ARCHITECTURAL PRINCIPLES AND EXPERIMENTATION OF DISTRIBUTED HIGH PERFORMANCE VIRTUAL CLUSTERS

Andrew J. Younge

Submitted to the faculty of

Indiana University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in the School of Informatics and Computing

Indiana University

February 2017

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Geoffrey C. Fox, Ph.D, Chair

Judy Qiu, Ph.D

Thomas Sterling, Ph.D

D. Martin Swany, Ph.D

October 10th, 2016

ACKNOWLEDGMENTS

I am grateful to many, many people for making this dissertation possible.

First, I want to express my sincere gratitude and admiration to my advisor, Geoffrey Fox. Geoffrey has provided invaluable advice, knowledge, insight, and direction in a way that made this dissertation possible. It has been a distinct honor and privilege working with him, something which I will carry with me throughout the rest of my career.

I would like to thank my committee members, Judy Qiu, Thomas Sterling, and Martin Swany. The guidance received in the formation of this dissertation has been of great help and a fruitful learning experience. Conversations with each member throughout the dissertation writing have proven exceptionally insightful, and I have learned a great deal working with each of them. I also want to explicitly thank Judy Qiu for giving me the opportunity to help organize and lecture in the Distributed Systems and Cloud Computing courses over the years. This teaching experience has taught me how to be both a better lecturer and mentor.

Over the years I have had the pleasure of working with a number of members in the Digital Science Center lab and others throughout Indiana University. I would like to thank Javier Diaz-Montes, Thilina Ekanayaka, Saliya Ekanayaka, Xiaoming Gao, Robert Henschel, Supun Kamburugamuve, Nate Keith, Gary Miksik, Jerome Mitchell, Julie Overfield, Greg Pike, Allan Streib, Koji Tanaka, Gregor von Laszewski, Fugang Wang, Stephen Wu, and Yudou

Zhou for their support over the years. Many of these individuals have helped me directly throughout my tenure at Indiana University and their efforts and encouragement is greatly appreciated.

I would like to express my gratitude to the APEX research group at the University Of Southern California Information Sciences Institute, where I was a visiting researcher and intern in 2012 and 2013. In this, I would like to give a special thanks to John Paul Walters at USC/ISI who worked directly with me on many of the research endeavors described within this dissertation. He has provided hands-on mentorship that aided with the basis of this dissertation, and his leadership is one in which I look to emulate in the future. I would also like to thank the Persistent Systems Fellowship through the School of Informatics and Computing for providing me with a fellowship to fund the last three years of my PhD.

I would also like to give thanks to the many close friends and roommates in Bloomington, Indiana over the years. Specifically, Id like to thank Ahmed El-Hassany, Jason Hemann, Matthew Jaffee, Aubrey Kelly, Haley MacLeod, Brent McGill, Andrea Sorensen, Dane Sorensen, Larice Thoroughgood, and Miao Zhang. Id like to make a special acknowledgement to my friends in the Bloomington cycling community, many of whom have become my Bloomington pseudo-family. This includes John Brooks, Michael Wallis Jr, Kasey Poracky, Brian Gessler, Niki Gessler, Davy McDonald, Jess Bare, Ian Yarbrough, Kelsey Thetonia, Rob Slover, Pete Stuttgen, Mike Fox, Tom Schwoegler, and the Chi Omega Little 500 cycling team. In this, I specifically need to acknowledge two wonderful people, Larice thoroughgood and John Brooks for their support, patience, and forbearance during the final stages in of my Ph.D. Larice has provided the love and encouragement necessary to see this dissertation through

to the end. John has donated countless hours of his time helping me edit and revise this manuscript, along with many after-hour philosophical discussions that proved crutial to the work at hand. Without all of these wonderful people in my life here in Bloomington, this dissertation would have likely been completed many months earlier.

Last but not least, I want to thank my parents Mary and Wayne Younge, my sisters Bethany Younge and Elizabeth Keller, and my extended family for their unwavering love and support over the years. Words cannot fully express the gratitude and love I have for you all.

Andrew J. Younge

ARCHITECTURAL PRINCIPLES AND EXPERIMENTATION OF DISTRIBUTED HIGH PERFORMANCE VIRTUAL CLUSTERS

With the advent of virtualization and Infrastructure-as-a-Service (IaaS), the broader scientific computing community is considering the use of clouds for their scientific computing needs. This is due to the relative scalability, ease of use, advanced user environment customization abilities, and the many novel computing paradigms available for data-intensive applications. However, a notable performance gap exists between IaaS and typical high performance computing (HPC) resources. This has limited the applicability of IaaS for many potential users, not only for those who look to leverage the benefits of virtualization with traditional scientific computing applications, but also for the growing number of big data scientists whose platforms are unable to build on HPCs advanced hardware resources.

Concurrently, we are at the forefront of a convergence in infrastructure between Big Data and HPC, the implications of which suggest that a unified distributed computing architecture could provide computing and storage ca-

pabilities for both differing distributed systems use cases. This dissertation proposes such an endeavor by leveraging the performance and advanced hardware from the HPC community and providing it in a virtualized infrastructure using High Performance Virtual Clusters. This will not only enable a more diverse user environment within supercomputing applications, but also bring increased performance and capabilities to big data platform services.

The project begins with an evaluation of current hypervisors and their viability to run HPC workloads within current infrastructure, which helps define existing performance gaps. Next, mechanisms to enable the use of specialized hardware available in many HPC resources are uncovered, which include advanced accelerators like the Nvidia GPUs and high-speed, low-latency InfiniBand interconnects. The virtualized infrastructure that developed, which leverages such specialized HPC hardware and utilizes best-practices in virtualization using KVM, supports advanced scientific computations common in today‘s HPC systems. Specifically, we find that example Molecular Dynamics simulations can run at near-native performance, with only a 1-2% overhead in our virtual cluster. These advances are incorporated into a framework for constructing distributed virtual clusters using the OpenStack cloud infrastructure. With high performance virtual clusters, we look to support a broad range of scientific computing challenges, from HPC simulations to big data analytics with a single, unified infrastructure.

Geoffrey C. Fox, Ph.D, Chair

Judy Qiu, Ph.D

Thomas Sterling, Ph.D

D. Martin Swany, Ph.D

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

For years, visionaries in computer science have predicted the advent of utility-based computing. This concept dates back to John McCarthy's vision stated at the MIT centennial celebrations in 1961:

> "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry."

Only recently has the hardware and software become available to support the concept of utility computing on a large scale.

The concepts inspired by the notion of utility computing have combined with the requirements and standards of Web 2.0 [1] to create Cloud computing [2–4]. Cloud computing is defined as "A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet." This concept of cloud computing is important to Distributed Systems because it represents a true paradigm shift [5]

within the entire IT infrastructure. Instead of adopting the in-house services, client-server model, and mainframes, clouds push resources out into abstracted services hosted *en masse* by larger organizations. This concept of distributing resources is similar to many of the visions of the Internet itself, which is where the "clouds" nomenclature originated, as many people depicted the internet as a big fluffy cloud one connects to.

At the core of most cloud infrastructure lies virtualization, a computer architecture technology by which 1 or more Virtual Machines (VMs) are run on the same physical host. In doing this, a layer of abstraction is inserted between and around the hardware and Operating System (OS). Specifically, hardware resources such as CPUs, memory, and I/O devices, and software resources analagous to OS functionality and low level libraries are abstracted and provided to VMs directly. Virtualization has existed for many years, but its availability with Intel x86 commodity hardware in conjunction with the rise of clouds has brought it to the forefront of distributed systems.

While cloud computing is changing IT infrastructure, it also has had a drastic impact on distributed systems as a field, which has a different evolution. Gone are the IBM Mainframes of the 1980s, which dominated the enterprise landscape. While some mainframes still exist, today they are used only for batch related processing tasks and not for scientific applications as they are inefficient at Floating Point Operations. As such, Beowulf Clusters [6], Massively Parallel Processors (MPPs) and Supercomputers of the 90s and 00s replaced the mainframes of before. A novelty of these distributed memory systems is that instead of just one large machine, many machines are connected together to achieve a common goal, thereby maximizing the overall speed of computation. Clusters represent a more commodity-based supercomputer with off-the-shelf CPUs instead of the highly customized and expensive processors and interconnects found in Supercomputers. Supercomputers and Clus-

ters are best suited for large-scale applications, due to their innate ability to divide the computational efforts into smaller concurrent tasks. These tasks often run in parallel on many CPUs, whereby each task communicates results to other tasks in some way as per the application design. These HPC applications can even include "Grand Challenge" applications [7] and can represent a sizable amount of the scientific calculations done on large-scale Supercomputing resources today. However, there exists a gap of many orders of magnitude between leading-class high performance computing and what is available on the common laboratory workshop. This gap, described here as mid-tier scientific computation, is a fast growing field that struggles to harness distributed systems efficiently while hoping to minimize extensive development efforts. These mid-tier scientific endeavors need to leverage distributed systems to complete the calculations at hand effectively, however, they may not require the extreme scale provided by the latest machines at the peak of the Top500 list [8]. Some small research teams may not have the resources available or the desire to handle the development complexity of high end supercomputing systems, and can look towards other options instead. This can include some scientific disciplines such as high energy physics [9], materials science [10], bioinformatics [11], and climate research [12], to name a few.

As more domain science turns to the aid of computational resources for conducting novel scientific endeavours, there is a continuing and growing need for national cyberinfrastructure initiatives to support an increasingly diverse set of scientific workloads. Historically, there have been a number of national and international cyberinfrastructure efforts to support high end computing. These range from traditional supercomputers deployed at Department of Energy computing facilities [13], to efforts led by the National Science Foundation such as the XSEDE environment [14] or domain specific initiatives such as the NSF iPlant collaborative [15].Substantial growth can be seen in the number of computational resource requests [14,16] from many of the larger

computational facilities. Concurrently, there has also been an increase in accelerators and hybrid computing models capable of quickly providing additional resources [17] beyond commodity clusters.

Historically, application diversity was separated into High Performance Computing (HPC) and High Throughput Computing (HTC). With HTC, computational problems can be split into independent tasks that execute in a pleasingly parallel fashion, happily gaining any available resources and rarely requiring significant communication or synchronization between tasks. Example of this can include independent tasks or parameter sweeps of an application to find the optimal application settings, or many similar tasks with only slightly different input datasets. HPC often represents computational problems that require significant communication and coordination to produce results effectively, usually with the use of a communication protocol such as MPI [18]. These applications are often tightly coupled sequential tasks operating concurrently and communicating often to complete the collective computational problem. While there are many examples of such HPC applications, common examples often involve significant matrix and/or vector multiplication mechanisms [19] for completing simulations.

However, many big-data paradigms [20] in distributed systems have been introduced that represent new computational models for distributed computing, such as MapReduce [21] and the corresponding Apache Big Data stack [22, 23]. While the term big data can mean a number of things, it generally refers to data sets large or complex enough to require advanced, non-traditional data processing mechanisms in order to extract feature knowledge. Some HTC and HPC applications could potentially be considered big data tools, generally the nomenclature has grown out of new industry and academic problem sets created by a data deluge [24]. This has lead to a number of novel platform services for handling big data problems in parallel, and

supporting these different distributed computational paradigms requires a flexible infrastructure capable of providing computational resources for all possible models in a fast and efficient manner.

Currently, we are at the forefront of a convergence within scientific computing between HPC and big data computation [25]. This amalgamation of historically differing viewpoints of Distributed Systems looks to combine the performance characteristics of HPC and the pursuit towards Exascale with the data and programmer oriented concurrency models found in Big Data and cloud services.

Much of the convergence effort has been focused on applications and platform services. Specifically, significant work towards convergent applications has been outlined with the Big Data Ogres [26] and the corresponding HPC-ABDS model [27]. This convergence can also be seen with efforts in bringing interconnect advances to classically big data platforms such as with InfiniBand and MapReduce [28]. However, the underlying hardware and OS environments are still something to be reconciled, which is something that virtualization can potentially help provide. It is expected that new big data efforts will continue to move in this direction [29], especially if virtualization can make HPC hardware that's traditionally prohibitive in such areas, such as accelerators and high-speed interconnects, readily available to cloud and big data platforms. As the deployment of big data applications and platform services on virtualized infrastructure is well defined and studied [30], this work instead focuses on the difficulty of running HPC applications on similar virtualized infrastructure. However, it is possible and hopeful that research regarding virtualization can also play a part in bringing advanced hardware and performance-focused considerations to Big Data applications, effectively cross-cutting the convergence with HPC. In Summary, success of the research in virtualization could be defined by the ability to support the convergence between HPC and Big Data.

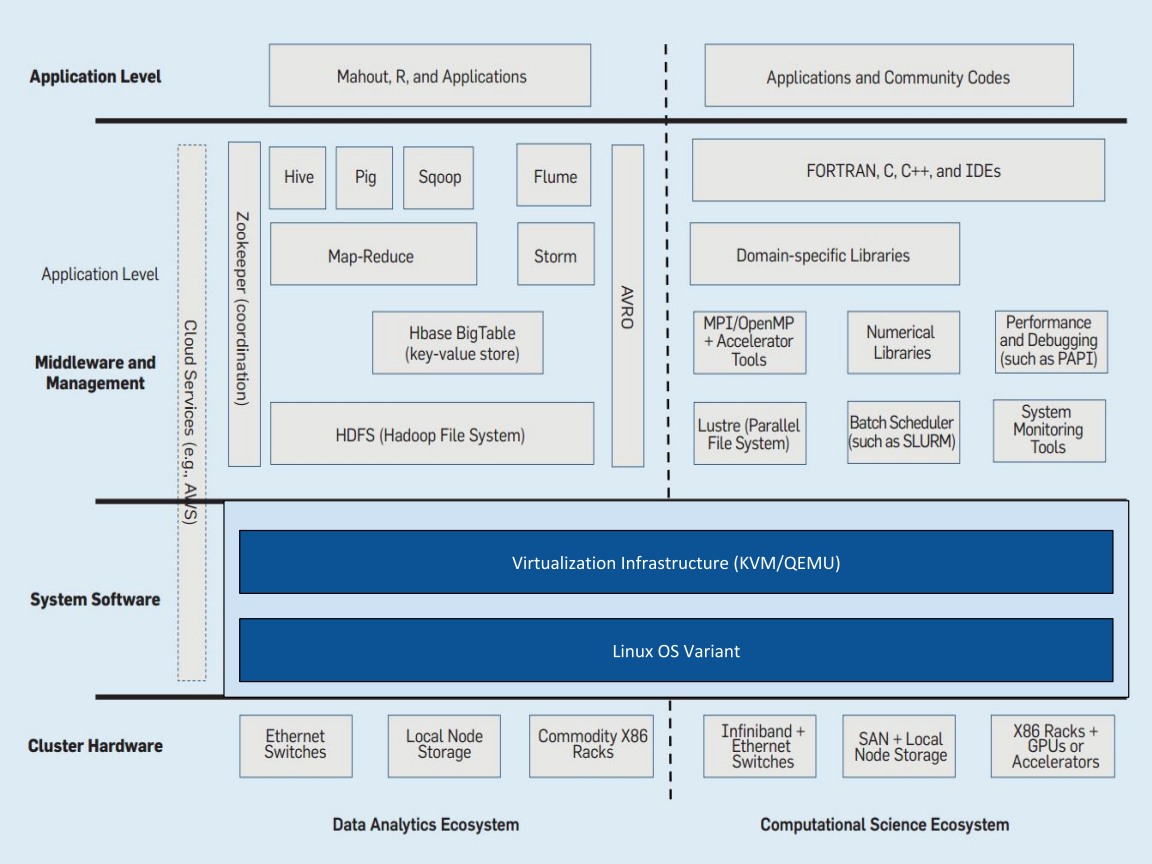


Figure 1.1 Data analytics and computing ecosystem compared (from [25]), with virtualization included

To further illustrate where virtualization can play a part in HPC and Big Data convergence, we look at Figure 1.1 from Reed & Dongarra [25]. While the two ecosystems depicted are only representative and in no way exhaustive, they do show how drastically different user environments are and how reliant they are on differing hardware. If we insert a performance-oriented virtualization mechanism within the system software capable of handling the advanced cluster hardware performing at near-native speeds (at or under 5% overhead, as loosely defined in [31]), it could provide a single, comprehensive *convergent ecosystem* that supports both HPC and Big Data efforts at a critical level.

This work proposes the use of virtual clusters [32] to provide distinct, separate

environments on similar resources using virtual machines. These virtual clusters, similar in design to the Beowulf clusters and commodity HPC systems, provide a distributed memory environment, usually with a local resource management system [33]. However, past concerns with virtualization have limited the adoption of virtual clusters in many large-scale cyberinfrastructure deployments. This has in part been due to the overhead of virtualization, whereby many scientific computations have experienced a significant and notable degradation in performance. In an ecosystem familiar with HPC systems in which performance is paramount, this has been an obstructive hurdle for deploying many tightly coupled applications.

1.2 Research Statement

With the rise of cloud computing within the greater realm of distributed systems, there have been a number of scientific computing endeavors that map well to cloud infrastructure. This first includes the simple and most common practice of on-demand computing whereby users can rent-a-workstation [34]. Perhaps these resources are more powerful than a given researcher's laptop and used to run their scientific applications or support greater laboratory collaborative efforts, such as a shared database or Web services. We have also seen virtualized cloud infrastructure support high throughput computing very well. Often times pleasingly parallel applications, be it from high energy physics such as the LHC effort [9,35] or bioinformatics with BLAST alignment jobs [11], have proven to run with high efficiency in public and private cloud environments. Furthermore, the rise of public cloud infrastructure has also coincided with increase in big data computation and analytics. Many of these big data platform services have evolved complimentary to cloud infrastructure, and as such have a symbiotic relationship with virtualization technologies [36].

However, with tightly coupled, high performance distributed memory applications,

the same endeavors that support leading class scientific efforts, run very poorly on virtualized cloud infrastructure [37]. This is due to a myriad of addressable reasons ranging from scheduling, abstraction overhead, and a lack of advanced hardware support necessary for tightly coupled communication. This postulates the question regarding whether virtualization can in fact support such tightly coupled large-scale applications without an imposed significant performance penalty. Simply put, *the goal of this dissertation is to investigate the viability of mid-tier scientific applications supported in virtualized infrastructure.*

Historically, mid-tier scientific applications are distributed memory HPC applications that require more complex process communication mechanisms. These systems need far more performance than a single compute resource (such as a workstation) can provide. This could include hundreds or thousands of processes calculating and communicating concurrently on a cluster, perhaps using a messaging interface such as MPI. These applications can be distinct (and sometimes simpler), either in application composition or operating parameters, from extreme-scale HPC applications that run at the highest end of the supercomputing resources today operating on petascale machines and beyond.

Given the current outlook on virtualization for supporting HPC applications, this dissertation proposes a framework for High Performance Virtual Clusters that enable advanced computational workloads, including tightly coupled distributed memory applications, to run with a high degree of efficiency in virtualized environments. This framework, outlined in Figure 1.2, illustrates the topics to be addressed to provide a supportive virtual cluster environment for high performance mid-tier scientific applications. Areas marked in darker green indicate topics this dissertation may touch upon, whereas light green areas in Figure 1.2 identify outstanding considerations to be investigated. Specifically, mid-tier distributed memory parallel computations is

identified as a focal point for the computational challenges at hand as a way to separate from some of the latest efforts towards Exascale computing [38–40]. While virtualization may in fact be able to play a role towards usable Exascale computing, such efforts fall outside the immediate scope of this dissertation.

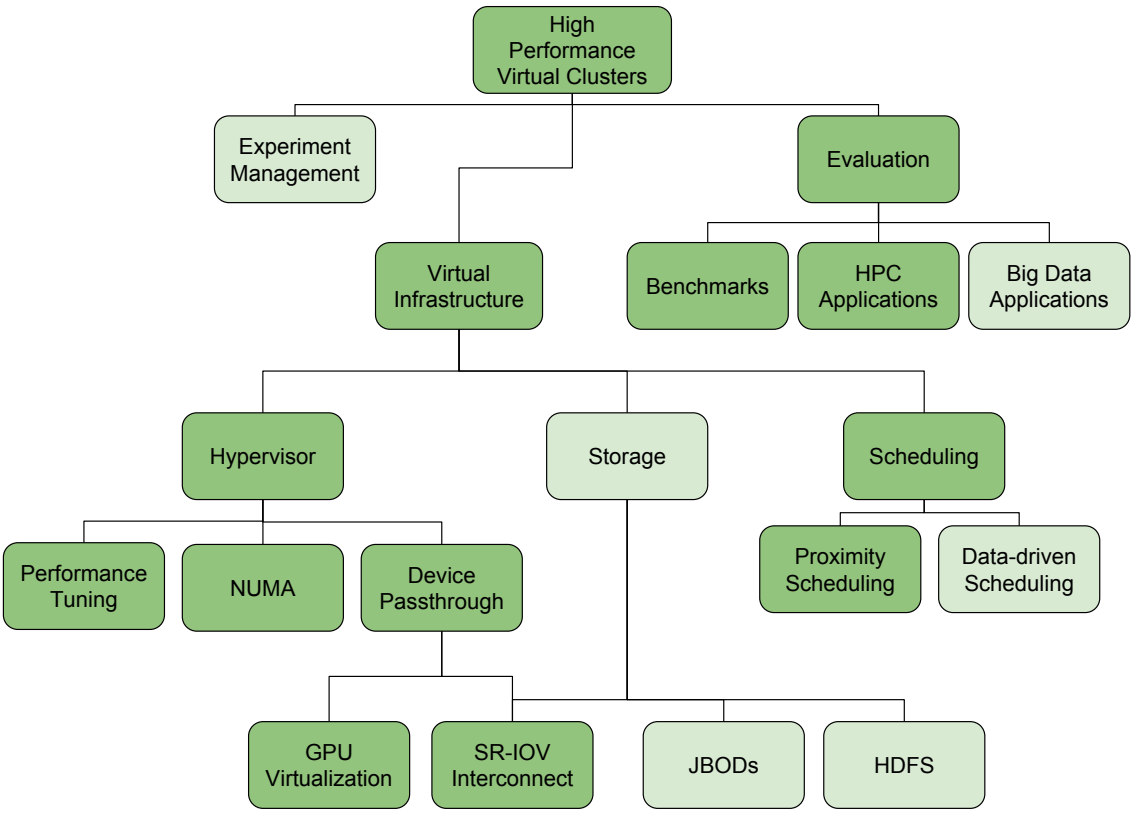


Figure 1.2 High Performance Virtual Cluster Framework

In order to provide high performance virtual clusters, there is a need to first look at a key area, the virtualized infrastructure itself. At the core, the hypervisor, or virtual machine monitor, has to be considered and the overhead and performance characteristics associated with it. This includes performance tuning considerations, non-uniform memory access (NUMA) effects, and advanced hardware device passthrough. Specifically, device passthrough in the context of this manuscript refers to two major device types; GPUs and InfiniBand interconnects (the later using SR-IOV). The virtual in-

frastructure also must consider scheduling as a major factor in performing efficient placement of workloads on the underlying host infrastructure, and in particular a Proximity scheduler is of interest [41]. Storage solutions in a virtualized environment is an increasingly important aspect of this framework, as some HPC and big data solutions are prioritizing I/O performance more than computation. Storage is also likely to be heavily dependent on interconnect considerations as well, which could be potentially provided by device passthrough. However, such I/O considerations lie beyond this dissertation's immediate scope.

