodes in native environments.

The latency test repeated ping-pong communication for many times and the average half round-trip time is reported as one-way latency. As shown in Figure 7, there is very little difference between the Xen and the native environment, with both achieving 4.9μs for 1 byte messages. This shows the benefit of VMM-bypass I/O, where all communication operations in Xen are completed by directly accessing the HCAs from the user domains.

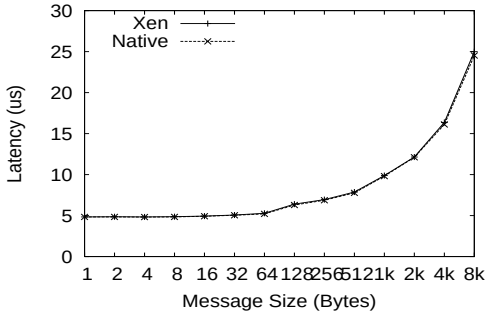

Figure 7. MPI latency test

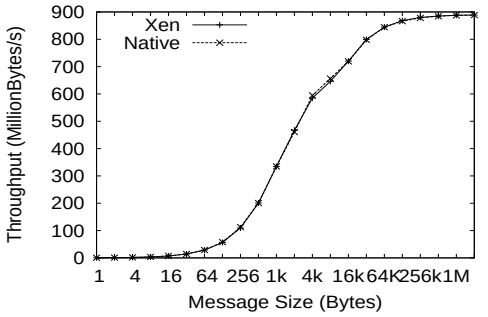

Figure 8. MPI bandwidth test

In the bandwidth tests, a sender sent a number of messages to a receiver using MPI non-blocking communication primitives. It then waited for an acknowledgment and reported bandwidth by dividing number of bytes transferred by the elapsed time. We again see almost no difference between Xen and native environments. As shown in Figure 8, in both cases we achieved 880MB/sec. (Please note that all bandwidth numbers in this paper are reported in millions of bytes per second.)

MVAPICH implements a registration cache [43], which allows zero-copy transfer for large messages without memory registration if the communication buffer is reused [21]. The above tests benefited from this technique. However, our previous work evaluated the buffer registration time and revealed that registration in Xen user domain is more costly, as illustrated in Figure 9. It is because registration is a privileged operation and needs to be conducted in the device domain.

To show the worst-case scenario in the Xen-based environment, we repeated the above tests with registration cache

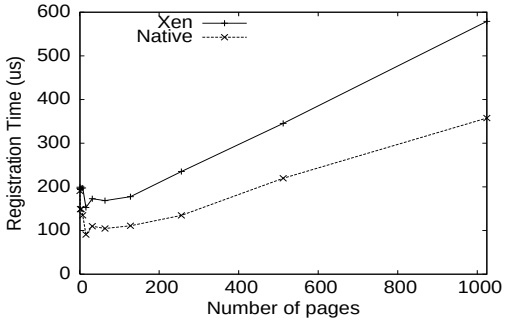

Figure 9. Memory registration time

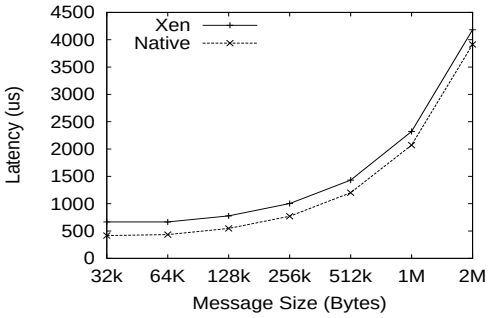

Figure 10. MPI latency test without registration cache (large messages)

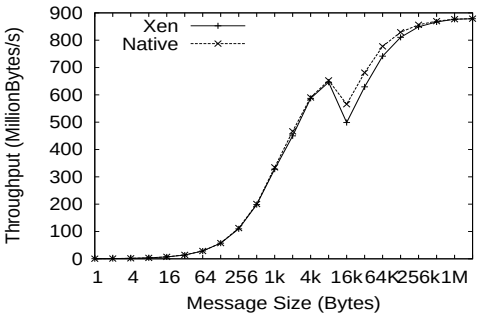

Figure 11. MPI bandwidth test without registration cache

disabled. We observed no difference for short message latency and bandwidth because MVAPICH copies the short messages to pre-registered buffers before sending them out. Figures 10 and 11 show the latency and bandwidth results for large messages. We observe that due to the extra cost of registration, medium to large message latencies in Xen environment is constantly around 200μs higher, which conforms to what we observed in Figure 9 since both the sender and the receiver side need to register the communication buffers. For bandwidth test, because registration time can be overlapped with network communication, there is only a maximum 11% difference between the Xen and native environment for medium size messages (16KB), and we see virtually no difference for large messages. Again, this is the worst case scenario for Xen-based environments. Actual HPC applications usually have good buffer reuse patterns and in most cases we will not see this difference. Moreover, the gap for medium size message latency can be reduced through performance tuning by allowing those messages to use pre-registered buffers for communication, which has been studied in [14].