However, simply providing an enhanced virtualized infrastructure may not guarantee that all implementations of high performance virtual clusters are performant. Specifically, proposed infrastructures need to be properly evaluated in a systematic way through the use of a wide array of benchmarks, mini-applications, and full-scale scientific applications. This effort can further be separated into three major problem sets; base level benchmarking tools, HPC applications, and big data applications. Evaluating the stringent performance requirements of all three sets, when compared with bare metal (no virtualization) solutions, will illuminate not only successful designs but also the focus areas that require more attention. As such, we look to continually use these benchmarks and applications as a tool to measure the viability of virtualization in this context.

Fundamentally, this framework is leveraging these real-world applications and benchmarks predominantly for the evaluation of various solutions under strong scaling conditions. Specifically, strong scaling is used to apply a fixed problem size (or predefined set of fixed problem sizes) that look to match real-world running conditions, and vary the number of processors and execution units (including GPUs) across multiple virtualized and non-virtualized architectures. Generally, a successful solution will demonstrate strong scaling characteristics as close to a native, bare-metal

runtime as possible, illustrating little overhead of the added methods. Furthermore, variability between runtimes are also important to consider, as high variability and noise within large scale distributed systems can lead to tail latencies and significant efficiency losses. While weak scaling experiments that increase the problem size relative to the number of available processors could be used for future evaluation, such efforts are outside of the immediate scope of this dissertation.

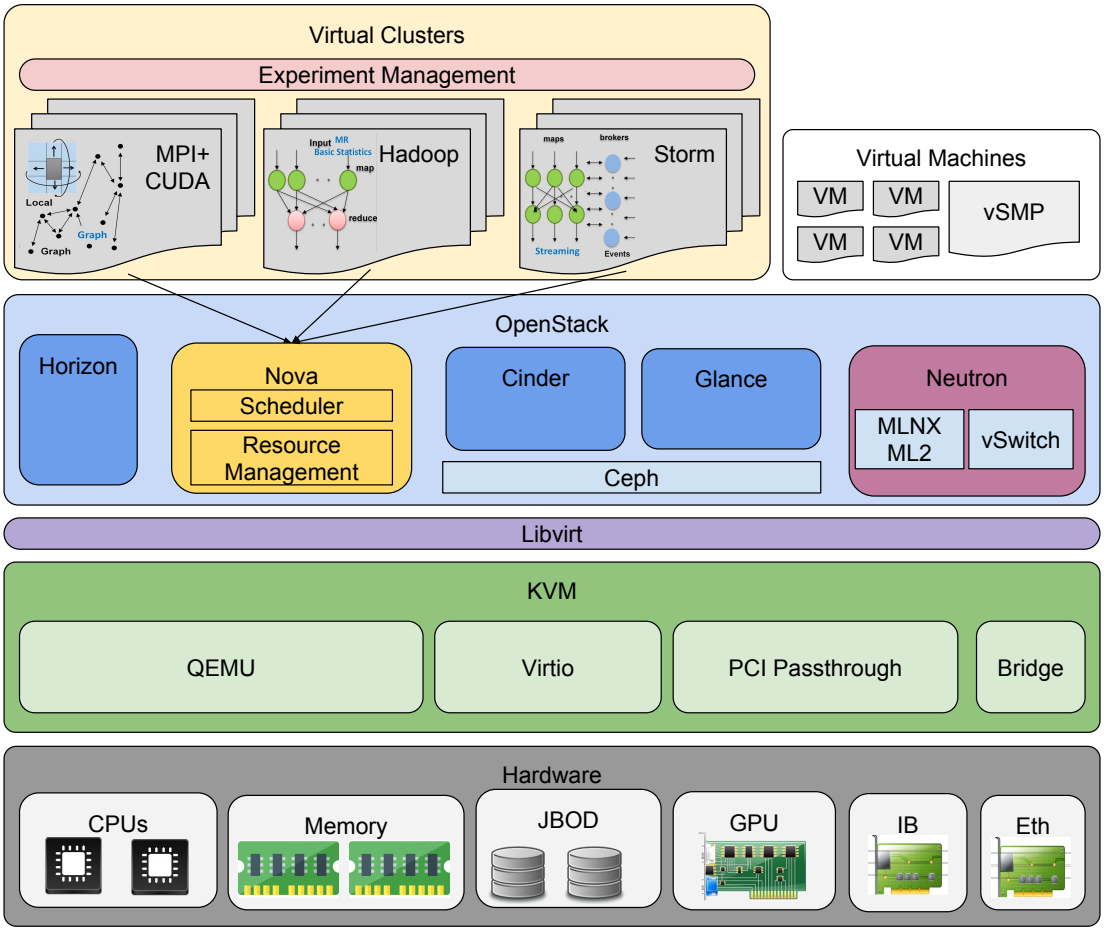


Figure 1.3 Architectural diagram for High Performance Virtual Clusters

Starting from the historical perspective of virtual clusters in Grid computing, we see the new architectural model for high performance virtual clusters illustrated in Figure 1.3 that is implied by the efforts described in this dissertation. Here, we

leverage commodity hardware, as well as some advanced HPC hardware withint he same system. Providing advanced hardware in this scenario has a twofold effect. First, we look to provide classic distributed memory HPC applications in an virtualized environment that still utilize the same hardware that such applications have grown accustomed to. Second, it allows for novel big data analytics and cloud platform services to move towards hardware that may drastically speed up their computational efforts. Specifically, we allow for such platform services to leverage the high-speed, low-latency interconnects that HPC systems have relied upon. While such cloud systems may not be able to take full advantage of the latency improvements, it is estimated the the increased bandwidth may have a significant impact on the overall runtime of these systems, reducing the overall time-to-solution. While this notion could incorporate a wide array of differing technologies, we focus here on GPU-based accelerators and InfiniBand interconnects in conjunction with x86 CPUs.

From this diverse yet highly optimized hardware, we leverage KVM and QEMU to provide an advanced hypervisor for creating and hosting VMs with direct hardware involvement from the lower level. Throughout this dissertation, the utilization of this diverse hardware can be wrangled and virtualized through the use of KVM and QEMU. This includes direct virtualization, para-virtualization with QEMU, and even direct passthrough mechanisms. Moving up, the Libvirt API is leveraged due to its hypervisor interoperability and popularity. Modifications are made throughout this work to enable Libvirt to support passthrough mechanisms in both Xen and KVM, changes which since have since made their way to libvirt upstream. Providing Libvirt as a virtualization API also insulates from higher level changes, whereby other resource management solutions could be swapped in if such a design was warranted.

Building form Libvirt comes the OpenStack private cloud infrastructure. In Figure 1.3 we illustrate some (but not all) of OpenStack's services including the Horizon

UI, Cinder and Glance storage mechanisms, and the Neutron (previously Quantum) components. The Nova component of OpenStack is the point of focus for providing comprehensive VM management. [1] OpenStack provides a comprehensive cloud infrastructure service for managing up to thousands of nodes in a cluster using virtualization. Its integration with KVM through Libvirt is highly tuned for efficient VM deployment and advanced plug-ins such as the ML2 Neutron service can managed an advanced interconnect such as InfniBand and 40GbE provided by Mellanox.

Atop OpenStack, we can create a wide array of virtual clusters and machines to support diverse scientific computing ecosystems necessary. This includes application models ranging from tightly coupled MPI+CUDA HPC applications, to emerging big data analytics toolkits such as Apache Storm [42]. Such virtual clusters can be built manually through users requesting highly tuned images from a repository and deploying them to scale. This also allows for the use of virtualized SMP systems such as ScaleMP [43].

One of the higher-level aspects of providing high performance virtual clusters is the orchestration of the virtual clusters themselves, which could be called experiment management. While this largely remains tangential to this immediate research, it is nonetheless a key aspect for a successful solution. Some effort has been put forth for virtual cluster experiment management [44], and many ongoing open sources solutions also offer compelling options, such as OpenStack Heat [45]. An example of a project delivering advanced orchestration mechanisms and a toolkit to aid in configurable virtual clusters on heterogeneous IaaS deployments is the Cloudmesh effort [46].

[1] While many of the features for nova's additions in GPU Passthrough and SR-IOV InfiniBand support have been put together at USC/ISI as an OpenStack Nova fork (https://libraries.io/github/usc-isi/nova), the features have since been modified and matured by the OpenStack community in later releases and made available in upstream Nova.

1.3 Research Challenges

The framework, architecture, and efforts described in this dissertation represent a movement forward in providing virtualized infrastructure to support a wide arrange of scientific applications. However, there still exist some challenges that will need to be addressed. This includes a stigma of virtualization being inherently slow and unable to support tightly coupled computations, limitations associated with operating at scale, and even that containers may provide a better alternative. While this work hopes to move beyond these challenges, they none the less must be considered.

The notion that virtualization and Cloud infrastructure are not able to support parallel distributed memory applications has been characterized many times. One of the most prominent examples of this is the Department of Energy's Magellan Project [47], whereby the Magellan Final Report [48] states the following finding as a Key Finding:

> "**Finding 2. Scientific applications with minimal communication and I/O are best suited for clouds.**
>
> We have used a range of application benchmarks and micro-benchmarks to understand the performance of scientific applications. Performance of tightly coupled applications running on virtualized clouds using commodity networks can be significantly lower than on clusters optimized for these workloads. This can be true at even mid-range computing scales. For example, we observed slowdowns of about 50x for PARATEC on the Amazon EC2 instances compared to Magellan bare-metal (non-virtualized) at 64 cores and about 7x slower at 1024 cores on Amazon Cluster Compute instances, the specialized high performance computing offering. As a result, current cloud systems are best suited for high-throughput, loosely coupled

applications with modest data requirements.“

These findings underscore how classical usage of virtualization in cloud infrastructure has serious performance issues when running tightly coupled distributed memory applications. Many of these performance concerns are sound, given the limitation of a number of virtualization overheads commonplace at the time, including shadow page tables, emulated Ethernet drivers, experimental hypervisors, and a complete lack of sophisticated hardware commonplace in supercomputers and clusters. As a result, the advantages of virtualization, including on-demand resource allocation, live migration and advanced hybrid migration, and user-defined environments, have not been able to show effectively their value in the context of the HPC community.

Other and related efforts within the scientific community too found limitations with HPC applications in public cloud environments. This includes the study by Jackson et. al [49] which illustrates how the Amazon Elastic Compute Cloud (EC2) creates a 6x performance impact compared to a local cluster, due in large part to the limiting Gigabit Ethernet on which benchmarks relied heavily within the EC2 system. Other studies also found similar results; Ostermann [37] for instance, concludes that Amazon EC2 “is insufficient for scientific computing at large, though it still appeals to the scientists that need resources immediately and temporarily.” However, these studies are now outdated and do not take into account the hardware and advancements in virtualization detailed in this dissertation. Specifically, it is estimated that with the KVM hypervisor in a performance-tuned environment, using accelerators and most certainly a high-speed, low latency interconnect as detailed in Chapter 6, the results would be drastically different.

One limitation in this research in high performance virtual clusters is the fact that the degree to which applications can scale remains relatively unknown. While initial results with SR-IOV InfiniBand are promising, scaling is naturally hard to predict. It

would be hypothetically possible that as the number of VM's increases, interconnect communication tail-latency could also increase, causing notable slowdowns during distributed memory synchronization barriers. It is only when infrastructure able to support high performance virtual clusters becomes available will scaling to thousands of cores and beyond be investigated.

Another potential challenge, and perhaps also a strength, is the rising use of containers within industry, such as we see in efforts like Docker [50]. Recently, we have seen efforts at NERSC/LBNL adapt a container solution called Shifter with the SLURM resource manager on CRAY XC based systems [51]. Shifter's goal is to provide user defined images for NERSC's bioinformatics users, and it adapts remarkably well to the HPC environment. While further efforts are needed by Cray and NERSC to fully provide a container-based solution on a large-scale Supercomputer for all applications, its efforts are in many ways related to virtualization. In Chapter 5, we specifically compare virtualization efforts with LXC [52], a popular Linux container solution, and find performance to be comparable and largely near-native.

While the major concern with virtualization in the HPC community is performance issues, virtualization itself may not be fundamentally limited by the overhead that causes issues in running high performance computing applications. Recent improvements in performance, along with increased usability of accelerators and high speed, low latency interconnects in virtualized environments, as demonstrated in this dissertation, have made virtual clusters a more viable solution for mid-tier scientific applications. Furthermore, it is possible for virtualization technologies to bring enhanced usability and enable specialized runtime components to future HPC resources, adding significant value over today's supercomputing resources. This could potentially include infrastructure advances for higher level cloud platform services for supporting big data applications [27].

1.4 Outline

The rest of this dissertation is organized into chapters, each signifying the steps to move forward the notion of a high performance virtual cluster solution.

Chapter 2 investigates the related research surrounding both cloud computing and high performance computing. Within cloud computing, an introduction to cloud infrastructure, virtualization, and containers will all be discussed. This also includes details regarding virtual clusters as well as an overview of some national scale cloud infrastructure efforts that exist. Furthermore, we investigate the state of high performance computing and supercomputing, as well touch upon some of the current Exascale efforts.

Chapter 3 takes a look at the potential for virtualization, in a base case, to support high performance computing. This includes a feature comparison for hardware availability of a few common hypervisors, specifically Xen, KVM, VirtualBox, and VMWare. Then, a few common HPC benchmarks are evaluated in a single node configuration to determine what overhead exists and where. This identifies how in some scenarios, virtualization adds only a minor overhead, whereas with other scenarios, overheads can be up to 50% compared to native configurations.

Chapter 4 starts to overcome one of the main limitations of virtualization for use in advanced scientific computing, specifically the lack of hardware availability. In this chapter, The Xen hypervisor is used to demonstrate the effect of GPU passthrough, allowing for GPUs to be used in a guest VM. The efficiency of this method is briefly evaluated using two different hardware setups, and finds hardware can play a notable role in single node performance.

Chapter 5 continues where chapter 4 leaves off, by demonstrating that GPU passthrough is possible on many other hypervisors, specifically also KVM and VMWare,

and compares with one of the main containerization solutions, LXC. Here, the GPUs are evaluated using not only the SHOC GPU benchmark suite developed at Oak Ridge National Laboratory, but also a diverse mix of real-world applications in order to examine how and where overhead exists with GPUs in VMs for each virtualization setup. Specifically, we find that with properly tuned hardware and NUMA-balanced configurations, the KVM solution can perform at roughly near-native performance, with on average 1.5% overhead compared to no virtualization. This illustrates that with the correct hypervisor selection, careful tuning, and advanced hardware, scientific computations can be supported using virtualized hardware.

Chapter 6 takes the findings from the previous chapter to the next level. Specifically, the lessons learned from successful KVM virtualization with GPUs are expanded and combined with a missing key component of supporting advanced parallel computations: a high speed, low latency interconnect, specifically InfiniBand. Using SR-IOV and PCI passthrough of QDR InfiniBand interconnect across a small cluster, it is demonstrated that two Molecular Dynamics simulations, both very commonly used in the HPC community, can be run at near-native performance in the designed virtual cluster.

Chapter 7 takes a look at the given situation of virtualization, and puts forth an argument for enhancements forthcoming in high performance virtual cluster solutions. Specifically, we look at the given state of the art, how virtual clusters can be used to provide an infrastructure to support the convergence between HPC and big data. Specifically, this chapter outlines and investigates potential next steps for virtualization, including the potential for advanced live migration techniques and VM cloning, which can be made available with the inclusion of a high-performance RDMA-capable interconnect.

Finally, this work concludes with an overall view of the current state of high

performance virtualization, as well as its potential to impact and support a wide array of disciplines.

Chapter 2

Related Research

In order to depict the research presented in this article accurately, the topics within Virtualization, Cloud computing, and High Performance Computing are reviewed in detail.

2.1 Virtualization

Virtualization is a way to abstract the hardware and system resources from an operating system. This is typically performed within a Cloud environment across a large set of servers using a Hypervisor or Virtual Machine Monitor (VMM), which lies in between the hardware and the Operating System (OS). From here, one or more virtualized OSs can be started concurrently, as seen in Figure 2.1, leading to one of the key advantages of virtualization. This, along with the advent of multi-core processing capabilities, allows for a consolidation of resources within a data center. It then becomes the cloud infrastructure's job, as discussed in the next section, to exploit this capability to its maximum potential while still maintaining a given QoS. It should be noted that virtualization is not specific to Cloud computing. IBM originally pioneered the concept in the 1960's with the M44/44X systems. It has only recently been reintroduced for general use on x86 platforms.

However, virtualization, in the most general sense, is just another form of abstrac-

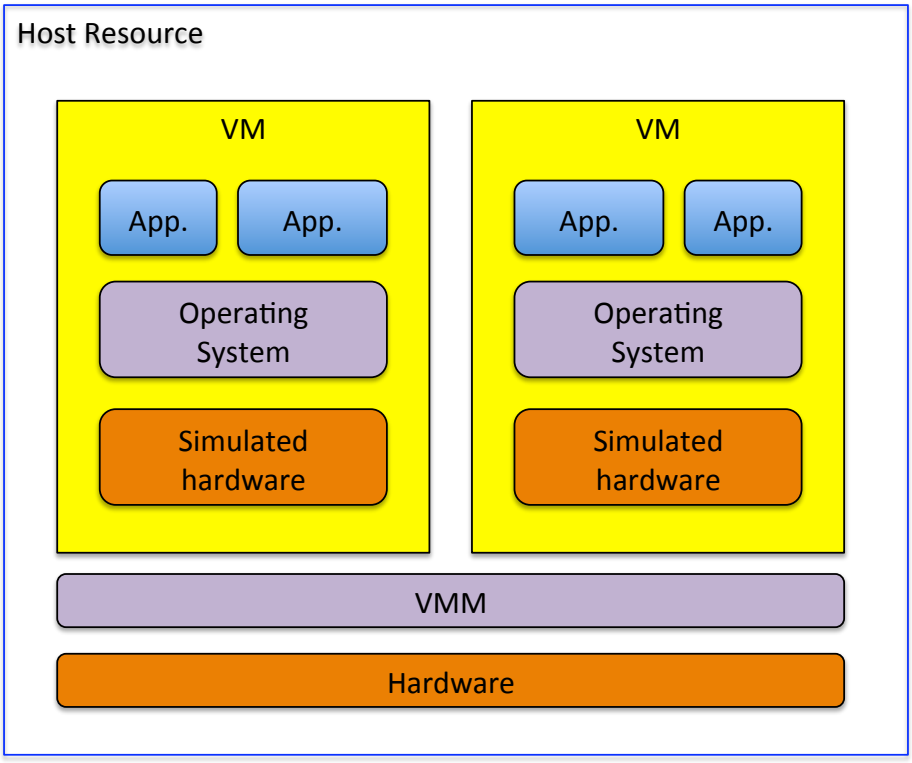


Figure 2.1 Virtual Machine Abstraction

tion. As such, there are in fact many levels to virtualization that exist [53].

- **ISA -** Virtualization can start from the instruction set architecture (ISA) level, whereby an entire processor instruction set is emulated. This may be useful for running software or services developed for one instruction set (say MIPS), but has to run on modern Intel x86 hardware. ISA level virtualization is usually emulated through an interpreter which translates source instructions to target instructions; however this can often be extremely inefficient. Dynamic binary translation can help aid in efficiency by translating blocks of source instructions to target instructions. This still can be limiting.

- **Hardware -** One of the most important technologies in cloud computing is hardware level virtualization [54, 55]. Hardware virtualization, in its most pure

form, refers to the process of creating virtual abstraction to hardware platforms, operating systems, or software resources. This enables the creation of 1 or more virtual machines (VMs) that are run concurrently on the same operating environment, be it hardware or some higher software. Here, a virtual hardware environment is generated for a VM, providing virtual processors, memory, and I/O devices that allow for VM multiplexing, as depicted in Figure 2.1. This layer also manages the physical hardware on behalf of a host OS, as well as for guests. While the most popular implementations of hardware virtualization is the Xen Virtual Machine Monitor [54]; this method has further separated into type 1 and type 2 hypervisors, as detailed further in Section 2.1.1.

- **Operating System -** Moving up the latter of implementation with virtualization, we find OS level virtualization, where multiplexing happens at the level of the OS, rather than the hardware. Usually, this refers to isolating a filesystem and associated process and runtime effects in a single "chroot" or *container* environment at the user level, where all system level operations are still handled by a single OS kernel. This containerization allows for multiple user environments to exist concurrently without the complexity of hardware or ISA level virtualization. Containers are also described in more detail in the next section.

- **Library (API) -** Moving up a layer, we find library or Application Level Interface (API) level virtualzation, where a programming interface is separated and the communication between this API and the back-end library with the computation is virtualized. This method removes the back end services from within the operating environment, which can give the effect that resources are local, when they are in fact remote. This is how GPUs are "virtualized" with

tools such as rCUDA [56] and vCUDA [57]. This method's performance often can be very dependent on the underlying communications mechanisms, as well as complete virtualization of the whole API (which may be difficult). API virtualization also exists for entire OS libraries to translate between differing OS, without actual containerization.

- **Process -** Virtualization can also take form at the process or user level, which can then be used to deliver a specialized or high level language that is the same across several different OSs. This is how the Java Virtual Machine (JVM) and Microsoft's .NET platform operate. This type of application predominantly falls outside of the scope of this dissertation.

2.1.1 Hypervisors and Containers

While there are many types of virtualization, this dissertation predominately focuses on hardware and OS level virtualization. With hardware virtualization, a Virtual Machine Monitor, or hypervisor, is used. Abstractly, a hypervisor is a piece of software that creates virtual machines or *guests*, usually that model the underlying physical machine's *host* system.

Hardware virtualization can actually be dissected in more detail to two different types of hypervisors. These hypervisor types are illustrated in Figure 2.2, and they can be directly compared to containers. With a Type 1 virtualization system, the hypervisor or VMM sits directly on the bare-metal hardware, below the OS. These native hypervisors provide direct control of the underlying hardware, and are controlled and operated usually through the use of a privileged VM. One example of a type 1 hypervisor is the Xen Virtual Machine Monitor [54], which uses its VMM as well as a privileged Linux OS, called Dom0, to create and manage other user DomU instances. The VMM in this place provides the necessary hardware abstraction of CPU, mem-

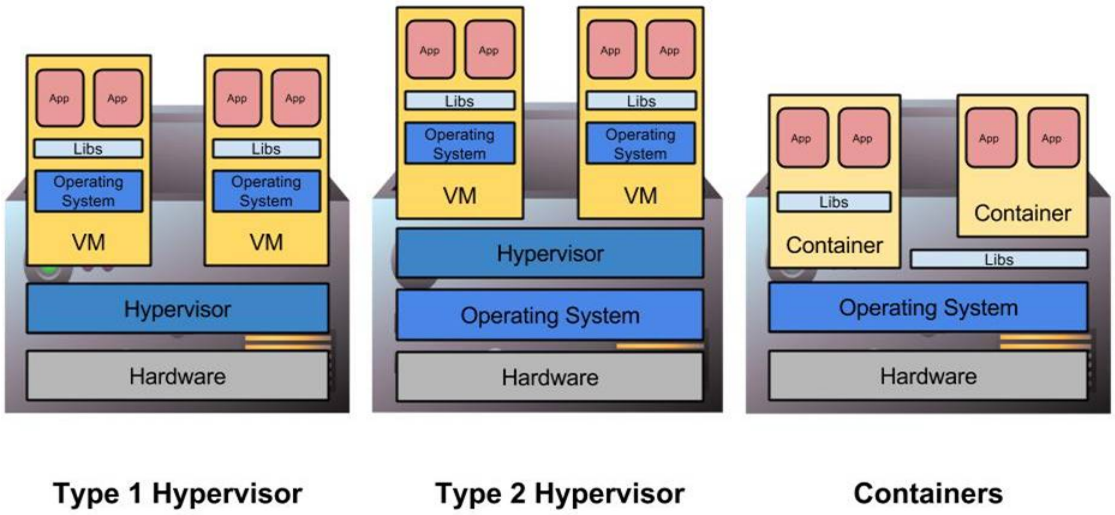


Figure 2.2 Hypervisors and Containers

ory, and some I/O aspects, leaving the control aspects of the other DomUs to Dom0. With a Type 1 hypervisor, all virtualization functions are kept separate from control and OS functionaity, effectively making a cleaner design. This design could lead to end application performance implications, as illustrated with the Palacios VMM [31].

Type 2 hypervisors utilize a different - and sometimes more convoluted - design. With a Type 2 hypervisor, there is a "host" OS that, like with native OSs, sits directly atop hardware. This OS is just like any normal native OS. However, the OS itself can abstract its own hardware, and provide and manage a VM, effectively as an OS process. In this case, the hypervisor providing the abstraction is effectively built within, atop, or as a module within the kernel. There are many different Type 2 hypervisors, the most common of which is the Linux Kernel Virtual Machine (KVM) [58]. KVM is often used in conjunction with QEMU, an ISA level emulator, to provide some basic ISA level virtualization and emulation capabilities. KVM is simply provided as a Linux kernel module within a given host, and guest VMs are run as a single process on the host OS.

Type 1 and Type 2 hypervisors are very distinct from OS level virtualization, also

known as containerization. With containers, there is a single OS; however, instead of direct hardware abstraction, a single kernel is used to simultaneously run multiple user-space instances in a jailed-root environment. These environments may look and feel like a separate machine, but in fact are not. Oftentimes the kernel itself provides resource management tools to help control resource utilization and allocations. Linux container (LXC) is a great example of this, with their use of namespaces for filesystem control and cgroups for resource management. With the recent advent of Docker, which looks to control versioning and easy deployment of traceable containers, this aspect of OS level virtualization has grown in popularity; however, security and usability concerns still exist, as dedicated isolation mechanisms do not exist and the kernel space is the only point of security separation.

2.2 Cloud Computing

Cloud computing [59] is one of the most explosively expanding technologies in the computing industry today. However, it is important to understand where it came from in order to figure out where it will be heading in the future. While there is no clear cut evolutionary path to Clouds, many believe the concepts originate from two specific areas: Grid Computing and Web 2.0.

Grid computing [60, 61], in its practical form, represents the concept of connecting two or more spatially and administratively diverse clusters or supercomputers together in a federating manner. The term "the Grid" was coined in the mid 1990s to represent a large distributed systems infrastructure for advanced scientific and engineering computing problems. Grids aim to enable applications to harness the full potential of resources through coordinated and controlled resource sharing by scalable virtual organizations. While not all of these concepts carry over to the Cloud, the control, federation, and dynamic sharing of resources is conceptually the same as in

the Grid. This is outlined by Foster et al [3], as Grids and clouds appear conceptually similar when compared abstractly. From a scientific perspective, the goals of clouds and Grids are also similar. Both systems attempt to provide large amounts of computing power by leveraging a multitude of sites running diverse applications concurrently in symphony.

The other major component, Web 2.0, is also a relatively new concept in the history of Computer Science. The term Web 2.0 was originally coined in 1999 in a futuristic prediction by Dracy DiNucci [62]: "The Web we know now, which loads into a browser window in essentially static screenfulls, is only an embryo of the Web to come. The first glimmerings of Web 2.0 are beginning to appear, and we are just starting to see how that embryo might develop. The Web will be understood not as screenfulls of text and graphics but as a transport mechanism, the ether through which interactivity happens. It will [...] appear on your computer screen, [...] on your TV set [...] your car dashboard [...] your cell phone [...] hand-held game machines [...] maybe even your microwave oven." Her vision began to form, as illustrated in 2004 by the O'Riley Web 2.0 conference, and since then the term has been a pivotal buzzword among the internet. While many definitions have been provided, Web 2.0 really represents the transition from static HTML to harnessing the Internet and the Web as a platform in of itself.

Web 2.0 provides multiple levels of application services to users across the Internet. In essence, the web becomes an application suite for users. Data is outsourced to wherever it is wanted, and the users have total control over what they interact with and spread accordingly. This requires extensive, dynamic, and scalable hosting resources for these applications, the demand for which provides the user-base for much of the commercial Cloud computing industry today. Web 2.0 software requires abstracted resources to be allocated and relinquished on the fly, depending on the

Web's traffic and service usage at each site. Furthermore, Web 2.0 brought into existence Web Services standards [63] and the Service Oriented Architecture (SOA) [64], which outlined the interaction between users and cyber infrastructure. In summary, Web 2.0 defined the interaction standards and user base, and Grid computing defined the underlying infrastructure capabilities.

A Cloud computing implementation typically enables users to migrate their data and computation to a remote location with some varying impact on system performance [65]. This provides a number of benefits which could not otherwise be achieved:

- *Scalable* - Clouds are designed to deliver as much computing power as any user needs. While in practice the underlying infrastructure is not infinite, the cloud resources are projected to ease the developer's dependence on any specific hardware.

- *Quality of Service (QoS)* - Unlike standard data centers and advanced computing resources, a well-designed Cloud can project a much higher QoS than traditionally possible. This is due to the lack of dependence on specific hardware, so any physical machine failures can be mitigated without the prerequisite user awareness.

- *Specialized Environment* - Within a Cloud, the user can utilize customized tools and services to meet their needs. This can be to utilize the latest library, toolkit, or to support legacy code within new infrastructure.

- *Cost Effective* - Users leverage only the resources required for their given task, rather than putting forth a large capital investment in infrastructure. This reduces the risk for institutions potentially looking to build a scalable system, thus providing greater flexibility, since the user is only paying for needed in-

frastructure while maintaining the option to increase services as needed in the future.

- *Simplified Interface* - Whether using a specific application, a set of tools or Web services, Clouds provide access to a potentially vast amount of computing resources in an easy and user-centric way. We have investigated such an interface within Grid systems through the use of the Cyberaide project [66, 67].

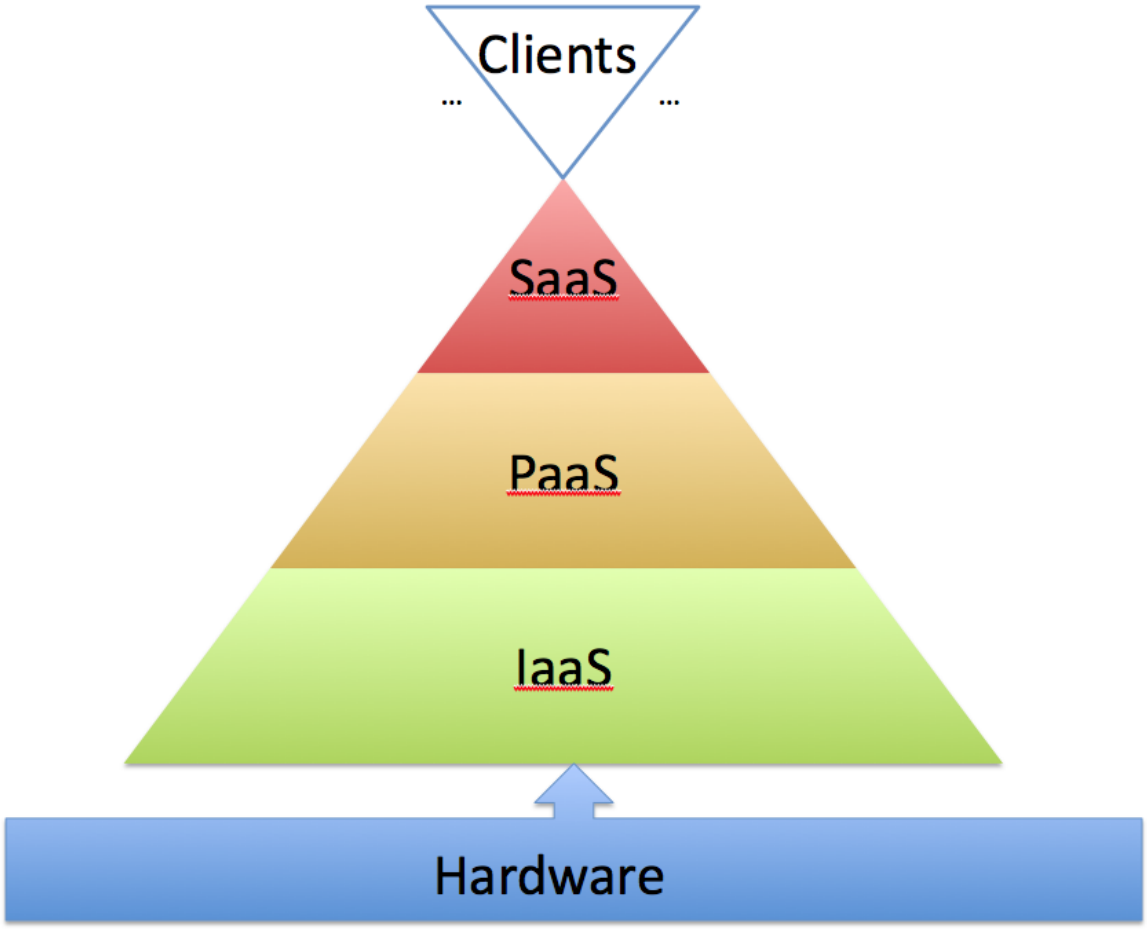


Figure 2.3 View of the Layers within a Cloud Infrastructure

Many of the features noted above define what Cloud computing can be from a user perspective. However, Cloud computing in its physical form has many different meanings and forms. Since Clouds are defined by the services they provide and not

by applications, an integrated as-a-service paradigm has been defined to illustrate the various levels within a typical Cloud, as in Figure 2.3.

- *Clients* - A client interacts with a Cloud through a predefined, thin layer of abstraction. This layer is responsible for communicating the user requests and displaying data returned in a way that is simple and intuitive for the user. Examples include a Web Browser or a thin client application.

- *Software-as-a-Service (SaaS)* - A framework for providing applications or software deployed on the Internet packaged as a unique service for users to consume. By doing so, the burden of running a local application directly on the client's machine is removed. Instead, all the application logic and data is managed centrally and displayed to the user through a browser or thin client. Examples include Google Docs, Facebook, or Pandora.

- *Platform-as-a-Service (PaaS)* - A framework for providing a unique computing platform or software stack on which applications and services can be developed. The goal of PaaS is to alleviate many of the burdens of developing complex, scalable software by proving a programming paradigm and tools that make service development and integration a tractable task for many. Examples include Microsoft Azure and Google App Engine.

- *Infrastructure-as-a-Service (IaaS)* - A framework for providing entire computing resources through a service. This typically represents virtualized Operating Systems, thereby masking the underlying complexity and details of the physical infrastructure. Users are allowed to rent or buy computing resources on demand for their own use without needing to operate or manage physical infrastructure. Examples include Amazon EC2, and OpenStack, and is the major cloud focal point for this dissertation.

- *Physical Hardware* - The underlying set of physical machines and IT equipment that host the various levels of service. These are typically managed at a large scale using virtualization technologies which provide the QoS users expect. This is the basis for all computing infrastructure.

When all of these layers are combined, a dynamic software stack is created to focus on large scale deployment of services to users.

2.2.1 Infrastructure-as-a-Service

Today, there are a number of Clouds that offer solutions for Infrastructure-as-a-Service (IaaS). There have been multiple comparison efforts between various IaaS service [4, 68–70], which provide insight to the similarities and differences between the long array of cloud infrastructure deployment solutions. However, at a high level, IaaS can be split into 3 tiers based on their availability.

- **Public** - Public IaaS is where the services and virtualizaton of hardware resources are provided over the internet. Usually this is in a centralized data center whereby many users concurrently access these resources from across the globe, often at a pre-negotiated price point and service level agreement (SLA) [71]. Public clouds, which sell hosting services en-masse at competitive costs, are best suited to utilize the economies of scale. The Amazon Elastic Compute Cloud (EC2) is a primary example of a public cloud.

- **Private** - Private IaaS is where the cloud infrastructure is limited to within a distinct group, set of users, business, or virtual organization. Usually such private cloud infrastructure is also on a private, dedicated network that can either be separate or connected to other services. Data within private clouds are often more secure than those on a public cloud, and as such can be the choice

for many users who have sensitive data, or who large-scale users who find better cost, performance, or QoS than what is provided within public clouds.

- **Hybrid** - Hybrid cloud IaaS combines the computational power of both private and public IaaS, enabling users to keep data or costs within a private IaaS, but then "burst" usage to a public cloud on peak computational demands. Virtual Private Networks (VPN) can be useful to try to handle such a hybrid cloud not only for management and network addressing but also for security.

Amazon EC2

The Amazon Elastic Compute Cloud (EC2) [72], is probably the most popular of cloud infrastructure offerings to date, and is used extensively in the IT industry. EC2 is the central component of Amazon Web Services platform. EC2 allows users to effectively rent virtual machines (called instances), hosted within Amazon's data centers, at a certain price point. Through their advanced UI or a RESTful API, users can start, stop, pause, migrate, and destroy instances exactly as needed to match the required computational tasks at hand.

Amazon EC2 predominantly relies on the Xen hypervisor to provide VMs on demand to users, with an equivalent compute unit equal to a 1.7Ghz Intel Xeon processor. However, recent advancements, instance types, and upgrades to EC2 have increased this compute unit's power. Instance reservations have 3 types: On-demand, Reserved, and Spot. With On-demand instances, users pay by the hour for however long the desired instance is running. Users can instead rent reserved instances, where they pay a one-time (discounted) cost based on a pre-determined allocation time. There is also Spot pricing, where VMs are provisioned only when a given spot price is met, which is determined simply based on supply and demand within the EC2 system itself.

EC2 supports a wide array of user environments and setups. From an OS perspective, this includes running Linux, Unix, and even Windows VM instances. EC2 also provides instances with persistent storage through Elastic Block Storage (EBS) and Simple Storage Service (S3) object storage mechanisms. These tools are necessary for data persistence, as EC2 instances, as with most IaaS solutions, do not implicitly persist data beyond the lifetime of the instance. EBS-rooted instances use an EBS volume as a root disk device and, as such, keep data beyond the lifetime of a given instance. EC2 also offers elastic IPs, whereby public IP addresses are assigned to instances at boot (or in-situ); however, these elastic IPs do not require the DNS updates to propagate or an administrator to adjust the network.

These advanced features, coupled with the first-to-market viability and continual updates have made EC2 the largest cloud infrastructure today. While vendor lock-in is a concern (EC2 is not available for download or replication), other alternatives exist such as Google's Compute Engine [73], the prevalence and support with EC will likely mean its status quo as the public cloud of choice will continue for the foreseeable future.

Nimbus

Nimbus [74, 75] is a set of open source tools that provide a private IaaS cloud computing solution. Nimbus is based on the concept of virtual workspaces previously introduced for Globus [75]. A virtual workspace is an abstraction of an execution environment that can be made dynamically available to authorized clients by using well-defined protocols. In this way, it can create customized environments by deploying virtual machines (VMs) among remote resources. To such an end, Nimbus provides a web interface called Nimbus Web. Its aim is to provide administrative and user functions in a friendly interface.

Within Nimbus, a storage cloud implementation called Cumulus [74] has been tightly integrated with the other central services, although it can also be used standalone. Cumulus is compatible with the Amazon Web Services S3 REST API [76], but extends its capabilities by including features such as quota management. The Nimbus cloud client uses the Jets3t library [77] to interact with Cumulus. However, since it is compatible with S3 REST API, other interfaces like boto [78] or s2cmd [79] can also be used to interact with Nimbus.

Nimbus supports two resource management strategies. The first one is the default "resource pool" mode. In this mode, the service has direct control of a pool of virtual machine managers (VMM) nodes and it assumes it can start VMs. The other supported mode is called "pilot". Here, the service makes requests to a cluster's Local Resource Management System (LRMS), to get a VMM available to gather where deploy VMs.

Nimbus also provides an implementation of EC2's interface that allows clients developed for the EC2 system to be used on Nimbus-based clouds.

Eucalyptus

Eucalyptus is a product from Eucalyptus Systems [80–82] that developed out of a research project at the University of California, Santa Barbara. Eucalyptus was initially aimed at bringing the cloud computing paradigm of computing to academic super computers and clusters. Eucalyptus provides an Amazon Web Services (AWS) complaint EC2 based web service interface for interacting with the Cloud service.

The architecture depicted in Figure 2.4 is based on a two level hierarchy of the Cloud controller and the Cluster controller [83]. The Cluster Controller usually manages the nodes within a single cluster and multiple Cluster Controllers can be used to connect to a single Cloud Controller. The Cloud Controller is responsible for the

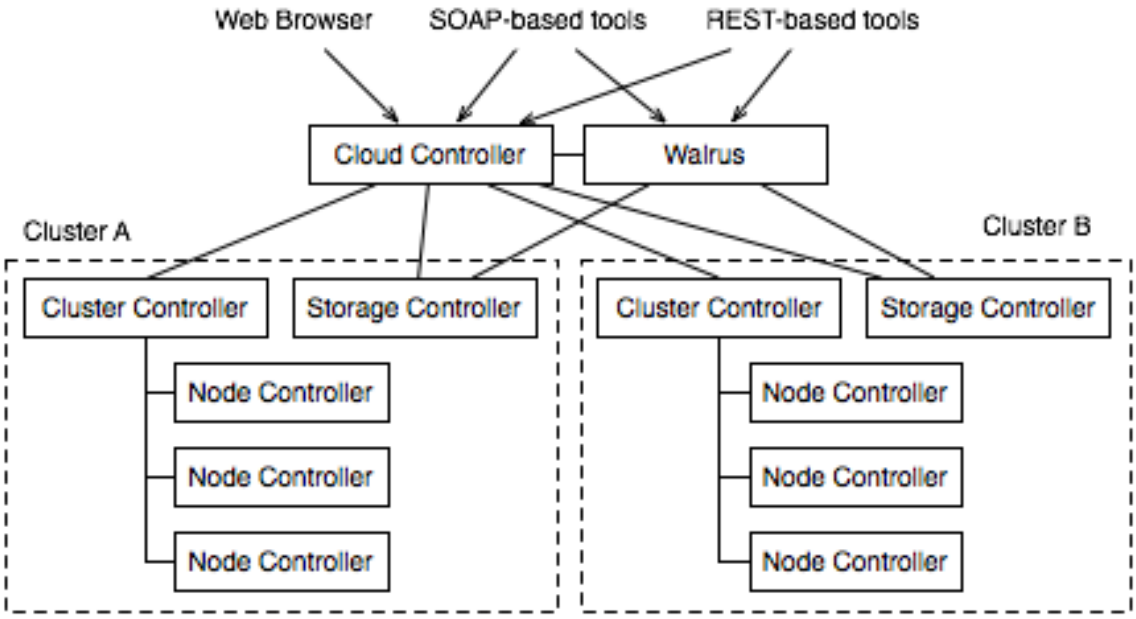


Figure 2.4 Eucalyptus Architecture

resource management, scheduling and accounting aspects of the Cloud.

Being one of the first private cloud computing solutions, Eucalyptus has a focus on the user interface. Much of Eucalyptus's design is based on the functionality of Amazon's EC2 cloud solution, and the user interface is a prime example of that model. While EC2 is a proprietary public cloud, it uses an open interface through the use of well designed Web Services which are open to all. Eucalyptus, looking to provide complete compatibility with EC2 to market the private cloud market, uses the same interface for all communication to the Cloud Controller. Because Eucalyptus's interface is AWS complaint, it provides the same form of authentication that AWS supports, namely the shared key and PKI models.