6.3 HPC Application Evaluations

In this section, we evaluate the Xen-based computing environment with actual HPC applications. We instantiate two DomUs on each physical machine and run one process per DomU for the Xen case. (Each DomU runs a uni-processor kernel.) And for native environment, we run two processes per physical node.

We evaluate both NAS parallel benchmarks and High Performance Linpack (HPL). HPL is the parallel implementation of Linpack [32] and the performance measure for ranking the computer systems of Top 500 supercomputer list.

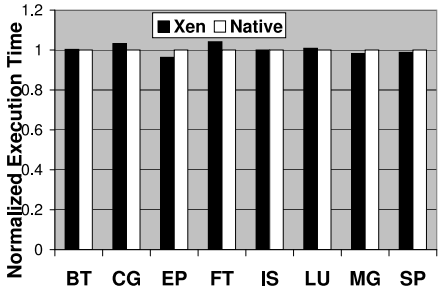

Figure 12. NAS Parallel Benchmarks (16 processes, class B)

The execution time for NAS has been normalized based on the native environment in Figure 12. We observed that Xen-based environment performs comparably with the native environment. For NAS applications CG and FT, where native environment performs around 4% better, the gap is due

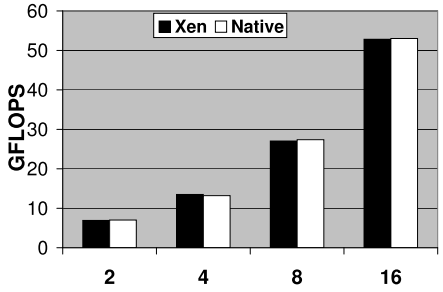

Figure 13. HPL on 2, 4, 8 and 16 processes

	Dom0	Xen	DomUs
IS	3.6	1.9	94.5
SP	0.3	0.1	99.6
BT	0.4	0.2	99.4
EP	0.6	0.3	99.3
CG	0.6	0.3	99.0
LU	0.6	0.3	99.0
FT	1.6	0.5	97.9
MG	1.8	1.0	97.3

Table 2. Distribution of execution time for NAS

to the ability of MVAPICH to utilize shared memory communication for processes on the same node in the native environment. Our prototype implementation of VM-based computing currently does not have this feature. However, high speed communication between Xen domains on the same physical node can be added by taking advantage of the page sharing mechanism provided by Xen.

We also report the Gflops achieved in HPL on 2, 4, 8 and 16 processes in Figure 13. We observe very close performance here with native environment outperforms at most 1%.

Table 2 shows the distribution of the execution time among Dom0, Xen hypervisor, and two DomUs (sum) for NAS benchmarks. As we can see, due to VMM-bypass approach, most of the instructions are executed locally in user domains, which achieves highly efficient computing. For IS, the time taken by Dom0 and the Xen hypervisor is slightly higher. This is because IS has very short running time and Dom0/hypervisor are mainly involved during application startup/finalize, where all the connections need to be set up and communication buffers need to be pre-registered.

6.4 VM Image Management Evaluation

In this section, we evaluate the VM image management overhead for the Xen-based environment.

Table 3 shows the time to startup and shutdown a domain in Xen environments as well as the extra memory consumed by the OSes (after a fresh startup) running in each user domain. As we can see, managing a guest domain with

	startup	shutdown	memory
ttylinux-domu	5.3s	5.0s	23.6MB
AS4-domu	24.1s	13.2s	77.1MB
AS4-native	58.9s	18.4s	90.0MB

Table 3. Startup, shutdown time and extra memory consumption

Scheme	1	2	4	8
Binomial tree	1.3s	2.8s	3.7s	5.0s
NFS	4.1s	6.2s	12.1s	16.1s

Table 4. VM image distribution time

a customized OS/kernel (ttylinux) has much less overhead compared with a guest domain with normal full featured OS (AS4). Additionally, the extra memory consumption is also reduced to around 23.6 MB for ttylinux, which is about 1.2% of the total memory we have on each physical node. We also include the startup/shutdown time and memory consumption for RedHat AS4 on native servers as a reference. We can clearly see that starting a VM is much faster than starting an physical server.