While Eucalyptus can be controlled using the EC2 AMI tools, it also provides its own specific tool set: euca2ools. Euca2ools provides support for creating and managing keypairs, querying the cloud system, managing VMs, starting and terminating instances, network configuration, and block storage usage. The Eucalyptus system

also provides a secure web front-end to allow new users to create and manage account information, view available VMs, and download their security credentials.

As seen with the user interface, Eucalyptus takes many design queues from Amazons EC2, and the image management system is no different. Eucalyptus stores images in Walrus, the block storage system that is analogous to the Amazon S3 service. As such, any user can bundle his/her own root filesystem, upload and then register this image and link that image with a particular kernel and ramdisk image. This image is uploaded into a user-defined bucket within Walrus, and can be retrieved anytime from any availability zone. This allows users to create specialty virtual appliances and deploy them within Eucalyptus with ease.

In 2014, Eucalyptus was acquired by Hewlett-Packard, which now maintains the HPE Helion Eucalyptus Cloud to have full compatibility with Amazon EC2. The most recent release of Helion eucalyptus is version 4.2.2 in early 2016.

OpenStack

OpenStack [84, 85], another private cloud infrastructure service, was introduced by Rackspace and NASA in July 2010. The project aims to build an open source community spanning technologists, developers, researchers, and industry that allows for the sharing of resources and technologies in order to create a massively scalable and secure cloud infrastructure. In tradition with other open source projects, the entire software is open source.

Historically, OpenStack focuses on the development of two aspects of cloud computing to address compute and storage aspects with their OpenStack Compute and OpenStack Storage solutions. According to the documentation "OpenStack Compute is the internal fabric of the cloud creating and managing large groups of virtual private servers" and "OpenStack Object Storage is software for creating redundant,

scalable object storage using clusters of commodity servers to store Terabytes or even petabytes of data." However, OpenStack as a platform has evolved much more than its original efforts, and has created a wide array of new sub-projects.

As part of the computing support effort, OpenStack utilizes a cloud fabric controller known under the name Nova. The architecture for Nova is built on the concepts of shared-nothing and messaging-based information exchange. Hence most communications in Nova are facilitated by message queues. To prevent blocking components while waiting for a response from others, deferred objects are introduced. Nova supports multiple scheduling paradigms and includes plugins for a wide array of hypervisors, including Xen, KVM, and VMWare. The flexibility found within Nova is useful for supporting a wide array of cloud IaaS computational efforts. Recently, OpenStack has even looked to implement containers and bare-metal provisioning to keep on pace with the latest technologies.

The OpenStack Swift storage solution is build around a number of interacting components and concepts, including a Proxy Server, a Ring, Object Server, a Container Server, an Account Server, Replication, Updaters, and Auditors. This distributed architecture attempts to have no centralized components in order to enable scalability and resiliency for data. Swift represents the long-term, object-based storage similar to Amazon S3, and attempts to maintain rough API compatibility with S3. As Swift looks to use simple data replication as a main form of resiliency and fast read/write is rarely a priority, Swift is often built using commodity disk drives instead of more costly flash solutions.

With OpenStack Nova's increased prevalence, the number of auxiliary OpenStack projects has also increased to support Nova. While there are many other recent OpenStack projects, these listed OpenStack efforts, along with Nova and Swift, represent the common core of a current OpenStack deployment.

- **Cinder** for persistent block-level storage mechanisms to support VM instances and elastic block storage.
- **Neutron** provides advanced networking and SDN solutions, IP addressing, and VLAN configuration.
- **Glance** delivers comprehensive image management, including image discovery, registration, and delivery mechanisms.
- **Keystone** identity and authentication service for all OpenStack services.
- **Horizon**, a dashboard web-based UI framework, complimentary to the RESTful client API.

Currently, OpenStack exists as one of the largest ongoing private IaaS efforts, with over 500 companies contributing to the effort, and thousands of deployments. While releases have pushed forth approximately every 6 months, the latest current release at the time of writing is *Mitaka*, which now includes full support for GPUs and SR-IOV interconnects, as detailed later in this dissertation. It is expected that OpenStack's prevalence in the cloud computing community will only increase in the next few years.

OpenNebula

OpenNebula [86, 87] is an open-source toolkit which allows administrators to transform existing infrastructure into an Infrastructure as a Service (IaaS) cloud with cloud-like interfaces. Figure 2.5 shows the OpenNebula architecture and their main components.

The architecture of OpenNebula has been designed to be flexible and modular to allow its integration with different storage and network infrastructure configurations,

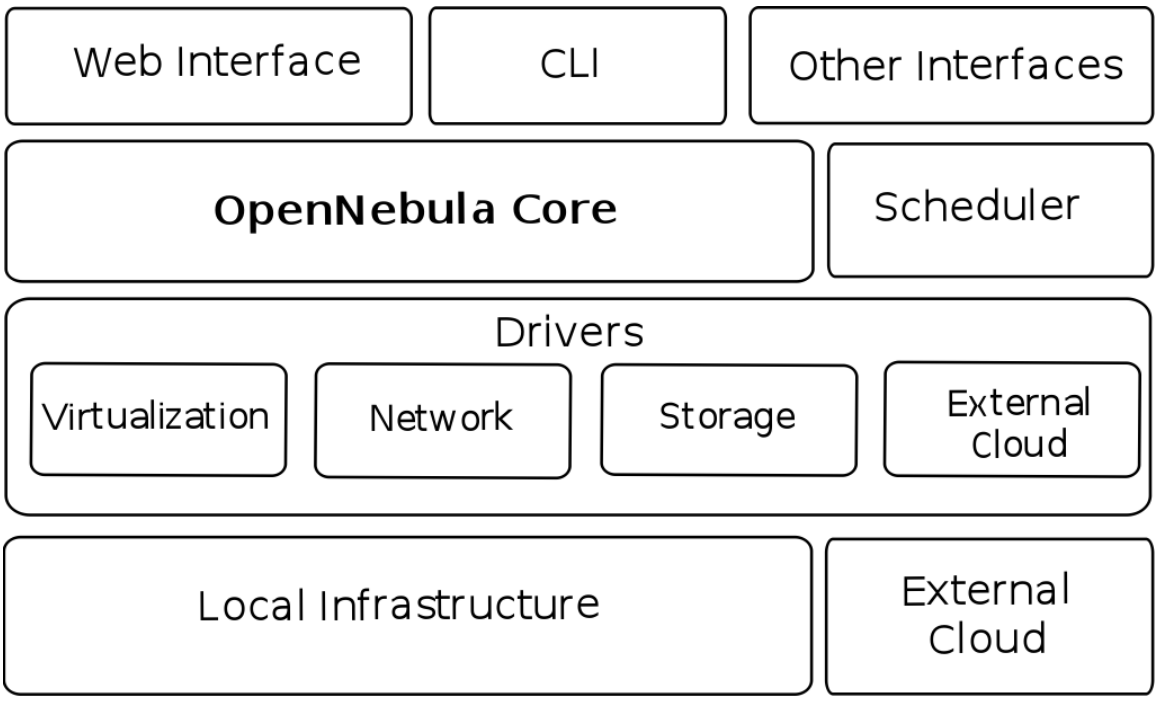


Figure 2.5 OpenNebula Architecture

as well as hypervisor technologies. Here, the core is a centralized component that manages the virtual machine's (VM) full life cycle, including setting up networks dynamically for groups of VMs and managing their storage requirements, such as VM disk image deployment or on-the-fly software environment creation. Another important component is the capacity manager, which governs the functionality provided by the core for scheduling. The default capacity scheduler is a requirement/rank matchmaker. However, it is also possible to develop more complex scheduling policies through a lease model and advance reservations like Haizea [88]. The last main components are the access drivers. They provide an abstraction of the underlying infrastructure to expose the basic functionality of the monitoring, storage and virtualization services available in the cluster. Therefore, OpenNebula is not tied to any specific environment and can provide a uniform management layer regardless of the virtualization platform.

Additionally, OpenNebula offers management interfaces to integrate the core's functionality within other data center management tools, such as accounting or monitoring frameworks. To this end, OpenNebula implements the libvirt API [89], an open interface for VM management, as well as a command line interface (CLI). A subset of this functionality is exposed to external users through a cloud interface.

Due to its architecture, OpenNebula is able to adapt to organizations with changing resource needs, including the addition or failure of physical resources [70]. Some essential features to support changing environments are the live migration and the snapshotting of VMs [86]. Furthermore, when the local resources are insufficient, OpenNebula can support a hybrid cloud model by using cloud drivers to interface with external clouds. This lets organizations supplement the local infrastructure with computing capacity from a public cloud to meet peak demands, or implement high availability strategies. OpenNebula includes an EC2 driver, which can submit requests to Amazon EC2 and Eucalyptus [80], as well as an ElasticHosts driver [90].

Regarding the storage, an OpenNebula Image Repository allows users to easily specify disk images from a catalog without worrying about low-level disk configuration attributes or block device mapping. Also, image access control is applied to the images registered in the repository, hence simplifying multi-user environments and image sharing. Nevertheless, users can also set up their own images.

Others

Other cloud specific projects exist, such as In-VIGO [91], Cluster-on-Demand [92], and VMWare's own proprietary vCloud Air [93]. Each effort provides its own interpretation of private cloud services within a data center, often with the ability to interplay with pubic cloud offerings such as Amazon's EC2. Docker [50] also looks to provide IaaS capabilities with specialized and easily configurable containers, based

on LXC and libcontainer solutions. While it is still to be determined how Docker and containers will change the private IaaS landscape, they do provide similar functionality for Linux users without some of the complexities of traditional virtualized IaaS.

2.2.2 Virtual Clusters

While virtualization and cloud IaaS provide many key advancements, this technology alone is not sufficient. Rather, a collective scheduling and management for virtual machines is required to piece together a working virtual cluster.

Let us consider a typical usage for a Cloud data center that is used in part to provide computational power for the Large Hadron Collider at CERN [94], a global collaboration from more than 2000 scientists of 182 institutes in 38 nations. Such a system would have a small number of experiments to run. Each experiment would require a very large number of jobs to complete the computation needed for the analysis. Examples of such experiments are the ATLAS [95] and CMS [96] projects, that (combined) require Petaflops of computing power on a daily basis. Each job of an experiment is unique, but the application runs are often the same. Therefore, virtual machines are deployed to execute incoming jobs. There is a file server which provides virtual machine templates. All typical jobs are preconfigured in virtual machine templates. When a job arrives at the head node of the cluster, a correspondent virtual machine is dynamically started on a certain compute node within the cluster to execute the job. While the LHC project and CERN's cloud effort is a formidable one, it only covers pleasingly parallel HTC workloads, and often times HPC and big data workloads can be equally complex yet drastically different.

Cluster computing has become one of the core tools in distributed systems for use in parallel computation. Cluster computing revolves around the desire to get

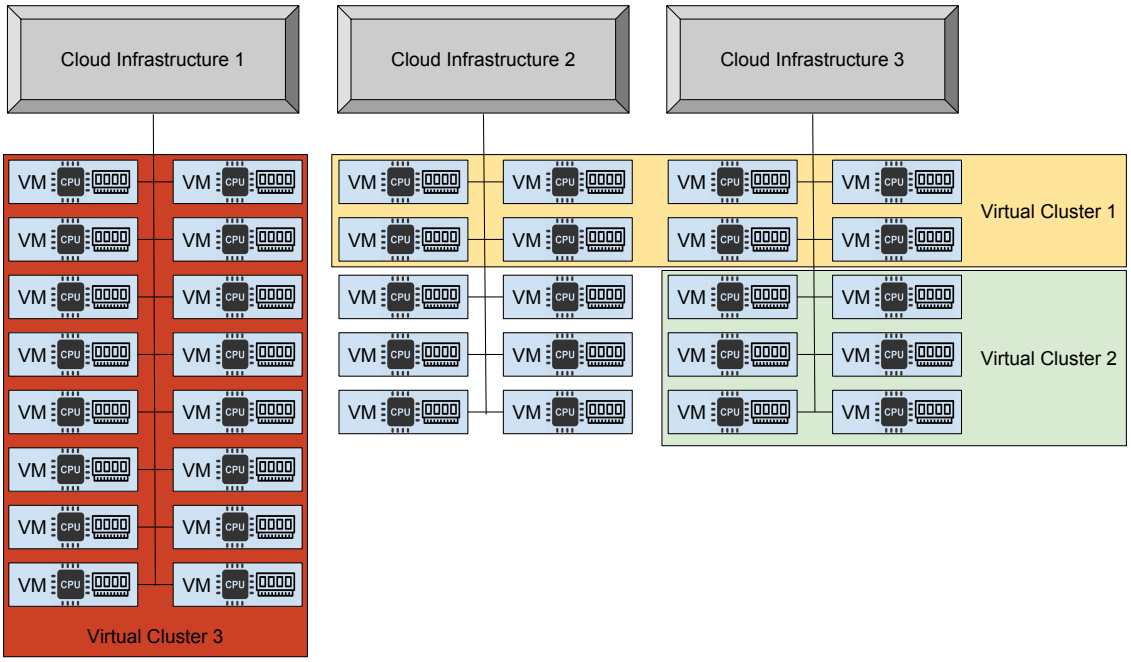


Figure 2.6 Virtual Clusters on Cloud Infrastructure

more computing power and better reliability by utilizing many computers together across a network to achieve larger computational tasks. Clusters have manifested themselves in many different ways, ranging from Beowulf clusters [6], which run using commodity PCs to some of the TOP500 [8] supercomputing systems today. Virtual clusters represent the growing need of users to organize computational resources in an environment specific to their tasks at hand effectively, instead of sharing a common architecture across many users. With the advent of modern virtualization, virtual clusters are deployed across a set of VMs in order to gain relative isolation and flexibility between disjoint virtual clusters. Virtual clusters, or a set of multiple cluster computing deployments on a single, larger physical cluster infrastructure, often have the following properties and attributes [53]:

- Resources allocation based on a VM unit
- Clusters built of many VMs together, or by provisioning physical nodes

- Leverage local infrastructure management tools to provide a middleware solution for virtual clusters
  - Implementations could be a cloud IaaS such as OpenStack
  - Some instances use a queueing system such as PBS
- User experience based on virtual cluster management, not single VM management
- Consolidates functionality on a smaller resource platform using multiple VMs
- Can provide fault tolerance through VM migration and management
- Can utilize dynamic scaling through the addition or deletion of VMs from the virtual cluster
- Connection to back-end storage solution to provide virtual persistent storage

Given the properties, virtual clusters can take on many forms; however, a very simplified set of virtual clusters across cloud infrastructure is provided as a representation in Figure 2.6. This could lead to the simple provisioning of multiple disjoint OSs on a single physical resource. Virtual clusters generally have the ability to provide and manage their own user environment and tuned internal middleware. Virtual clusters may enable the separation of multiple tasks into separate VMs, which still in fact run on the same or similar underlying physical resources, effectively providing task isolation. Virtual clusters can be deployed to be persistent, stored, shared, or re-provisioned on demand. The size of a virtual cluster could potentially expand and contrast relative to the necessary resource requirements, taking advantage of elasticity found with virtualization. Furthermore, VM migration may enable fault tolerance in the event of physical machine errors if properly managed.

With virtual clusters, the capability to deploy custom environments quickly becomes critical. As such, efforts have been put forth to configure and create VM images on-demand. This includes custom efforts with configuration engines such as CfEngine, Chef, Ansible, and others. Within FutureGrid, an image management system was defined to provide preconfigured VM images for cloud infrastructure using the BCFG2 engine [97].

Initially, virtual clusters were proposed for the use of Grid communities [32]. Specifically, Foster et al look to provide commodity clusters to various Virtual Organizations [98], whereby grid services can instantiate and deploy VMs. This design and implementation was further refined through the use of metadata and contextualization using appliances [99]. Some of these ideas have even come to take shape in larger scale supercomputing deployments, such as with SDSC Comet's virtual cluster availability [100].

Virtual clusters require orchestration services to be able to organize, deploy, manage, and re-play the desired user environment, and there have been a number of efforts to bring this orchestration to utility. One efforts within FutureGrid is with the experiment management design [44], which attempts to define how resources are connected to and monitored, as well as how VMs are stored in a repository and provisioned across multiple heterogeneous resources. This effort moved forward with Cloudmesh [46], which provides a simple client interface to access multiple cloud resources with a command-line shell interface.

Another virtual cluster orchestration, named OpenStack Heat [45], has developed within the OpenStack community. Heat provides a method by which an individual or group can deploy "stacks", which could essentially be virtual clusters, using OpenStack infrastructure. Specifically, Heat provides a human readable and machine-accessible template for specifying environments and requirements, as well as a REST-

ful API. In submitting a Heat orchestration template to the API, heat will interpret and build the designed custom environment within a given OpenStack cloud deployment. Kubernetes [101], a related effort, is a infrastructure orchestration framework for managing containerized applications within Docker.

2.2.3 The FutureGrid Project

FutureGrid was a NSF-funded national-scale Grid and Cloud test-bed facility that included a number of computational resources across many distributed locations. This FutureGrid test-bed allowed users to evaluate differing systems for applicability with their given research task or application. These areas include computer science research topics ranging from authentication, authorization, scheduling, virtualization, middleware design, interface design and cybersecurity, to the optimization of grid-enabled and cloud-enabled computational schemes for Astronomy, Chemistry, Biology, Engineering, High Energy Physics, or Atmospheric Science. This project started at an opportune time, when cloud infrastructure was still in its experimental stages and its applicability to mid-tier scientific efforts were unknown.

The FutureGrid features a unique WAN network structure that lent itself to a multitude of experiments specifically designed for evaluating middleware technologies and experiment management services. This network can be dedicated to conduct experiments in isolation using a network impairment device for introducing a variety of predetermined network conditions. Figure 2.7 depicts the geographically distributed resources that are outlined in Table 2.1 in more detail. All network links within FutureGrid are dedicated 10GbE links with the exception of a shared 10GbE link to TACC over the TeraGrid [102, 103] network, enabling high-speed data management and transfer between each partner site within FutureGrid.

Although the total number of systems within FutureGrid is comparatively conser-

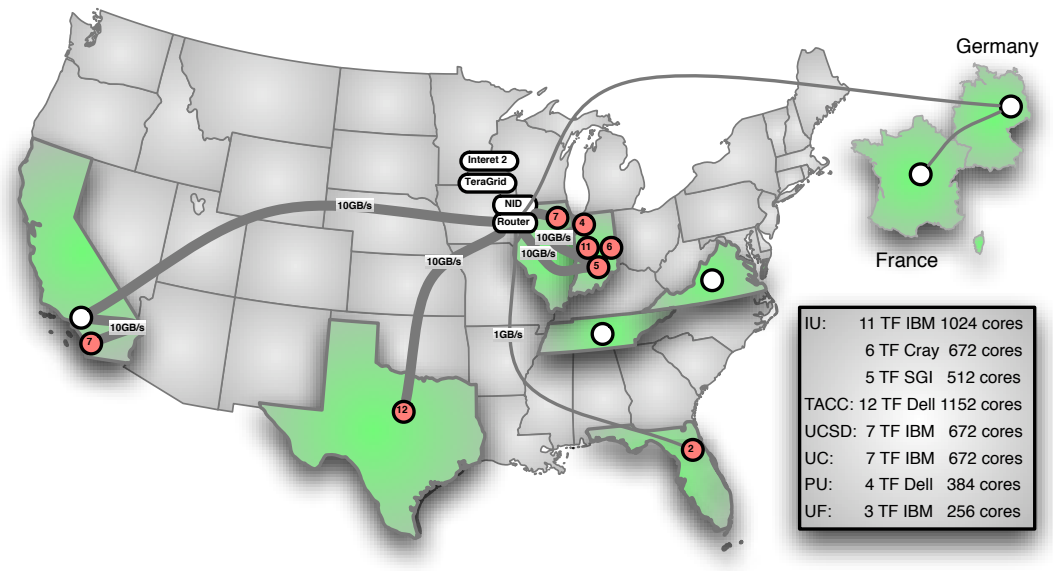


Figure 2.7 FutureGrid Participants, Network, and Resources

vative, they provide some heterogeneity to the architecture and are connected by the high-bandwidth network links. One important feature to note is that most systems can be dynamically provisioned; e.g., these systems can be reconfigured when needed by special software that is part of FutureGrid with proper access control by users and administrators. Therefore, it is believed that this hardware infrastructure can fully accommodate the needs of an experiment management system.

As of Fall 2014, the FutureGrid project has ended. The computing resources and facilities at Indiana University have continued on as FutureSystems, continuing to provide a cloud, big data, and HPC testbed to approved researchers. Recently, with new projects as part of the digital Science Center, the FutureSystems effort has added two new machines, *Romeo* and *Juliet*, presenting clusters with Intel Haswell CPU architectures to be used for big data research.

Table 2.1 FutureGrid hardware

System type	Name	CPUs	Cores	TFLOPS	RAM	Disk	Site
IBM iDataPlex	India	256	1024	11	3072	†335	IU
Dell PowerEdge	Alamo	192	1152	12	1152	15	TACC
IBM iDataPlex	Hotel	168	672	7	2016	120	UC
IBM iDataPlex	Sierra	168	672	7	2688	72	UCSD
Cray XT5m	Xray	168	672	6	1344	†335	IU
ScaleMP vSMP	Echo	32	192	3	5872	192	IU
Dell PoweEdge	Bravo	32	128	2	3072	192	IU
SuperMicro	Delta	32	192	‡20	3072	128	IU
IBM iDataPlex	Foxtrot	64	256	2	768	5	UF
Total		1112	4960	70	23056	1394	

†Indicates shared file system. ‡Best current estimate

2.3 High Performance Computing

2.3.1 Brief History of Supercomputing

Supercomputing can date back to some of the forefront of computing itself, especially if we consider the ENIAC [104], the first Turing Complete general purpose digital computer, also as the first supercomputer. ENIAC was first deployed to calculate artillery firing tables, but was later used during the Second World War for helping the Manhattan project's thermonuclear calculations and later dedicated to the University of Pennsylvania after the war.

The first properly termed supercomputer was the Control Data Corporation's 6600 mainframe [105], first delivered to CERN in 1965. The CDC 6600 was notably faster than the IBM counterparts, and the first deployments were able to perform on the

order of 1 MFLOP. Interestingly, the CPU design that came from Seymour Cray's CDC 6600 took advantage of a simplified yet fast CPU design with silicon-based transistors, which founded the basis of the RISC processor architecture.

Cray's efforts eventually lead him to start his own company, and in 1975 released the Cray 1 system [106]. The Cray 1 took the powerful aspects of vector processing and memory pipelining from the STAR architecture (developed later by CDC) and introduced scalar performance through splitting vectors and instruction chaining. This resulted in an overall performance of around 250 MFLOPS at peak, but realistically closer to 100 MFLOPS for general applications. The Cray 1 system also helped push forward the integrated circuit design, which was finally performant enough to be used. Interestingly enough, the Cray 1 also required an entirely new Freon-based coolant system.

The Cray 1 system gave way to the Cray X-MP and Y-MP in the mid 1980s. These machines were shared-memory vector processors, with two processors in the X-MP and up to 8 processors for the later Y-MP systems. These first shared-memory systems were aided by increased memory speeds. The X-MP machine was capable of 200 MFLOPS sustained and 400 MFLOPS peak performance, whereas the Y-MP variants were capable of over 2 GFLOPS.

Concurrently, during the 1980s, the advent of distributed memory architectures for supercomputing were also starting to emerge. Specifically work on Caltech's Cosmic Cube, also known as a Hypercube, by Seitz and Fox [19,107], started the movement of concurrent or parallel computing. The Cosmic Cube leveraged new VLSI techniques and assembled Intel 8086/87 processors together with a novel hypercube interconnect which required no switching, creating one of the first truly parallel computers. SIMD programming and computation was done through a novel message passing architecture, instead of shared variables. This design was first commercialized with Intel's

iPSC, and later contributed directly to designs in the Intel Paragon, ASCI Red, and Cray T3D/E systems.

While concurrent and parallel processor supercomputers continued into the 90s with aforementioned hypercube designs and the IBM Thinking Machines [108], a new commodity-based strategy emerged with Becker and Sterling's Beowulf clusters [6]. Effectively, a Beowulf cluster is simply a cluster of commodity x86 machines linked together with a simplified LAN network. Beowulf clusters often (but not always) run Linux OS and leverage Ethernet solutions, and are programmed using a message passing construct such as MPI [18]. This allows for the building of massively parallel systems with relatively low cost and investment. Many specialized clusters today still utilize commodity x86 hardware and Linux OSs similar to the original Beowulf systems.

Concurrency and parallel computation on supercomputing resources has only flourished since. This has been even more pronounced as CPU clock frequencies stabilized and multi-core architectures took hold, driving the need for concurrency not only at an increased rate for supercomputing, but also even for commodity systems. While commodity single-CPU, multi-core systems often look towards exploiting shared memory parallelism, distributed memory architectures have become a way of life for high performance computing.

2.3.2 Distributed Memory Computation

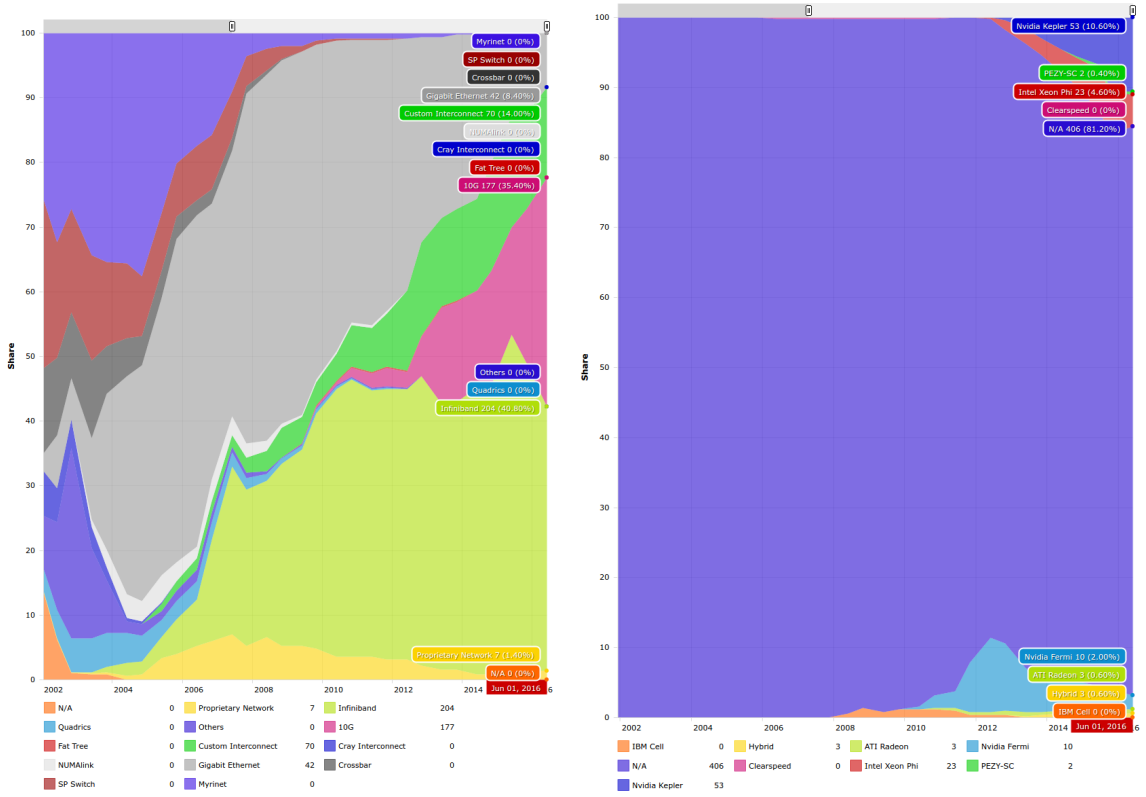
Abstractly, distributed memory architectures consist of multiple instances of a processor, memory, and an interconnect, which allows each instance to perform independent computations and communicate over the interconnect. These interconnects could be built using point-to-point, or through more complex and advanced switching hardware, building a larger topology. While a simple example could involve several

commodity PCs connected through an Ethernet switch, this model scales to the latest supercomputing resources of today with millions of cores [109].

Programming such distributed memory systems is a nontrivial task that the greater HPC community has been wrangling for years. In the early days, this took the form of either the Message Passing Interface (MPI) [18] or PVM [110]; however, MPI has been far more successful and dominates the market for distributed memory parallel computation. MPI is a standardized message passing system, which defines the semantics and syntax for writing parallel programs in C, C++, or Fortran. MPI has even been implemented in other languages such as Java [111]. As MPI is a standardization, there are may implementations that exist, including OpenMPI [112], MPICH [113], and MVAPICH [114], to name a few. Many MPI-enabled applications have been shown to be, with proper and careful design, the most efficient way to run a parallel application across a large subset of tightly coupled distributed resources, and can often represent the status-quo for HPC applications today.

More recently, the MPI programming model has been modified or joined with other models. This change or deviation largely revolves around the hardware that has changed within HPC resources themselves to meet the need for more computational power. This includes hybrid MPI + OpenMP models which become useful as multi-core and many-core technologies becomes increasingly available [115], or the MPI+CUDA model which interleaves message passing with GPU utilization [116]. As some of the latest supercomputing resources have been deployed specifically with Nvidia GPUs for GPGPU programming, such as the ORNL's Titan system [13], the need for distributed memory computation with GPUs has increased.

To get an idea of some of the hardware advances over the past few years, it is useful to examine the Top 500 [8], a comprehensive list of the top 500 supercomputers known and their key characteristics. Looking at the past decade in HPC, we



(a) Interconnect Family

(b) Accelerator/Coprocessor Family

Figure 2.8 Top 500 Development over time from 2002 to 2016 [8]

can see some trends emerge within hardware. Specifically, looking at Interconnect and Coprocessor architectures in Figure 2.8, we see a notable jump in the number of deployed system that are using both InfiniBand and GPUs in the past decade. In particular, InfiniBand usage has increased to roughly 40% of the total number of deployed systems and remained somewhat stable as an interconnect family. Concurrently, the use of accelerators has increased form only 1 in 500 systems a decade ago, to almost a 20% of the top 500 systems that are using coprocessors. Within that factor, the majority of such accelerator-equipped systems have been using GPUs, with a concurrent increase in the Intel Xeon Phi coprocessor as well. While these do not represent all of supercomputing nor exclusively the most high end of systems, they do represent advanced hardware that has been increasingly common in the last decade, yet relatively underutilized in comparison within cloud infrastructure. As such, much of the effort in this dissertation focuses on, but is not limited to, these two technology families.

2.3.3 Exascale

As the forefront of supercomputing moves beyond the latest petaflop machines of the past few years [13], the HPC community is setting their sights on the next significant milestone: Exascale. Exascale computing refers broadly to performing roughly one exaFLOPS, or 10^{18} floating point operations per second. However, exascale itself is far more than just a theoretical FLOPS goal; instead, it is a set of new computing advancements and challenges that requires reaching computational power at such magnitude. While FLOPs are often used as the ubiquitous yard stick for supercomputing with the LINPACK benchmark [117], other efforts have taken hold to classify systems under a different set of parameters [118, 119], with the loose understanding that these may incorporate a richer application set destined for exascale systems. This

could include, for instance, integer calculations at a similar scale to satisfy defence and intelligence perspectives, or graph processing with billions of vertices.

With exascale, there are a number of barriers that exist with current technologies that must be overcome to reach exascale. The exascale Computing Study [39] specifically lists 4 major focal areas:

1. Energy and Power Challenge

    - Describes the physical difficulties in providing the amount of power needed to drive a sufficiently large exascale system. The US DOE estimates the maximum power envelope for a deployed first exascale system to be within 20-40MW. Extrapolating current technology power utilization shows an order of magnitude more energy utilization than the specificity power envelop. As such, new architectures and conversation techniques will need to be investigated.

2. Memory and Storage Challenge

    - This challenge illustrates the problem that has grown in relation to the memory wall, defined by the exponential difference between processor and memory performance, as well as the storage capacity limits to support calculations at the level of performance necessary. This challenge incorporates not only main memory limitations, but also tertiary storage issues as well.

3. Concurrency and Storage Challenge

    - This challenge is born from the recent limit in CPU clock rates as a way to gain performance. Instead, performance must be gained through paral-

lelism. The depth of this challenge is especially profound when we consider parallelism on the order of millions, if not billions, of threads.

4. Resiliency Challenge

    - The resiliency challenge defines the necessity of computation to recover and continue in the event of a fault or fluctuation. As parallelism and the number of individualized components substantially increases in a path towards exascale, the mean time to failure of any given component also increases.

Current exascale efforts are as wide as they are varying, not only with concepts, architectures, and runtime systems, but also with deployment plans and expectations between future deployments. Of particular interest in current exascale research is in Operating System and runtime (OS/R) developments to support new extreme-scale applications in an efficient manner. Two examples of novel OS approaches are the Hobbes project [120] and the ARGO Exascale Operating System [121]. These OS efforts, along with novel programming models for exascale such as ParalleX [122] look to fundamentally change the relationship between HPC hardware architectures and the libraries and applications to be leveraged on such future exascale deployments.

It is possible that virtualization itself may have an impact in OS and runtime services in exascale [120]. While some of the work herein may tangentially be of utility to such efforts, the immediate goal of this dissertation is not to investigate the applicability of virtualization for exascale systems, but rather to enable the diversification of HPC towards cloud infrastructure.

Chapter 3

Analysis of Virtualization Technologies for High Performance Computing Environments

3.1 Abstract

As Cloud computing emerges as a dominant paradigm in distributed systems, it is important to fully understand the underlying technologies that make Clouds possible. One technology, and perhaps the most important, is virtualization. Recently virtualization, through the use of hypervisors, has become widely used and well understood by many. However, there are a large spread of different hypervisors, each with their own advantages and disadvantages. This chapter provides an in-depth analysis of some of today's commonly accepted virtualization technologies from feature comparison to performance analysis, focusing on the applicability to High Performance Computing environments using FutureGrid resources. The results indicate virtualization sometimes introduces slight performance impacts depending on the hypervisor type, however the benefits of such technologies are profound and not all virtualization technologies are equal.

3.2 Introduction

Cloud computing [59] is one of the most explosively expanding technologies in the computing industry today. A Cloud computing implementation typically enables

users to migrate their data and computation to a remote location with some varying impact on system performance [65]. This provides a number of benefits which could not otherwise be achieved.

Such benefits include:

- *Scalability* - Clouds are designed to deliver as much computing power as any user needs. While in practice the underlying infrastructure is not infinite, the cloud resources are projected to ease the developer's dependence on any specific hardware.

- *Quality of Service (QoS)* - Unlike standard data centers and advanced computing resources, a well-designed Cloud can project a much higher QoS than traditionally possible. This is due to the lack of dependence on specific hardware, so any physical machine failures can be mitigated without the prerequisite user awareness.

- *Customization* - Within a Cloud, the user can utilize customized tools and services to meet their needs. This can be to utilize the latest library, toolkit, or to support legacy code within new infrastructure.

- *Cost Effectiveness* - Users finds only the hardware required for each project. This reduces the risk for institutions potentially want build a scalable system, thus providing greater flexibility, since the user is only paying for needed infrastructure while maintaining the option to increase services as needed in the future.

- *Simplified Access Interfaces* - Whether using a specific application, a set of tools or Web services, Clouds provide access to a potentially vast amount of computing resources in an easy and user-centric way.

While Cloud computing has been driven from the start predominantly by the industry through Amazon [72], Google [123] and Microsoft [124], a shift is also occurring within the academic setting as well. Due to the many benefits, Cloud computing is becoming immersed in the area of High Performance Computing (HPC), specifically with the deployment of scientific clouds [125] and virtualized clusters [32].

There are a number of underlying technologies, services, and infrastructure-level configurations that make Cloud computing possible. One of the most important technologies is virtualization. Virtualization, in its simplest form, is a mechanism to abstract the hardware and system resources from a given Operating System. This is typically performed within a Cloud environment across a large set of servers using a Hypervisor or Virtual Machine Monitor (VMM), which lies in between the hardware and the OS. From the hypervisor, one or more virtualized OSs can be started concurrently, leading to one of the key advantages of Cloud computing. This, along with the advent of multi-core processors, allows for a consolidation of resources within any data center. From the hypervisor level, Cloud computing middleware is deployed atop the virtualization technologies to exploit this capability to its maximum potential while still maintaining a given QoS and utility to users.

The rest of this chapter is as follows: First, we look at what virtualization is, and what current technologies currently exist within the mainstream market. Next we discuss previous work related to virtualization and take an in-depth look at the features provided by each hypervisor. We follow this by outlining an experimental setup to evaluate a set of today's hypervisors on a novel Cloud test-bed architecture. Then, we look at performance benchmarks which help explain the utility of each hypervisor and the feasibility within an HPC environment. We conclude with our final thoughts and recommendations for using virtualization in Clouds for HPC.

3.3 Related Research

While the use of virtualization technologies has increased dramatically in the past few years, virtualization is not specific to the recent advent of Cloud computing. IBM originally pioneered the concept of virtualization in the 1960's with the M44/44X systems [126]. It has only recently been reintroduced for general use on x86 platforms. Today there are a number of public Clouds that offer IaaS through the use of virtualization technologies. The Amazon Elastic Compute Cloud (EC2) [127] is probably the most popular Cloud and is used extensively in the IT industry to this day. Nimbus [128] and Eucalyptus [80] are popular private IaaS platforms in both the scientific and industrial communities. Nimbus, originating from the concept of deploying virtual workspaces on top of existing Grid infrastructure using Globus, has pioneered scientific Clouds since its inception. Eucalyptus has historically focused on providing an exact EC2 environment as a private cloud to enable users to build an EC2-like cloud using their own internal resources. Other scientific Cloud specific projects exist such as OpenNebula [129], In-VIGO [130], and Cluster-on-Demand [92], all of which leverage one or more hypervisors to provide computing infrastructure on demand. In recent history, OpenStack [131] has also come to light from a joint collaboration between NASA and Rackspace which also provide compute and storage resources in the form of a Cloud.

While there are currently a number of virtualization technologies available today, the virtualization technique of choice for most open platforms over the past 5 years has typically been the Xen hypervisor [54]. However more recently VMWare ESX [132] [1], Oracle VirtualBox [133] and the Kernel-based Virtual Machine (KVM) [58] are becoming more commonplace. As these look to be the most popular and feature-

[1]Due to the restrictions in VMWare's licensing agreement, benchmark results are unavailable.

rich of al virtualization technologies, we look to evaluate all four to the fullest extent possible. There are however, numerious other virtualizaton technologies also available, including Microsoft's Hyper-V [134], Parallels Virtuozzo [135], QEMU [136], OpenVZ [137], Oracle VM [138], and many others. However, these virtualization technologies have yet to seen widespread deployment within the HPC community, at least in their current form, so they have been placed outside the scope of this work.

In recent history there have actually been a number of comparisons related to virtualization technologies and Clouds. The first performance analysis of various hypervisors started with, unsurprisingly, the hypervisor vendors themselves. VMWare has happy to put out its on take on performance in [139], as well as the original Xen article [54] which compares Xen, XenoLinux, and VMWare across a number of SPEC and normalized benchmarks, resulting in a conflict between both works. From here, a number of more unbiased reports originated, concentrating on server consolidation and web application performance [132, 140, 141] with fruitful yet sometimes incompatible results. A feature base survey on virtualization technologies [142] also illustrates the wide variety of hypervisors that currently exist. Furthermore, there has been some investigation into the performance within HPC, specifically with InfiniBand performance of Xen [143] and rather recently with a detailed look at the feasibility of the Amazon Elastic Compute cloud for HPC applications [49], however both works concentrate only on a single deployment rather than a true comparison of technologies.

As these underlying hypervisor and virtualization implementations have evolved rapidly in recent years along with virtualization support directly on standard x86 hardware, it is necessary to carefully and accurately evaluate the performance implications of each system. Hence, we conducted an investigation of several virtualization technologies, namely Xen, KVM, VirtualBox, and in part VMWare. Each hypervisor

is compared alongside one another with base-metal as a control and (with the exeption of VMWare) run through a number of High Performance benchmarking tools.

3.4 Feature Comparison

With the wide array of potential choices of virtualization technologies available, its often difficult for potential users to identify which platform is best suited for their needs. In order to simplify this task, we provide a detailed comparison chart between Xen 3.1, KVM from RHEL5, VirtualBox 3.2 and VMWWare ESX in Figure 2.

	Xen	KVM	VirtualBox	VMWare
Para-virtualization	Yes	No	No	No
Full virtualization	Yes	Yes	Yes	Yes
Host CPU	x86, x86-64, IA-64	x86, x86-64,IA64,PPC	x86, x86-64	x86, x86-64
Guest CPU	x86, x86-64, IA-64	x86, x86-64,IA64,PPC	x86, x86-64	x86, x86-64
Host OS	Linux, UNIX	Linux	Windows, Linux, UNIX	Proprietary UNIX
Guest OS	Linux, Windows, UNIX	Linux, Windows, UNIX	Linux, Windows, UNIX	Linux, Windows, UNIX
VT-x / AMD-v	Opt	Req	Opt	Opt
Cores supported	128	16	32	8
Memory supported	4TB	4TB	16GB	64GB
3D Acceleration	Xen-GL	VMGL	Open-GL	Open-GL, DirectX
Live Migration	Yes	Yes	Yes	Yes
License	GPL	GPL	GPL/proprietary	Proprietary

Figure 3.1 A comparison chart between Xen, KVM, VirtualBox, and VMWare ESX

The first point of investigation is the virtualization method of each VM. Each hypervisor supports full virtualization, which is now common practice within most x86 virtualization deployments today. Xen, originating as a para-virtualized VMM, still supports both types, however full virtualization is often preferred as it does not require the manipulation of the guest kernel in any way. From the Host and Guest CPU lists, we see that x86 and, more specifically, x86-64/amd64 guests are all universally supported. Xen and KVM both suport Itanium-64 architectures for full virtualization (due to both hypervisors dependency on QEMU), and KVM also claims support for some recent PowerPC architectures. However, we concern ourselves only

with x86-64 features and performance, as other architectures are out of the scope of this manuscript. Of the x86-64 platforms, KVM is the only hypervisor to require either Intel VT-X or AMD-V instruction sets in order to operate. VirtualBox and VMWare have internal mechanisms to provide full virtualization even without the virtualization instruction sets, and Xen can default back to para-virtualized guests.

Next, we consider the host environments for each system. As Linux is the primary OS type of choice within HPC deployments, its key that all hypervisors support Linux as a guest OS, and also as a host OS. As VMWare ESX is meant to be a virtualization-only platform, it is built upon a specially configured Linux/UNIX proprietary OS specific to its needs. All other hypervisors support Linux as a host OS, with VirtualBox also supporting Windows, as it was traditionally targeted for desktop-based virtualization. However, as each hypervisor uses VT-X or AMD-V instructions, each can support any modern OS targeted for x86 platforms, including all variants of Linux, Windows, and UNIX.

While most hypervisors have desirable host and guest OS support, hardware support within a guest environment varies drastically. Within the HPC environment, virtual CPU (vCPU) and maximum VM memory are critical aspects to choosing the right virtualization technology. In this case, Xen is the first choice as it supports up to 128 vCPUs and can address 4TB of main memory in 64-bit modes, more than any other. VirtualBox, on the other hand, supports only 32 vCPUs and 16GB of addressable RAM per guest OS, which may lead to problems when looking to deploy it on large multicore systems. KVM also faces an issue with the number of vCPU supported limited to 16, recent reports indicate it is only a soft limit [144], so deploying KVM in an SMP environment may not be a significant hurdle. Furthermore, all hypervisors provide some 3D acceleration support (at least for OpenGL) and support live migration across homogeneous nodes, each with varying levels of success.