We also measure the VM image distribution time. As we are using customized OS for the guest VMs, we were able to reduce the VM image size to around 30MB. We distribute the VM image to the physical nodes through a binomial tree topology over IPoIB (IP over InfiniBand [52]). As we can see in Table 4, the binomial tree distribution algorithm can distribute the image to 8 physical nodes within 5 seconds. While using normal NFS, the distribution process needs around 16 seconds. Since the distribution time with the binomial tree algorithm increases proportional to logN, where N is the number of destination nodes, we believe our VM image distribution scales for very large size clusters. Note that the VM image distribution occurs only if VM image is not cached or has been updated, in practice, the distribution time can be even smaller due to caching.

7 Discussion

In this section we discuss several issues with our current prototype of a virtualized InfiniBand cluster and how they can be addressed in future.

7.1 Checkpoint and Migration

Although our framework can take advantage of VM checkpoint and migration, the VMM-bypass approach that we propose to reduce the I/O virtualization overhead poses additional challenges. With VMM-bypass I/O, the VMM is not involved in I/O operations, and thus cannot easily suspend and buffer those operations when checkpoint or migration starts. Further, high-speed interconnects where VMM-bypass approaches are applicable usually store part of the communication context on board. For example, InfiniBand HCAs will store information such as created queue-

pair structure, a translation table between virtual address and actual PCI address, etc. on the device. This makes checkpointing or migration even more difficult since part of the VM status is not accessible from the VMM. To address these issues, we plan to involve guest virtual machines, which have the knowledge of the communication status, in the checkpoint and migration process.

7.2 Memory Consumption of the Privileged Domain

The Xen virtual machine environment needs a privileged domain (Dom0) running on each machine to manage other user domains. Xen must allocate memory for the privileged domain at the boot time of the physical machine. At our current stage of research, we are focusing on customizing guest OS/kernels in user domains. Thus we just statically allocate 256 MB for Dom0, which is hosting a normal RedHat AS4 OS. Since this part of memory is used for Dom0, it cannot be used for actual computing applications. However, since Dom0 is hardly involved in HPC applications especially with VMM-bypass I/O, this overhead can be significantly reduced by careful tuning of the host OS. Meanwhile, since privileged operations will still need to be taken care of in Dom0, it is important to understand the memory requirements for Dom0 for hosting different applications in the user domains. We plan to carry more studies along this direction in future.

7.3 Performance Isolation

With VMM-bypass I/O, every VM will carry out network I/O operations, irrespective of any performance isolation or QoS policies among different VMs on the same physical machine. In future work, this issue can be addressed by utilizing the QoS schemes implemented in the hardware of many high-performance interconnects. For example, in current Mellanox InfiniBand hardware, a weighted round-robin scheme is provided to schedule the workloads among different queue pairs. It also supports different QoS levels for each individual queue pair, which are specified when the queue pair is initialized. After that the hardware will enforce the QoS policies. Those schemes can be utilized to make better performance isolations among VMs.

8 Related Work

Currently, many VM designs exploit multiple VMs in cluster-like environments. For example, the Xenoserver project [38] builds a VM-based infrastructure for wide-area distributed computing. μDenali [50] allows users to construct a virtual service platform by assembling multiple virtual hardware elements and virtual machines. The vMatrix [1] project proposed the idea of using a network of virtual machine monitors for dynamic content distribution. However, none of the above address the problem of using VMs for high performance computing. In [11], the authors advocate an approach of using VMs for grid computing. However, the hardware environments for grid computing are usually loosely-coupled, wide-area systems. Therefore, they did not explicitly address the issues of I/O performance degradation due to virtualization. In our work, we focus on traditional high performance computing environments, which are more tightly-coupled and have very stringent requirements for communication performance.