Another vital juxtaposition of these virtualization technologies is the license agreements for its applicability within HPC deployments. Xen, KVM, and VirtualBox are provided for free under the GNU Public License (GPL) version 2, so they are open to use and modification by anyone within the community, a key feature for many potential users. While VirtualBox is under GPL, it has recently also offered with additional features under a more proprietary license dictated by Oracle since its acquirement from Sun last year. VMWare, on the other hand, is completely proprietary with an extremely limited licensing scheme that even prevents the authors from willfully publishing any performance benchmark data without specific and prior approval. As such, we have neglected VMWare form the remainder of this chapter. Whether going with a proprietary or open source hypervisor, support can be acquired (usually for an additional cost) with ease from each option.

3.4.1 Usability

While side by side feature comparison may provide crucial information about a potential user's choice of hypervisor, that may also be interested in its ease of installation and use. We will take a look at each hypervisor from two user perspectives, a systems administrator and normal VM user.

One of the first things on any system administrator's mind on choosing a hypervisor is the installation. For all of these hypervisors, installation is relatively painless. For the FutureGrid support group, KVM and VirualBox are the easiest of the all tested hypervisors to install, as there are a number of supported packages available and installation only requires the addition of one or more kernel modules and the support software. Xen, while still supported in binary form by many Linux distributions, is actually much more complicated. This is because Xen requires a full modification to the kernel itself, not just a module. Loading a new kernel into the boot process

which may complicate patching and updating later in the system's maintenance cycle. VMWare ESX, on the other hand, is entirely separate from most other installations. As previously noted, ESX is actually a hypervisor and custom UNIX host OS combined, so installation of ESX is likewise to installing any other OS from scratch. This may be either desirable or adverse, depending on the system administrator's usage of the systems and VMWare's ability to provide a secure and patched environment.

While system administrators may be concerned with installation and maintenance, VM users and Cloud developers are more concerned with daily usage. The first thing to note about all of such virtualiation technologies is they are supported (to some extent) by the libvirt API [145]. Libvirt is commonly used by many of today's IaaS Cloud offerings, including Nimbus, Eucalyptus, OpenNebula and OpenStack. As such, the choice of hypervisor for Cloud developer's is less of an issue, so long as the hypervisor supports the features they desire. For individual command line usage of each tool, it varies quite a bit more. Xen does provide their own set of tools for controlling and monitoring guests, and seem to work relatively well but do incur a slight learning curve. KVM also provides its own CLI interface, and while it is often considered less cumbersome it provides less advanced features directly to users, such as power management or quick memory adjustment (however this is subject to personal opinion). One advantage of KVM is each guest actually runs as a separate process within the host OS, making it easy for a user to manage and control the VM inside the host if KVM misbehaves. VirtualBox, on the other hand, provides the best command line and graphical user interface. The CLI, is especially well featured when compared to Xen and KVM as it provides clear, decisive and well documented commands, something most HPC users and system administrators alike will appreciate. VMWare provides a significantly enhanced GUI as well as a Web-based ActiveX client interface that allows users to easily operate the VMWare host remotely. In summary, there

is a wide variance of interfaces provided by each hypervisor, however we recommend Cloud developers to utilize the libvirt API whenever possible.

3.5 Experimental Design

In order to provide an unaltered and unbiased review of these virtualization technologies for Clouds, we need to outline a neutral testing environment. To make this possible, we have chosen to use FutureGrid as our virtualization and cloud test-bed.

3.5.1 The FutureGrid Project

FutureGrid (FG) [146] provides computing capabilities that enable researchers to tackle complex research challenges related to the use and security of Grids and Clouds. These include topics ranging from authentication, authorization, scheduling, virtualization, middleware design, interface design and cybersecurity, to the optimization of Grid-enabled and cloud-enabled computational schemes for researchers in astronomy, chemistry, biology, engineering, atmospheric science and epidemiology.

The test-bed includes a geographically distributed set of heterogeneous computing systems, a data management system that will hold both metadata and a growing library of software images necessary for Cloud computing, and a dedicated network allowing isolated, secure experiments, as seen in Figure 3.2. The test-bed supports virtual machine-based environments, as well as operating systems on native hardware for experiments aimed at minimizing overhead and maximizing performance. The project partners are integrating existing open-source software packages to create an easy-to-use software environment that supports the instantiation, execution and recording of grid and cloud computing experiments.

One of the goals of the project is to understand the behavior and utility of Cloud computing approaches. However, it is not clear at this time which of these toolkits will

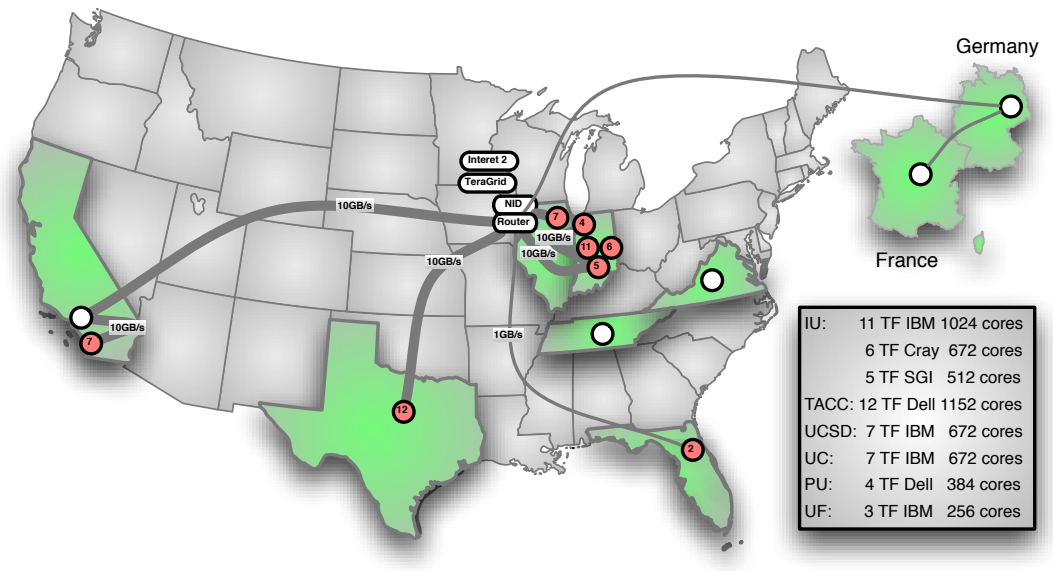


Figure 3.2 FutureGrid Participants and Resources

become the users' choice toolkit. FG provides the ability to compare these frameworks with each other while considering real scientific applications [44]. Hence, researchers are be able to measure the overhead of cloud technology by requesting linked experiments on both virtual and bare-metal systems, providing valuable information that help decide which infrastructure suits their needs and also helps users that want to transition from one environment to the other. These interests and research objectives make the FutureGrid project the perfect match for this work. Furthermore, we expect that the results gleaned from this chapter will have a direct impact on the FutureGrid deployment itself.

3.5.2 Experimental Environment

Currently, one of FutureGrid's latest resources is the *India* system, a 256 CPU IBM iDataPlex machine consisting of 1024 cores, 2048 GB of ram, and 335 TB of storage within the Indiana University Data Center. In specific, each compute node of India

has two Intel Xeon 5570 quad core CPUs running at 2.93Ghz, 24GBs of Ram, and a QDR InfiniBand connection. A total of four nodes were allocated directly from India for these experiments. All were loaded with a fresh installation of Red Hat Enterprise Linux server 5.5 x86_64 with the 2.6.18-194.8.1.el5 kernel patched. Three of the four nodes were installed with different hypervisors; Xen version 3.1, KVM (build 83), and VirtualBox 3.2.10, and the forth node was left as-is to act as a control for bare-metal native performance.

Each guest virtual machine was also built using Red Hat EL server 5.5 running an unmodified kernel using full virtualization techniques. All tests were conducted giving the guest VM 8 cores and 16GB of ram to properly span a compute node. Each benchmark was run a total of 20 times, with the results averaged to produce consistent results, unless indicated otherwise.

3.5.3 Benchmarking Setup

As this chapter aims to objectively evaluate each virtualization technology from a side-by-side comparison as well as from a performance standpoint, the selection of benchmarking applications is critical.

The performance comparison of each virtual machine is based on two well known industry standard performance benchmark suites; HPCC and SPEC. These two benchmark environments are recognized for their standardized reproducible results in the HPC communit, and the National Science Foundation (NSF), Department of Energy (DOE), and DARPA are all sponsors of the HPCC benchmarks. The following benchmarks provide a means to stress and compare processor, memory, inter-process communication, network, and overall performance and throughput of a system. These benchmarks were selected due to their importance to the HPC community sinse they are often directly correlated with overall application performance [147].

HPCC Benchmarks

The HPCC Benchmarks [148, 149] are an industry standard for performing benchmarks for HPC systems. The benchmarks are aimed at testing the system on multiple levels to test their performance. It consists of 7 different tests:

- *HPL* - The Linpack TPP benchmark measures the floating point rate of execution for solving a linear system of equations. This benchmark is perhaps the most important benchmark within HPC today, as it is the basis of evaluation for the Top 500 list [8].
- DGEMM - Measures the floating point rate of execution of double precision real matrix-matrix multiplication.
- *STREAM* - A simple synthetic benchmark program that measures sustainable memory bandwidth (in GB/s) and the corresponding computation rate for simple vector kernel.
- *PTRANS* - Parallel matrix transpose exercises the communications where pairs of processors communicate with each other simultaneously. It is a useful test of the total communications capacity of the network.
- *RandomAccess* - Measures the rate of integer random updates of memory (GUPS).
- *FFT* - Measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform (DFT).
- *Communication bandwidth and latency* - A set of tests to measure latency and bandwidth of a number of simultaneous communication patterns; based on b_eff (effective bandwidth benchmark).

This benchmark suite uses each test to stress test the performance on multiple aspects of the system. It also provides reproducible results which can be verified by other vendors. This benchmark is used to create the Top 500 list [8] which is the list of the current top supercomputers in the world. The results that are obtained from these benchmarks provide an unbiased performance analysis of the hypervisors. Our results provide insight on inter-node PingPong bandwidth, PingPong latency, and FFT calculation performance.

SPEC Benchmarks

The Standard Performance Evaluation Corporation (SPEC) [150, 151] is the other major standard for evaluation of benchmarking systems. SPEC has several different testing components that can be utilized to benchmark a system. For our benchmarking comparison we will use the SPEC OMP2001 because it appears to represent a vast array of new and emerging parallel applications wile simultaniously providing a comparison to other SPEC benchmarks. SPEC OMP continues the SPEC tradition of giving HPC users the most objective and representative benchmark suite for measuring the performance of SMP (shared memory multi-processor) systems.

- The benchmarks are adapted from SPEC CPU2000 and contributions to its search program.
- The focus is to deliver systems performance to real scientific and engineering applications.
- The size and runtime reflect the needs of engineers and researchers to model large complex tasks.
- Two levels of workload characterize the performance of medium and large sized systems.

- Tools based on the SPEC CPU2000 toolset make these the easiest ever HPC tests to run.
- These benchmarks place heavy demands on systems and memory.

3.6 Performance Comparison

The goal of this chapter is to effectively compare and contrast the various virtualization technologies, specifically for supporting HPC-based Clouds. The first set of results represent the performance of HPCC benchmarks. Each benchmark was run a total of 20 times, and the mean values taken with error bars represented using the standard deviation over the 20 runs. The benchmarking suite was built using the Intel 11.1 compiler, uses the Intel MPI and MKL runtime libraries, all set with defaults and no optimizations whatsoever.

We open first with High Performance Linpack (HPL), the de-facto standard for comparing resources. In Figure 3.3, we can see the comparison of Xen, KVM, and Virtual Box compared to native bare-metal performance. First, we see that native is capable of around 73.5 Gflops which, with no optimizations, achieves 75% of the theoretical peak performance. Xen, KVM and VirtualBox perform at 49.1, 51.8 and 51.3 Gflops, respectively when averaged over 20 runs. However Xen, unlike KVM and VirtualBox, has a high degree of variance between runs. This is an interesting phenomenon for two reasons. First, this may impact performance metrics for other HPC applications and cause errors and delays between even pleasingly-parallel applications and add to reducer function delays. Second, this wide variance breaks a key component of Cloud computing providing a specific and predefined quality of service. If performance can sway as widely as what occurred for Linpack, then this may have a negative impact on users.

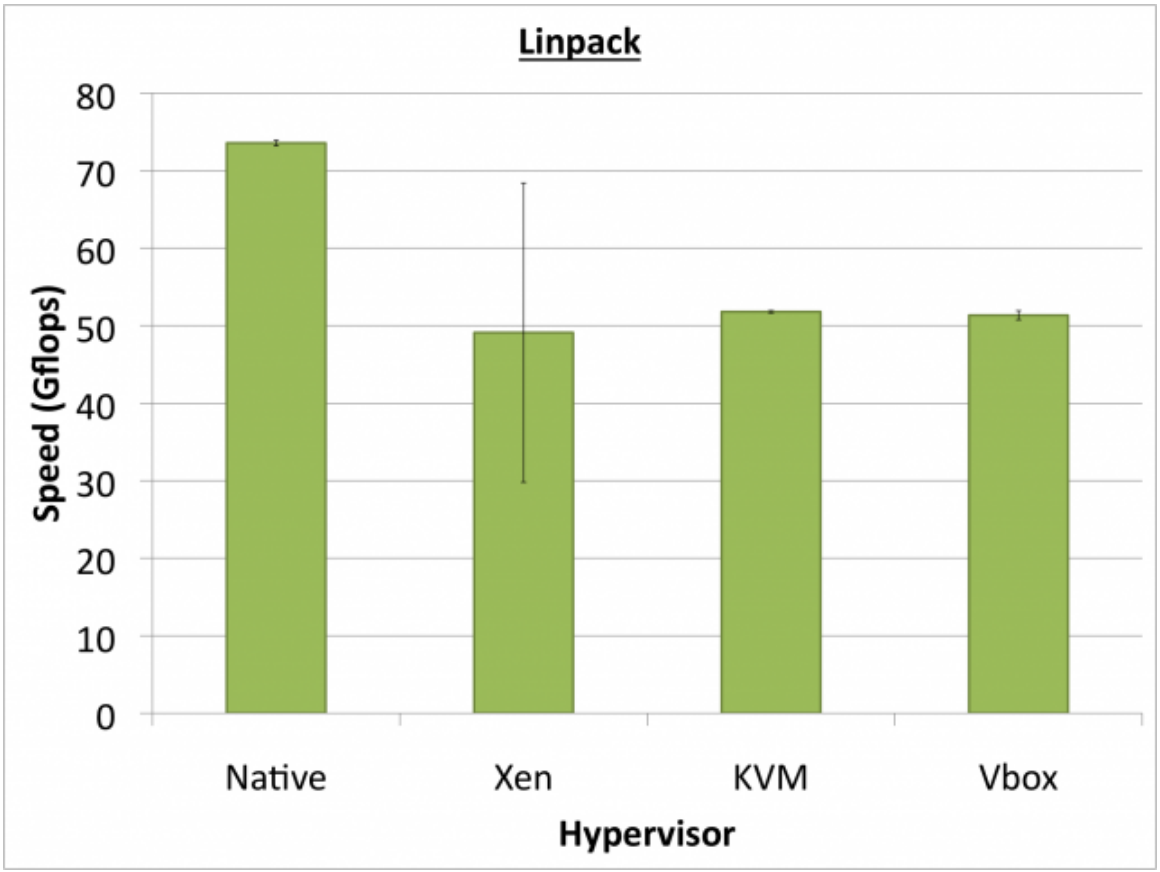


Figure 3.3 Linpack performance

Next, we turn to another key benchmark within the HPC community, Fast Fourier Transforms (FFT). Unlike the synthetic Linpack benchmark, FFT is a specific, purposeful benchmark which provides results which are often regarded as more relative to a user's real-world application than HPL. From Figure 3.4, we can see rather distinct results from what was previously provided by HPL. Looking at Star and Single FFT, its clear performance across all hypervisors is roughly equal to bare-metal performance, a good indication that HPC applications may be well suited for use on VMs. The results for MPI FFT also show similar results, with the exception of Xen, which has a decreased performance and high variance as seen in the HPL benchmark. Our current hypothesis is that there is an adverse affect of using Intel's MPI runtime on Xen, however the investigation is still ongoing.

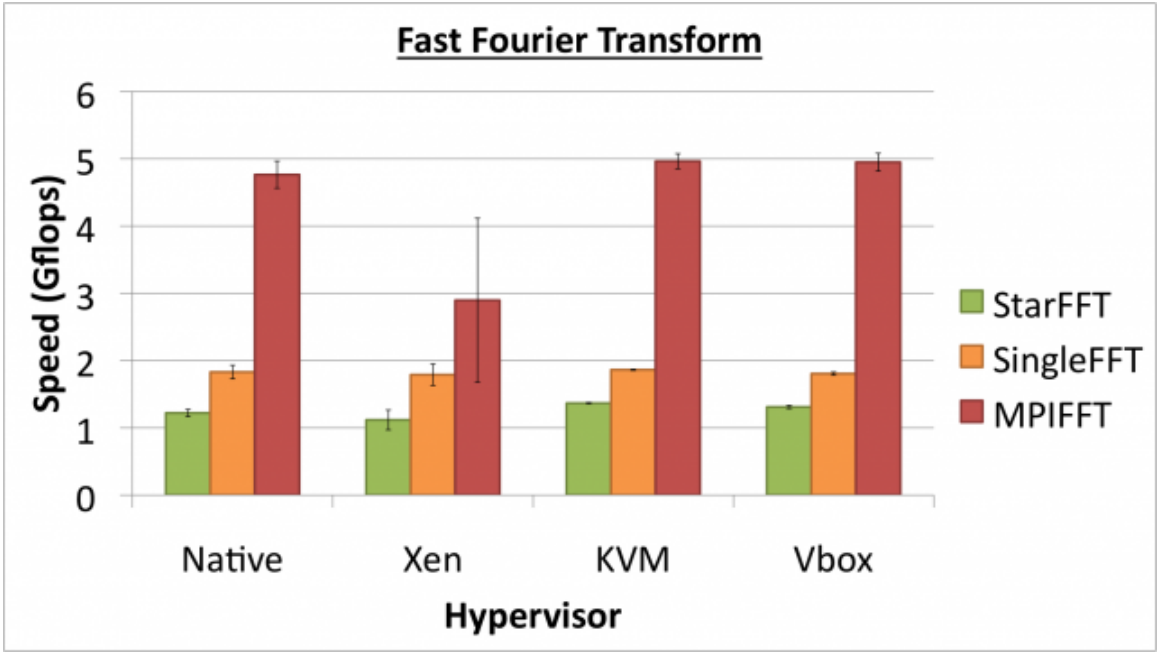


Figure 3.4 Fast Fourier Transform performance

Another useful benchmark illustrative of real-world performance between bare-metal performance and various hypervisors are the ping-pong benchmarks. These benchmarks measure the bandwidth and latency of passing packets between multiple CPUs. With this experiment, all ping-pong latencies are kept within a given node, rather than over the network. This is done to provide further insight into the CPU and memory overhead withing each hypervisor. From Figure 3.5 the intranode bandwidth performance is uncovered, with some interesting distinctions between each hypervisor. First, Xen performs, on average, close to native speeds, which is promising for the hypervisor. KVM, on the other hand, shows consistent overhead proportional to native performance across minimum, average, and maximum bandwidth. VirtualBox, on the other hand, performs well, in fact too well to the point that raises alarm. While the minimum and average bandwidths are within native performance, the maximum bandwidth reported by VirtualBox is significantly greater than native measurements, with a large variance. After careful examination, it appears this is

due to how VirtualBox assigns its virtual CPUs. Instead of locking a virtual CPU to a real CPU, a switch may occur which could benefit on the off-chance the two CPU's in communication between a ping-pong test could in fact be the same physical CPU. The result would mean the ping-pong packet would remain in cache and result in a higher perceived bandwidth than normal. While this effect may be beneficial for this benchmark, it may only be an illusion towards the real performance gleaned from the VirtualBox hypervisor.

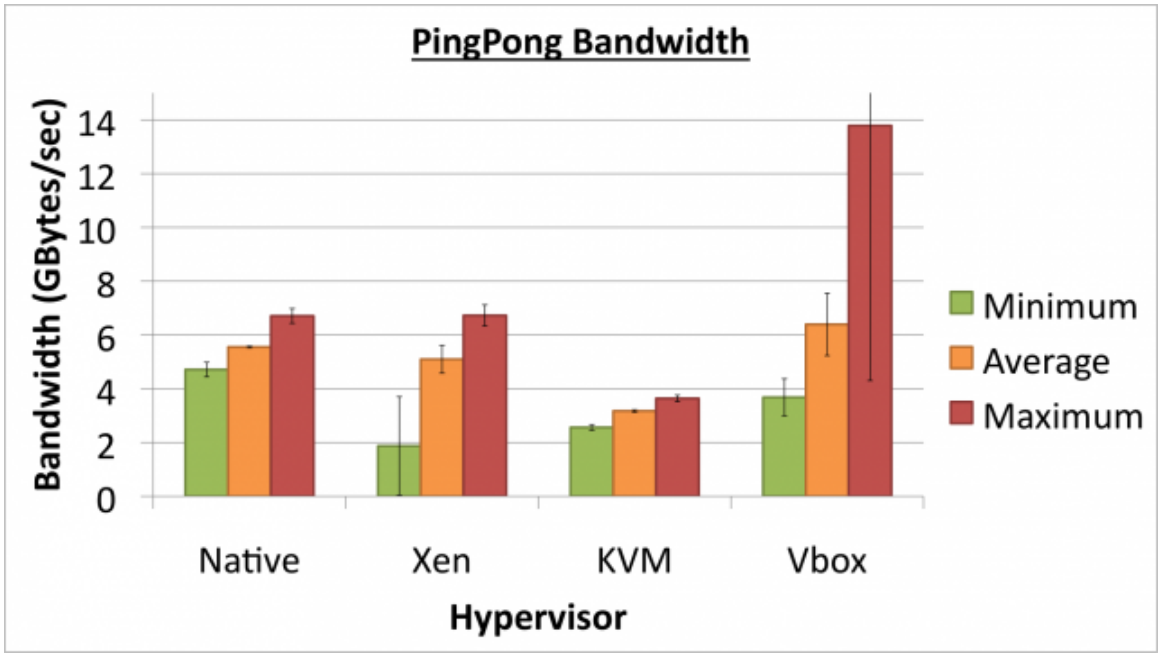


Figure 3.5 Ping Pong bandwidth performance

The Bandwidth may in fact be important within the ping-ping benchmark, but the latency between each ping-pong is equally useful in understanding the performance impact of each virtualization technology. From Figure 3.6, we see KVM and VirtualBox have near-native performance; another promising result towards the utility of hypervisors within HPC systems. Xen, on the other hand, has extremely high latencies, especially at for maximum latencies, which in turn create a high variance within the average latency within the VM's performance.

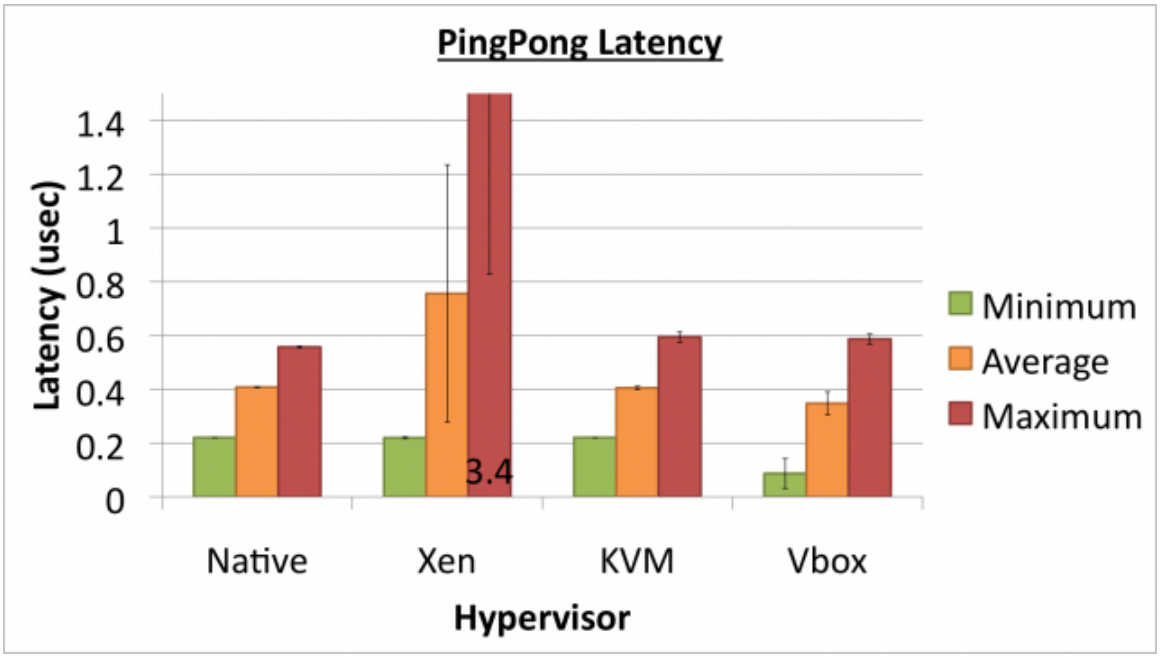


Figure 3.6 Ping Pong latency performance (lower is better)

While the HPCC benchmarks provide a comprehensive view for many HPC applications including Linpack and FFT using MPI, performance of intra-node SMP applications using OpenMP is also investigated. Figure 3.7 illustrates SPEC OpenMP performance across the VMs we concentrate on, as well as baseline native performance. First, we see that the combined performance over all 11 applications executed 20 times yields the native testbed with the best performance at a SPEC score of 34465. KVM performance comes close with a score of 34384, which is so similar to the native performance that most users will never notice the difference. Xen and VirtualBox both perform notably slower with scores of 31824 and 31695, respectively, however this is only an 8% performance drop compared to native speeds. Further results can be found on the SPEC website [152].

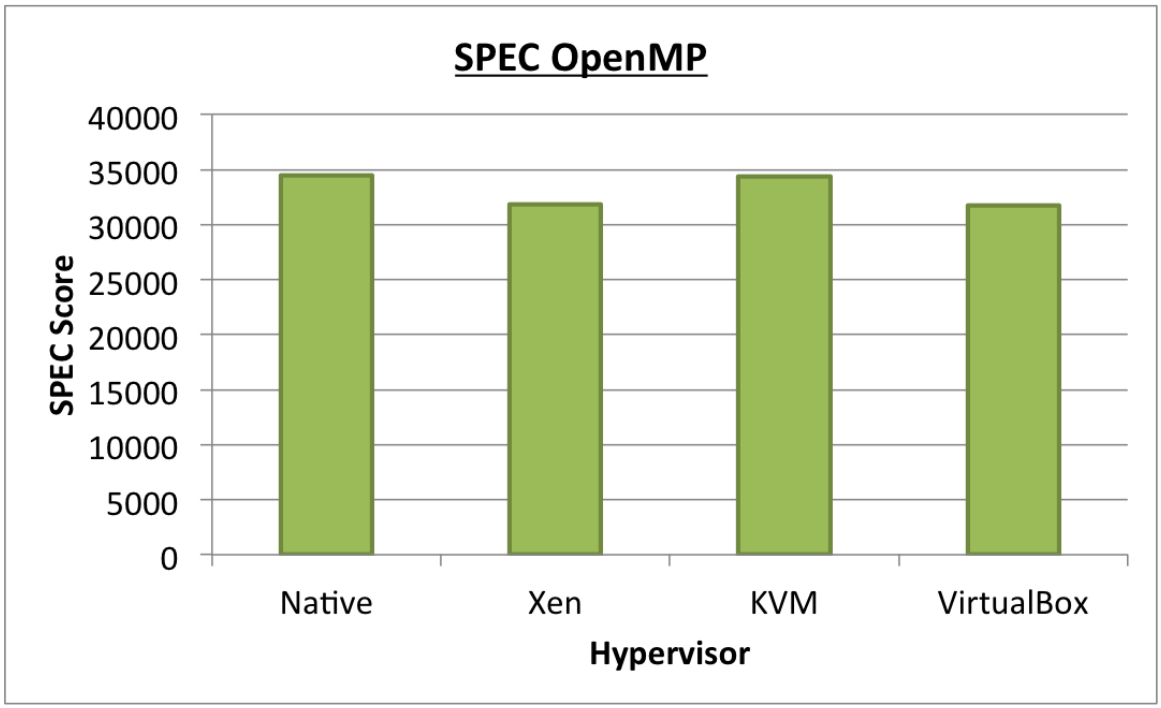


Figure 3.7 Spec OpenMP performance

3.7 Discussion

The primary goal of this chapter is to evaluate the viability of virtualization within HPC. After our analysis, the answer seems to be a resounding "yes." However, we also hope to select the best virtualization technology for such an HPC environment. In order to do this, we combine the feature comparison along with the performance results, and evaluate the potential impact within the FutureGrid testbed.

From a feature standpoint, most of today's virtualization technologies fit the bill for at least small scale deployment, including VMWare. In short, each support Linux x86_64 platforms, use VT-X technology for full virtualization, and support live migration. Due to VMWare's limited and costly licensing, it is immediately out of contention for most HPC deployments. From a CPU and memory standpoint, Xen seems to provide the best expandability, supporting up to 128 cpus and 4TB of ad-

dressable RAM. So long as KVM's vCPU limit can be extended, it too shows promise as a feature-full virtualization technology. One of Virtualbox's greatest limitations was the 16GB maximum memory allotment for individual guest VMs, which actually limited us from giving VMs more memory for our performance benchmarks. If this can be fixed and Oracle does not move the product into the proprietary market, VirtualBox may also stand a chance for deployment in HPC environments.

	Xen	KVM	VirtualBox
Linpack	3	1	2
FFT	3	1	2
Bandwidth	2	3	1
Latency	3	2	1
OpenMP	2	1	3
Total Rating	13	8	9

Figure 3.8 Benchmark rating summary (lower is better)

From the benchmark results previously described, the use of hypervisors within HPC-based Cloud deployments is mixed batch. Figure 3.8 summarizes the results based on a 1-3 rating, 1 being best and 3 being worst. While Linpack performance seems to take a significant performance impact across all hypervisors, the more practical FFT benchmarks seem to show little impact, a notably good sign for virtualization as a whole. The ping-pong bandwidth and latency benchmarks also seem to support this theory, with the exception of Xen, who's performance continually has wide fluctuations throughout the majority of the benchmarks. OpenMP performance through the SPEC OMP benchmarking suite also shows promising results for the use of hypervisors in general, with KVM taking a clear lead by almost matching native

speeds.

While Xen is typically regarded as the most widely used hypervisor, especially within academic clouds and grids, it's performance has shown lack considerably when compared to either KVM or VirtualBox. In particular, Xen's wide and unexplained fluctuations in performance throughout the series of benchmarks suggests that Xen may not be the best choice for building a lasting quality of service infrastructure upon. From Figure 3.8, KVM rates the best across all performance benchmarks, making it the optimal choice for *general* deployment in an HPC environment. Furthermore, this work's illustration of the variance in performance among each benchmark and the applicability of each benchmark towards new applications may make possible the ability to preemptively classify applications for accurate prediction towards the ideal virtualized Cloud environment. We hope to further investigate this concept through the use of the FutureGrid experiment management framework at a later date.

In summary, it is the authors' projection that KVM is the best overall choice for use within HPC Cloud environments. KVM's feature-rich experience and near-native performance makes it a natural fit for deployment in an environment where usability and performance are paramount. Within the FutureGrid project specifically, we hope to deploy the KVM hypervisor across our Cloud platforms in the near future, as it offers clear benefits over the current Xen deployment. Furthermore, we expect these findings to be of great importance to other public and private Cloud deployments, as system utilization, Quality of Service, operating cost, and computational efficiency could all be improved through the careful evaluation of underlying virtualization technologies.

Chapter 4

Evaluating GPU Passthrough in Xen for High Performance Cloud Computing

4.1 Abstract

With the advent of virtualization and Infrastructure-as-a-Service (IaaS), the broader scientific computing community is considering the use of clouds for their technical computing needs. This is due to the relative scalability, ease of use, advanced user environment customization abilities clouds provide, as well as many novel computing paradigms available for data-intensive applications. However, there is concern about a performance gap that exists between the performance of IaaS when compared to typical high performance computing (HPC) resources, which could limit the applicability of IaaS for many potential scientific users.

Most recently, general-purpose graphics processing units (GPGPUs or GPUs) have become commonplace within high performance computing. We look to bridge the gap between supercomputing and clouds by providing GPU-enabled virtual machines (VMs) and investigating their feasibility for advanced scientific computation. Specifically, the Xen hypervisor is utilized to leverage specialized hardware-assisted I/O virtualization and PCI passthrough in order to provide advanced HPC-centric Nvidia GPUs directly in guest VMs. This methodology is evaluated by measuring the performance of two Nvidia Tesla GPUs within Xen VMs and comparing to bare-metal

hardware. Results show PCI passthrough of GPUs within virtual machines is a viable use case for many scientific computing workflows, and could help support high performance cloud infrastructure in the near future.

4.2 Introduction

Cloud computing [4] has established itself as a prominent paradigm within the realm of Distributed Systems [153] in a very short period of time. Clouds are an internet-based solution that provide computational and data models for utilizing resources, which can be accessed directly by users on demand in a uniquely scalable way. Cloud computing functions by providing a layer of abstraction on top of base hardware to enable a new set of features that are otherwise intangible or intractable. These benefits and features include Scalability, a guaranteed Quality of Service (QoS), cost effectiveness, and direct user customization via a simplified user interface [65].

While the origin of cloud computing is based in industry through solutions such as Amazon's EC2 [154], Google's MapReduce [155], and Microsoft's Azure [156], the paradigm has since become integrated in all areas of science and technology. Most notably, there is an increasing effort within the High Performance Computing (HPC) community to leverage the utility of clouds for advanced scientific computing to solve a number of challenges still standing in the field. This can be clearly seen in large-scale efforts such as the FutureGrid project [157], the Magellan project [48], and through various other Infrastructure-as-a-Service projects including OpenStack [158], Nimbus [75], and Eucalyptus [159].

Within HPC, there has also been a notable movement toward dedicated accelerator cards such as general purpose graphical processing units (GPGPUs, or GPUs) to enhance scientific computation problems by upwards of two orders of magnitude. This is accomplished through dedicated programming environments, compilers, and

libraries such as CUDA [160] from Nvidia as well as the OpenCL effort [161]. When combining GPUs in an otherwise typical HPC environment or supercomputer, major gains in performance and computational ability have been reported in numerous fields [162, 163], ranging from Astrophysics to Bioinformatics. Furthermore, these gains in computational power have also reportedly come at an increased performance-per-watt [164], a metric that is increasingly important to the HPC community as we move closer to exascale computing [165] where power consumption is quickly becoming the primary constraint.

With the advent of both clouds and GPUs within the field of scientific computing, there is an immediate and ever-growing need to provide heterogeneous resources, most immediately GPUs, within a cloud environment in the same scalable, on-demand, and user-centric way that many cloud users are already accustomed to [166]. While this task alone is nontrivial, it is further complicated by the high demand for performance within HPC. As such, it is performance that is paramount to the success of deploying GPUs within cloud environments, and thus is the central focus of this work.

The rest of this chapter is organized as follows. First, in Section 2, we discuss the related research and the options currently available for providing GPUs within a virtualized cloud environment. In Section 3, we discuss the methodology for providing GPUs directly within virtual machines. In Section 4 we outline the evaluation of the given methodology using two different Nvidia Tesla GPUs and compare to the best-case native application in Section 5. Then, we discuss the implications of these results in Section 6 and consider the applicability of each method within a production cloud system. Finally, we conclude with our findings and suggest directions for future work.

4.3 Virtual GPU Directions

Recently, GPU programming has been a primary focus for numerous scientific computing applications. Significant progress has been accomplished in many different workloads, both in science and engineering, based on parallel abilities of GPUs for floating point operations and very high on-GPU memory bandwidth. This hardware, coupled with CUDA and OpenCL programming frameworks, has led to an explosion of new GPU-specific applications. In some cases, GPUs outperform even the fastest multicore counterparts by an order of magnitude [167]. In addition, further research could leverage the per-node performance of GPU accelerators with the high speed, low latency interconnects commonly utilized in supercomputers and clusters to create a hybrid GPU + MPI class of applications. The number of distributed GPU applications is increasing substantially in supercomputing, usually scaling many GPUs simultaneously [168].

Since the establishment of cloud computing in industry, research groups have been evaluating its applicability to science [3]. Historically, HPC and Grids have been on similar but distinct paths within distributed systems, and have concentrated on performance, scalability, and solving complex, tightly coupled problems within science. This has led to the development of supercomputers with many thousands of cores, high speed, low latency interconnects, and sometimes also coprocessors and FPGAs [169, 170]. Only recently have these systems been evaluated from a cloud perspective [48]. An overarching goal exists to provide HPC Infrastructure as its own service (HPCaaS) [171], aiming to classify and limit the overhead of virtualization, and reducing the bottlenecks classically found in CPU, memory, and I/O operations within hypervisors [49, 172]. Furthermore, the transition from HPC to cloud computing becomes more complicated when we consider adding GPUs to the equation.

GPU availability within a cloud is a new concept that has sparked a large amount of interest within the community. The first successfully deployment of GPUs within a cloud environment was the Amazon EC2 GPU offering. A collaboration between Nvidia and Citrix also exists to provide cloud-based gaming solutions to users using the new Kepler GPU architecture [173]. However, this is currently not targeted towards HPC applications.

The task of providing a GPU accelerator for use in a virtualized cloud environment is one that presents a myriad of challenges. This is due to the complicated nature of virtualizing drivers, libraries, and the heterogeneous offerings of GPUs from multiple vendors. Currently, two possible techniques exist to fill the gap in providing GPUs in a cloud infrastructure: back-end I/O virtualization, which this chapter focuses on, and Front-end remote API invocation.

4.3.1 Front-end Remote API invocation

One method for using GPUs within a virtualized cloud environment is through front-end library abstractions, the most common of which is remote API invocation. Also known as API remoting or API interception, it represents a technique where API calls are intercepted and forwarded to a remote host where the actual computation occurs. The results are then returned to the front-end process that spawned the invocation, potentially within a virtual machine. The goal of this method is to provide an emulated device library where the actual computation is offloaded to another resource on a local network.

Front-end remote APIs for GPUs have been implemented by a number of different technologies for different uses. To solve the problem of graphics processing in VMs, VMWare [174] has developed a device-emulation approach that emulates the Direct3D and OpenGL calls to leverage the host OS graphics processing capabili-

ties to provide a 3D environment within a VM. API interception through the use of wrapper binaries has also been implemented by technologies such as Chromium [175], and Blink. However these graphics processing front-end solutions are not suitable for general purpose scientific computing, as they do not expose interfaces that CUDA or OpenCL can use.

Currently, efforts are being made to provide a front-end remote API invocation solutions for the CUDA programming architecture. vCUDA [57] was the first of such technologies to enable transparent access of GPUs within VMs by API call interception and redirection of the CUDA API. vCUDA substitutes the CUDA runtime library and supports a transmission mode using XMLRPC, as well as a sharing mode that is built on VMRPC, a dedicated remote procedure call architecture for VMM platforms. This share model can leads to better performance, especially as the volume of data increases, although there may be limitations in VMM interoperability.

Like vCUDA, gVirtuS uses API interception to enable transparent CUDA, OpenCL, and OpenGL support for Xen, KVM, and VMWare virtual machines [176]. gVirtuS uses a front-end/back-end model to provide a VMM-independent abstraction layer to GPUs. Data transport from gVirtuS' front-end to the back-end is accomplished through a combination of shared memory, sockets, or other hypervisor-specific APIs. gVirtuS' primary disadvantage is in its decreased performance in host-to-device and device-to-host data movement due to overhead of data copies to and from its shared memory buffers. Recent work has also enabled the dynamic sharing of GPUs by leveraging the gVirtus back-end system with relatively good results [177], however process-level GPU resource sharing is outside the scope of this manuscript.

rCUDA [56], a recent popular remote CUDA framework, also provides remote API invocation to enable VMs to access remote GPU hardware by using a sockets based implementation for high-speed near-native performance of CUDA based applications.

rCUDA recently added support for using InfiniBand's high speed, low latency network to increase performance for CUDA applications with large data volume requirements. rCUDA version 4.1 also supports the CUDA runtime API version 5.0, which supports peer device memory access and unified addressing. One drawback of this method is that rCUDA cannot implement the undocumented and hidden functions within the runtime framework, and therefore does not support all CUDA C extensions. While rCUDA provides some support tools, native execution of CUDA programs is not possible and programs need to be recompiled or rewritten to use rCUDA. Furthermore, like gVirtuS and many other solutions, performance between host-to-device data movement is only as fast as the underlying interconnect, and in the best case with native RDMA InfiniBand, is roughly half as fast as native PCI Express usage when using the standard QDR InfiniBand.

4.3.2 Back-end PCI passthrough

Another approach to using a GPU in a virtualized environment is to provide a VM with direct access to the GPU itself, instead of relying on a remote API. This chapter focuses on such an approach. Devices on a host's PCI-express bus are virtualized using directed I/O virtualization technologies recently implemented by chip manufacturers, and then direct access is relinquished upon request to a guest VM. This can be accomplished using the VT-d and IOMMU instruction sets from Intel and AMD, respectively. This mechanism, typically called PCI passthrough, uses a memory management unit (MMU) to handle direct memory access (DMA) coordination and interrupt remapping directly to the guest VM, thus bypassing the host entirely. With host involvement being nearly non-existent, near-native performance of the PCI device within the guest VM can be achieved, which is an important characteristic for using a GPU within a cloud infrastructure.

PCI passthrough itself has recently become a standard technique for many other I/O systems such as storage or network controllers. However, GPUs (even from the same vendor) have additional legacy VGA compatibility issues and non-standard low-level interface DMA interactions that make direct PCI passthrough nontrivial. VMWare has started use of a vDGA system for hardware GPU utilization, however it remains in tech preview and only documentation for Windows VMs is present [174]. In our experimentation, we have found that the Xen hypervisor provides a good platform for performing PCI passthrough of GPU devices to VMs due to its open nature, extensive support, and high degree of reconfigurability. Work with Xen in [178] gives hints at good performance for PCI passthrough in Xen, however further evaluation with independent benchmarks is needed when looking at scientific computing with GPUs.

Today's GPUs can provide a variety of frameworks for application programmers to use. Two common solutions are CUDA and OpenCL. CUDA, or the Compute Unified Device Architecture, is a framework for creating and running parallel applications on Nvidia GPUs. OpenCL provides a more generic and open framework for parallel computation on CPUs and GPUs, and is available for a number of Nvidia, AMD, and Intel GPUs and CPUs. While OpenCL provides a more robust and portable solution, many HPC applications utilize the CUDA framework. As such, we focus only on Nvidia based CUDA-capable GPUs as they offer the best support for a wide array of programs, although this work is not strictly limited to Nvidia GPUs.

4.4 Implementation

In this chapter we use a specific host environment to enable PCI passthrough. First, we start with the Xen 4.2.2 hypervisor on Centos 6.4 and a custom 3.4.50-8 Linux kernel with Dom0 Xen support. Within the Xen hypervisor, GPU devices are seized

upon boot and assigned to the xen-pciback kernel module. This process blocks the host devices form loading the Nvidia or nouveau drivers, keeping the GPUs uninitialized and therefore able to be assigned to DomU VMs.

Xen, like other hypervisors, provides a standard method of passing through PCI devices to guest VMs upon creation. When assigning a GPU to a new VM, Xen loads a specific VGA BIOS to properly initialize the device enabling DMA and interrupts to be assigned to the guest VM. Xen also relinquishes control of the GPU via the xen-pciback module. From there, the Linux Nvidia drivers are loaded and the device is able to be used as expected within the guest. Upon VM termination, the xen-pciback module re-seizes the GPU and the devices can be re-assigned to new VMs in the future.