We first presented the idea of VMM-bypass in [20]. VMM-bypass extends the idea of OS-bypass to VM environments. With VMM-bypass, I/O and communication operations can be initiated directly by user space applications, bypassing the guest OS, the VMM, and the device driver VM. VMM-bypass also allows an OS in a guest VM to carry out many I/O operations directly, although virtualizing interrupts still needs the involvement of the VMM. OS-bypass is a feature found in user-level communication protocols such as active messages [47], U-Net [46], FM [31], VMMC [2], and Arsenic [35]. Later, it was adopted by the industry [10, 15] and appeared in commercial products [26, 37].

Virtual machine OS image distribution and start-up are important issues in any VM-based environment. However, our use of VMs in cluster-based HPC systems introduces new challenges because these systems can be very large scale, which requires a highly scalable approach. Complementary to our work, scalable job launching in traditional HPC environments has been discussed in studies such as [53] and [4].

High level middleware- and language-based virtual machines have been studied and used for high performance computing. Examples include PVM [13], HPVM [6], and Java [3]. In our work, virtual machines refer to those which provide abstractions identical or similar to existing hardware. Therefore, our VM-based platform is implemented at a much lower level and provides more flexibility. For example, our approach can not only support MPI, but also all the middleware- and language-based virtual machines mentioned above.

9 Conclusions and Future Work

In this paper, we proposed a framework for VM-based computing for HPC applications. We presented a case for building an InfiniBand cluster with the Xen virtual machine environment. We explained how we reduced I/O virtualization overhead with VMM-bypass I/O and addressed the efficiency for management operations in such VM-based environments. Our performance evaluation showed that HPC applications can achieve almost the same performance as those running in a native, non-virtualized environment. We also demonstrated quantitatively that other costs of virtualization, such as extra memory consumption and VM image management, can be reduced significantly by optimization. Our approach can potentially benefit HPC applications with desir-

able features of virtual machine technologies, including ease of management, OS customization, performance isolation, check-pointing, and migration, with very little performance degradation.

In future, we will work on checkpoint/migration support for VMM-bypass I/O. We also plan to provide better performance isolation among VMs by implementing QoS with Xen-IB, our VMM-bypass I/O implementation to virtualize InfiniBand in the Xen virtual environment. We are also investigating scheduling and resource management schemes for the efficient management in such VM-based clusters. We will also carry out more detailed performance evaluations of VM-based computing on clusters with larger numbers of nodes.

References

[1] A. Awadallah and M. Rosenblum. The vmatrix: A network of virtual machine monitors for dynamic content distribution. In *Proceedings of Seventh International Workshop on Web Content Caching and Distribution*, 2002.

[2] M. Blumrich, C. Dubnicki, E. W. Felten, K. Li, and M. R. Mesarina. Virtual-Memory-Mapped Network Interfaces. In *IEEE Micro*, pages 21–28, Feb. 1995.

[3] F. Breg, S. Diwan, J. Villacis, J. Balasubramanian, E. Akman, and D. Gannon. Java RMI performance and object model interoperability: experiments with Java/HPC++. *Concurrency: Practice and Experience*, 10(11–13):941–955, 1998.

[4] R. Brightwell and L. A. Fisk. Scalable Parallel Application Launch on Cplant. In *Proceedings of Supercomputing '01*, 2001.

[5] P. M. Chen and B. D. Noble. When Virtual is Better than Real. *Hot Topics in Operating Systems*, pages 133–138, 2001.

[6] A. Chien, M. Lauria, R. Pennington, M. Showerman, G. Iannello, M. Buchanan, K. Connelly, L. Giannini, G. Koenig, S. Krishnamurthy, Q. Liu, S. Pakin, and G. Sampemane. Design and evaluation of an HPVM-based Windows NT supercomputer. *The International Journal of High Performance Computing Applications*, 13(3):201–219, Fall 1999.

[7] B. Chun and D. Culler. User-centric Performance Analysis of Market-based Cluster Batch Schedulers. In *Proceedings of CCGrid*, 2002.

[8] R. J. Creasy. The Origin of the VM/370 Time-sharing System. *IBM Journal of Research and Development*, 25(5):483–490, 1981.

[9] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, pages 164–177, October 2003.

[10] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A.M. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, pages 66–76, March/April 1998.

[11] R. Figueiredo, P. Dinda, and J. Fortes. A case for grid computing on virtual machines. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS), May 2003.*, 2003.

[12] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M̃. Williamson. Reconstructing I/O. Technical Report UCAM-CL-TR-596, University of Cambridge, UK, August 2004.

[13] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, USA, 1994.

[14] W. Huang, G. Santhanaraman, H.-W. Jin, Q. Gao, and D. K. Panda. Design and Implementation of High Performance MVAPICH2: MPI2 over InfiniBand. In *Proceedings of CCGrid '06*, Accepted for publication.

[15] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.2.

[16] K. Koch. How does ASCI Actually Complete Multi-month 1000-processor Milestone Simulations? In *Proceedings of the Conference on High Speed Computing*, Gleneden Beach, Oregon, 2002.

[17] Argonne National Laboratory. Mpich-a portable implementation of mpi. http://www-unix.mcs.anl.gov/mpi/mpich/.

[18] Argonne National Laboratory. Zeptoos: The small linux for big computers. http://www-unix.mcs.anl.gov/zeptoos/.

[19] MOLAR: Modular Linux, Adaptive Runtime Support for High-end Computing Operating, and Runtime Systems. http://forge-fre.ornl.gov/molar/.

[20] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. Technical Report OSU-CISRC-2/06-TR22, The Ohio State University, Feb 2006.

[21] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *Proceedings of 17th Annual ACM International Conference on Supercomputing (ICS '03)*, June 2003.

[22] B. Maccabe, P. G. Bridges, R. Brightwell, R. Riesen, and T. Hudson. Highly Configurable Operating Systems for Ultrascale Systems. In *Proceedings of the First International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters*, 2004.

[23] Mellanox Technologies. http://www.mellanox.com.

[24] Mellanox Technologies. Mellanox IB-Verbs API (VAPI), Rev. 1.00.

[25] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel. Diagnosing Performance Overheads in the Xen Virtual Machine Environment. In *Proceedings of the First ACM/Usenix Conference on Virtual Execution Environments (VEE'05)*, June 2005.

[26] Myricom, Inc. Myrinet. http://www.myri.com.

[27] NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.

[28] University of Wisconsin. Condor High Throughput Computing. http://www.cs.wisc.edu/condor/.

[29] Open InfiniBand Alliance. http://www.openib.org.

[30] OProfile. http://oprofile.sourceforge.net.

[31] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In *Proceedings of the Supercomputing*, 1995.

[32] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib.org/benchmark/hpl/.

[33] F. Petrini, D. J. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *Proceedings of SC '03*, Washington, DC, USA, 2003.

[34] I. Pratt. Xen Virtualization. Linux World 2005 Virtualization BOF Presentation.

[35] I. Pratt and K. Fraser. Arsenic: A User-Accessible Gigabit Ethernet Interface. In *INFOCOM*, pages 67–76, 2001.

[36] I. Pratt, K. Fraser, S. Hand, C. Limpach, and A. Warfield. Xen 3.0 and the Art of Virtualization. In *Proceedings of Linux Symposium*, 2005.

[37] Quadrics, Ltd. QsNet. http://www.quadrics.com.

[38] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accountable execution of untrusted programs. In *Workshop on Hot Topics in Operating Systems*, pages 136–141, 1999.

[39] M. Rosenblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer*, 38(5):39–47, 2005.

[40] HPC-Colony Project: Services and Interfaces for Very Large Linux Clusters. http://www.hpc-colony.org/.

[41] Top 500 Supercomputer Site. http://www.top500.com.

[42] J. Sugerman, G. Venkitachalam, and B. H. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of USENIX*, 2001.

[43] H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. In *Proceedings of the 12th International Parallel Processing Symposium*, 1998.

[44] FastOS: Forum to Address Scalable Technology for runtime and Operating Systems. http://www.cs.unm.edu/ fastos/.

[45] ttylinux. http://www.minimalinux.org/.

[46] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *ACM Symposium on Operating Systems Principles*, 1995.

[47] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *International Symposium on Computer Architecture*, pages 256–266, 1992.

[48] C. Waldspurger. Memory resource management in VMware ESX server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, 2002.

[49] MVAPICH Project Website. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/index.html.

[50] A. Whitaker, R. Cox, M. Shaw, and S. Gribble. Constructing services with interposable virtual hardware. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*, 2004.

[51] A. Whitaker, M. Shaw, and S. D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Technical report, University of Washington, February 2002.

[52] IETF IPoIB Workgroup. http://www.ietf.org/html.charters/ipoib-charter.html.

[53] W. Yu, J. Wu, and D. K. Panda. Fast and Scalable Startup of MPI Programs In InfiniBand Clusters. In *Proceedings of the International Conference on High Performance Computing '04*, Banglore, Inida, December 2004.