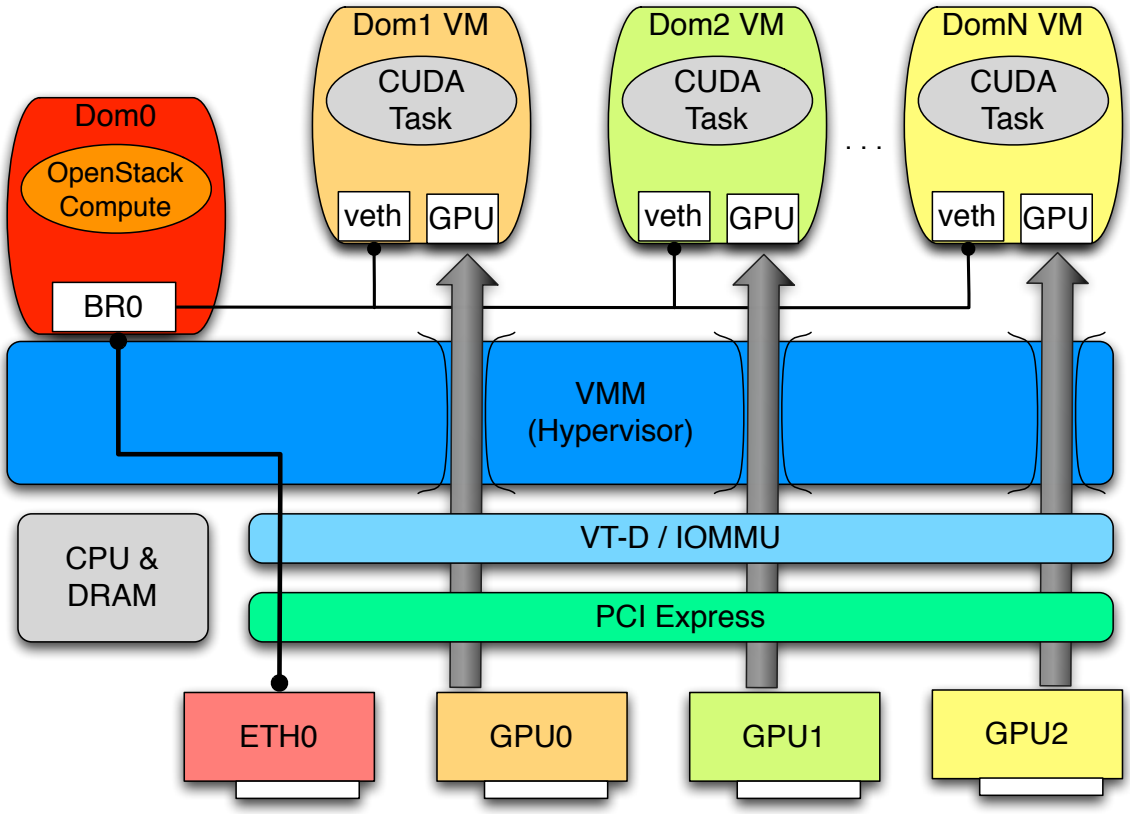


Figure 4.1 GPU PCI passthrough within the Xen Hypervisor

This mechanism of PCI passthrough for GPUs can be implemented using multiple devices per host, as illustrated in Figure 6.1. Here, we see how the device's connection to the VM totally bypasses the Dom0 host as well as the Xen VMM, and is managed by VT-d or IOMMU to access the PCI-Express bus which the GPUs utilize. This is in contrast to other common virtual device uses, where hardware is emulated by the host and shared across all guests. This is the common usage for Ethernet controllers and input devices to enable users to interact with VMs as they would with native hosts, unlike the bridged model shown in the figure. The potential downside of this method is there can be a 1:1 or 1:N mapping of VMs to GPUs only. A M:1 mapping where multiple VMs use a GPU is not possible. However, almost all scientific applications environments using GPUs generally do not share GPUs between processes or other nodes, as doing so would cause unpredictable and serious performance degradation. As such, this GPU isolation within a VM can be considered an advantage in many contexts.

4.4.1 Feature Comparison

Using the GPU PCI passthrough technique described previously has a number of advantages compared to front-end API implementations. First, it allows for an operating environment that more closely relates to native bare-metal usage of GPUs. Essentially, a VM provides a nearly identical infrastructure to clusters and supercomputers with integrated GPUs. This lowers the learning curve for many researchers, and even enables researchers to potentially use other tools within a VM that might not be supported within a supercomputer or cluster. Unlike with remote API implementations, users don't need to recompile or modify their code, as the GPUs are essentially local to the data. Further comparing to remote API implementations, using PCI passthrough within a VM allows users to leverage any GPU framework

available, OpenCL or CUDA, and any higher level programming frameworks such as within Matlab or Python.

Through the use of advanced scheduling techniques within cloud infrastructure, we can also take advantage of PCI passthrough implementation for efficiency purposes. Users could request VMs with GPUs which get scheduled for creation on machines that provide such resources, but can also request normal VMs as well. The scheduler can correctly map VM requirement requests to heterogeneous hardware. This enables large scale resources to support a wide array of scientific computing applications without the added cost of putting GPUs in all compute nodes.

4.5 Experimental Setup

In this chapter back-end GPU PCI passthrough to virtual machines using the Xen hypervisor is detailed, however proper evaluation of the performance of such method needs to be properly considered. As such, we ran an array of benchmarks that evaluate the performance of this method compared to the same hardware running native bare-metal GPU code without any virtualization. We focus our tests on single-node performance to best understand low level overhead.

To evaluate the effectiveness of GPU-enabled VMs within Xen, two different machines were used to represent two generations of Nvidia GPUs. The first system at Indiana University consists of dual-socket Intel Xeon X5660 6-core CPUs at 2.8Ghz with 192GB DDR3 RAM, 8TB RAID5 array, and two Nvidia Tesla "Fermi" C2075 GPUs. The system at USC/ISI uses a dual-socket Intel Xeon E5-2670 8-core CPUs at 2.6Ghz with 48GB DDR3 RAM, 3 600GB SAS disk drives, but with the latest Nvidia Tesla "Kepler" K20m GPU supplied by Nvidia. These machines represent the present Fermi series GPUs along with the recently release Kepler series GPUs, providing a well-rounded experimental environment. Native systems were installed

with standard Centos 6.4 with a stock 2.6.32-279 Linux kernel. Xen host systems were still loaded with Centos 6.4 but with a custom 3.4.53-8 Linux kernel and Xen 4.2.22. Guest VMs were also Centos 6.4 with N-1 processors and 24GB of memory and 1 GPU passed through in HVM full virtualization mode. Using both IU and USC/ISI machine configurations in native and VM modes represent the 4 test cases for our work.

In order to evaluate the performance, the SHOC Benchmark suite [179] was used to extensively evaluate performance across each test platform. The SHOC benchmarks were chosen because they provide a higher level of evaluation regarding GPU performance than the sample applications provided in the Nvidia SDK, and can also evaluate OpenCL performance in similar detail. The benchmarks were compiled using the NVCC compiler in CUDA5 driver and library, along with OpenMPI 1.4.2 and GCC 4.4. Each benchmark was run a total of 20 times, with the results given as an average of all runs.

4.6 Results

Results of all benchmarks are compressed into three subsections: floating point operations, device bandwidth and pci bus performance. Each represents a different level of evaluation for GPU-enabled VMs compared to bare-metal native GPU usage.

4.6.1 Floating Point Performance

Flops, or floating point operations per second, are a common measure of computational performance, especially within scientific computing. The Top 500 list [8] has been the relative gold standard for evaluating supercomputing performance for more than two decades. With the advent of GPUs in some of the fastest supercomputers today, including GPUs identical to those used in our experimentation, understanding

the performance relative to this metric is imperative.

Figure 4.2 shows the raw peak FLOPs available using each GPU in both native and virtualized modes. First, we observe the advantage of using the Kepler series GPUs over Fermi, with peak single precision speeds tripling in each case. Even for double precision FLOPs, there is roughly a doubling of performance with the new GPUs. However its most interesting that there is less than a 1% overhead when using GPU VMs compared to native case for pure floating point operations. The K20m-enabled VM was able to provide over 3 Teraflops, a notable computational feat for any single node.

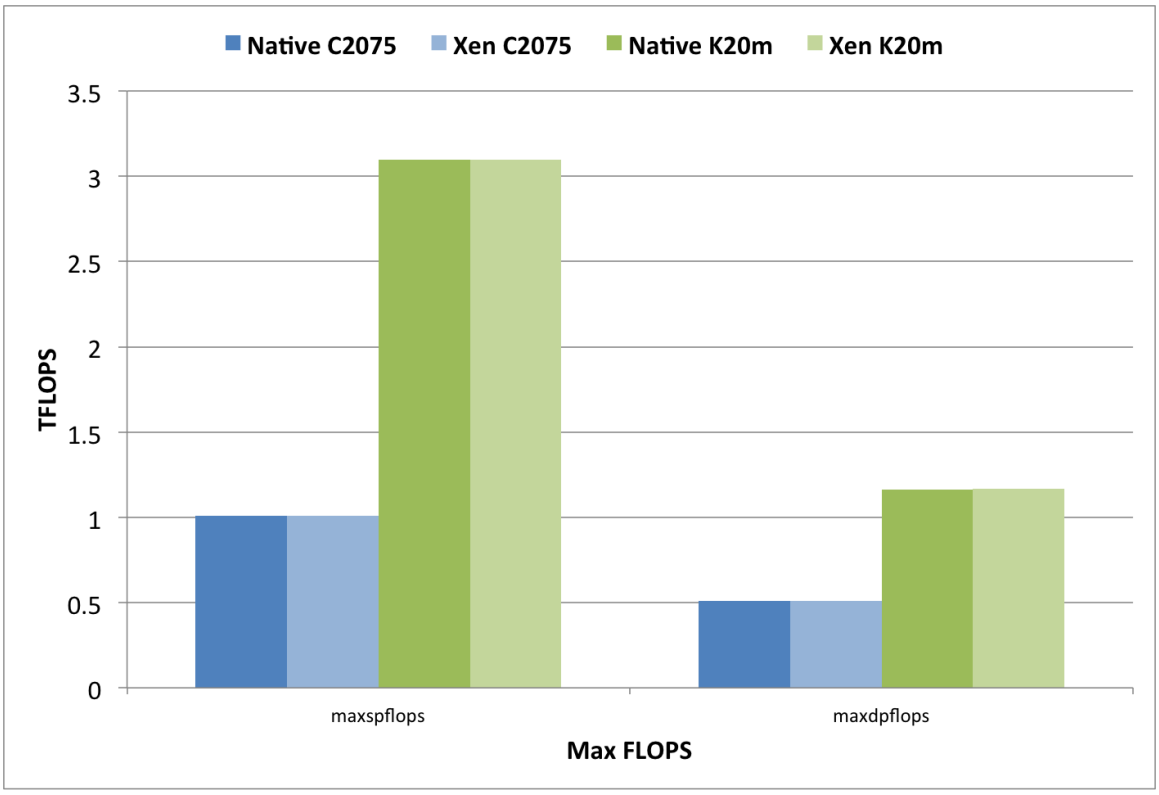


Figure 4.2 GPU Floating Point Operations per Second

Figures 4.3 and 4.4 show Fast Fourier Transform (FFT) and the Matrix Multiplication implementations across both test platforms. For all benchmarks that do not take into account the PCI-Express (pcie) bus transfer time, we again see near-native

performance using the Kepler GPUs and Fermi GPUs when compared to bare metal cases. Interestingly, overhead within Xen VMs is consistently less than 1%, confirming the synthetic MaxFlops benchmark above. However, we do see some performance impact when calculating the total FLOPs with the pcie bus in the equation. This performance decrease ranges significantly for the C2075-series GPU, roughly about a 15% impact for FFT and a 5% impact for Matrix Multiplication. This overhead in pcie runs is not as pronounced for the Kepler K20m test environment, with near-native performance in all cases (less than 1%).

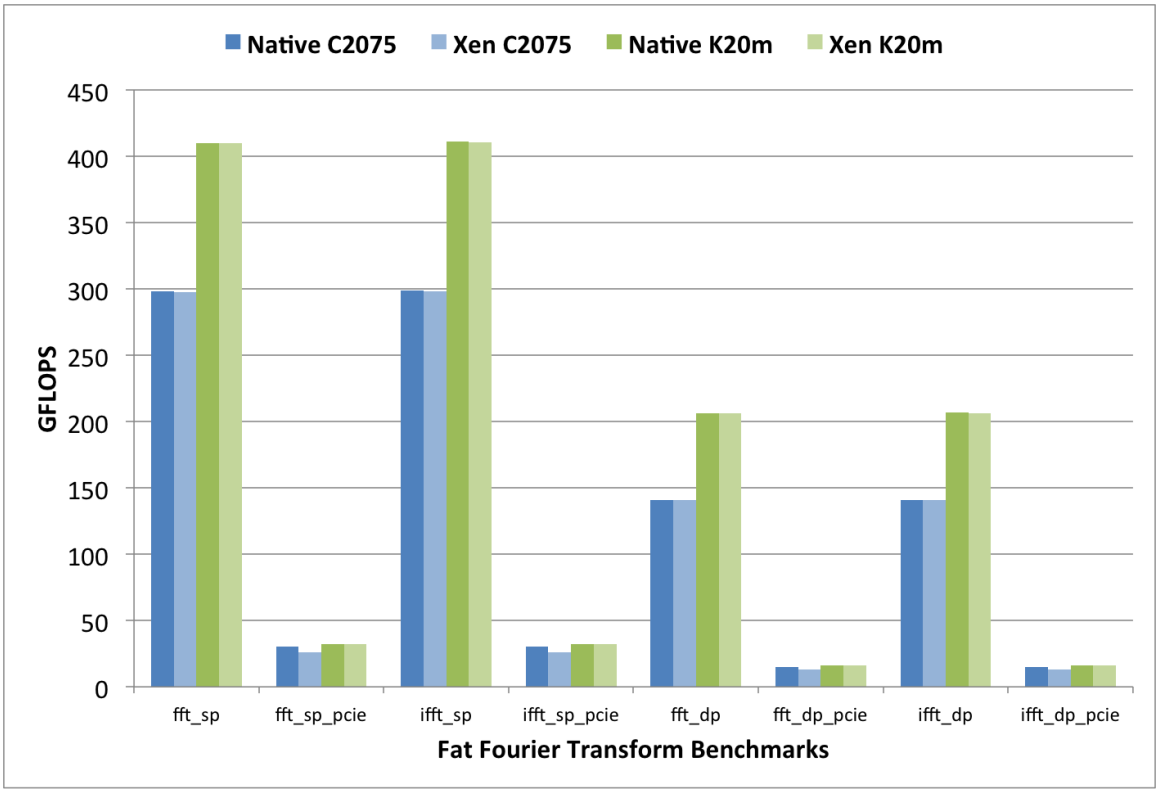


Figure 4.3 GPU Fast Fourier Transform

Other FLOP-based benchmarks are used to emulate higher level applications. Stencil represents a 9-point stencil operation applied to a 2D data set, and the S3D benchmark is a computationally-intensive kernel from the S3D turbulent combustion simulation program [180]. In Figure 4.5, we see that both the Fermi C2075 and Kepler

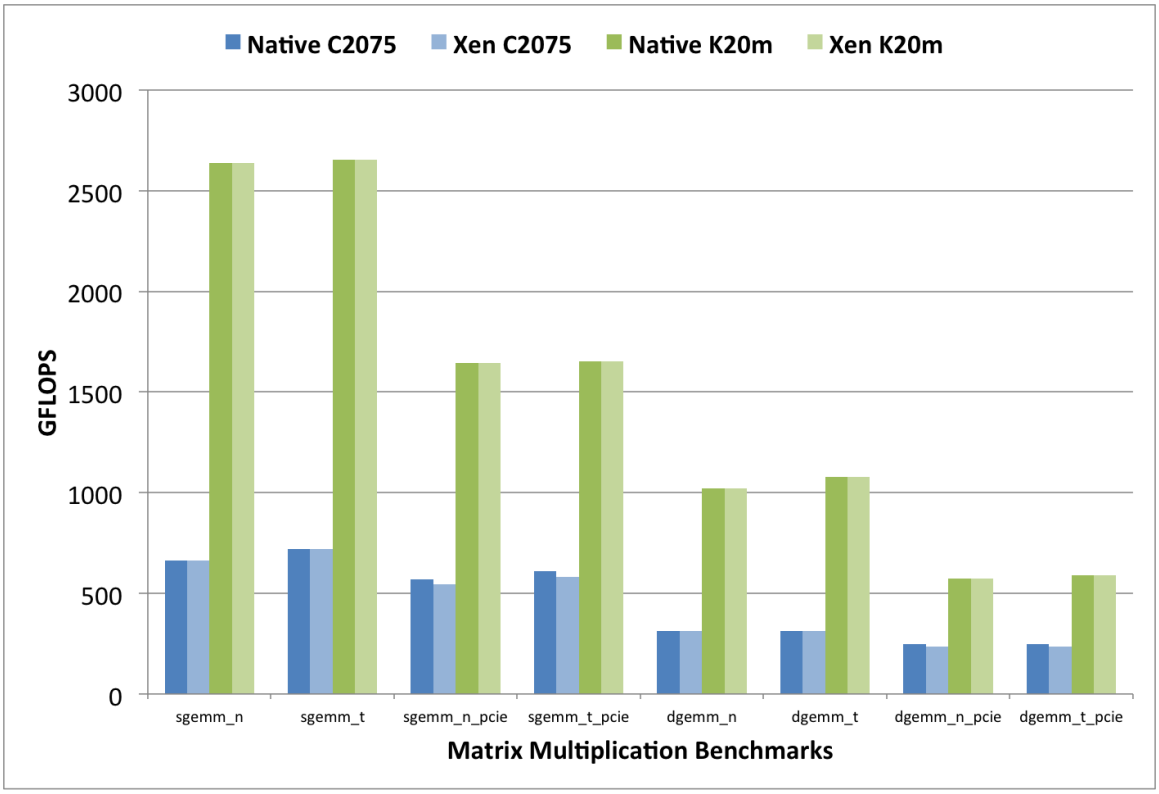


Figure 4.4 GPU Matrix Multiplication

K20m GPUs performing well compared to the native base case, showing the overhead of virtualization is low. The C2075-enabled VMs experience slightly more overhead when compared to native performance again for pcie runs, but overhead is at most 7% for the S3D benchmark.

4.6.2 Device Speed

While floating point operations allow for the proposed solution to relate to many traditional HPC applications, they are just one facet of GPU performance within scientific computing. Device speed, measured in both raw bandwidth and additional benchmarks, provides a different perspective towards evaluating GPU PCI passthrough in Xen. Figure 4.6 illustrates device level memory access of various GPU device memory structures. With both Nvidia GPUs, virtualization has little to no impact on the

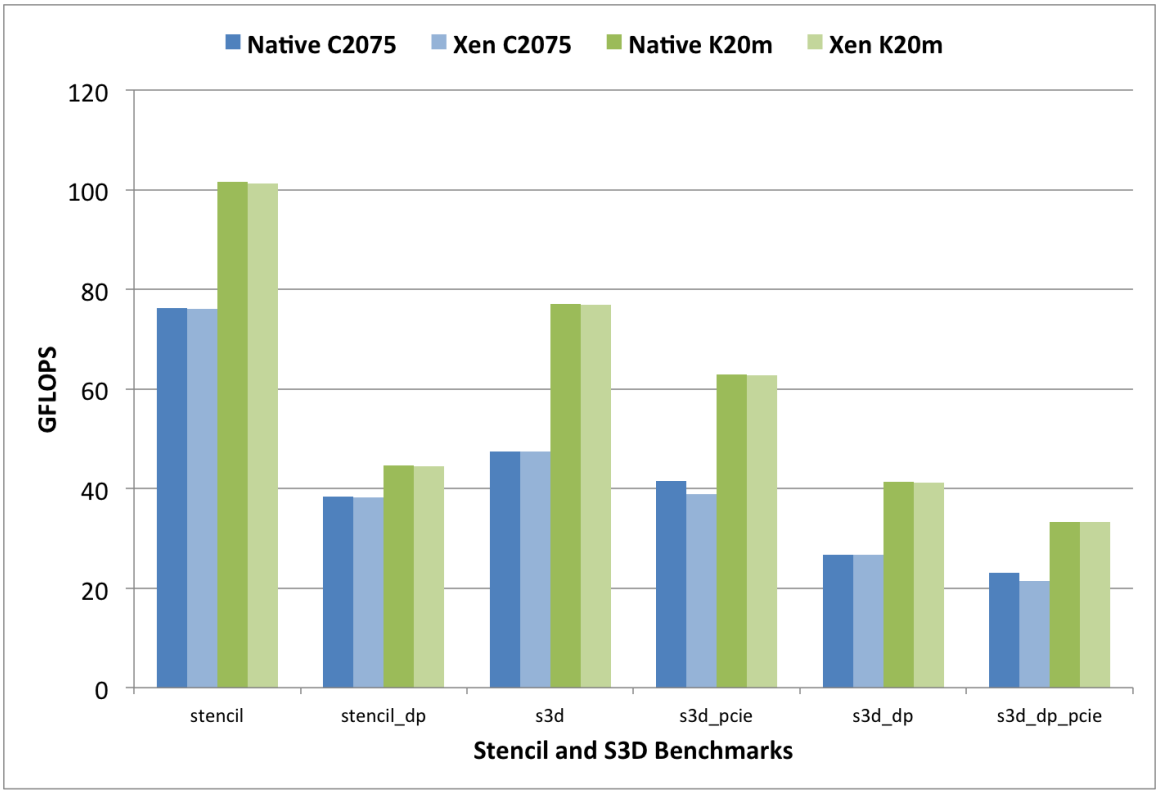


Figure 4.5 GPU Stencil and S3D

performance of inter-device memory bandwidth. As expected the Kepler K20m outperformed the C2075 VMs and there was a higher variance between runs with both native and VM cases. Molecular Dynamics and Reduction benchmarks in Figure 4.7 perform again at near-native performance without the pcie bus taken into account. However the overhead observed increases to 10-15% when the PCI-Express bus is considered when looking at the Fermi C2075 VMs.

4.6.3 PCI Express Bus

Upon evaluating PCI passthrough of GPUs, it is apparent that the PCI express bus is subject to the greatest potential for overhead, as was observed in the Fermi C2075 benchmarks. The VT-d and IOMMU chip instruction sets interface directly with the PCI bus to provide operational and security related mechanisms for each PCI device,

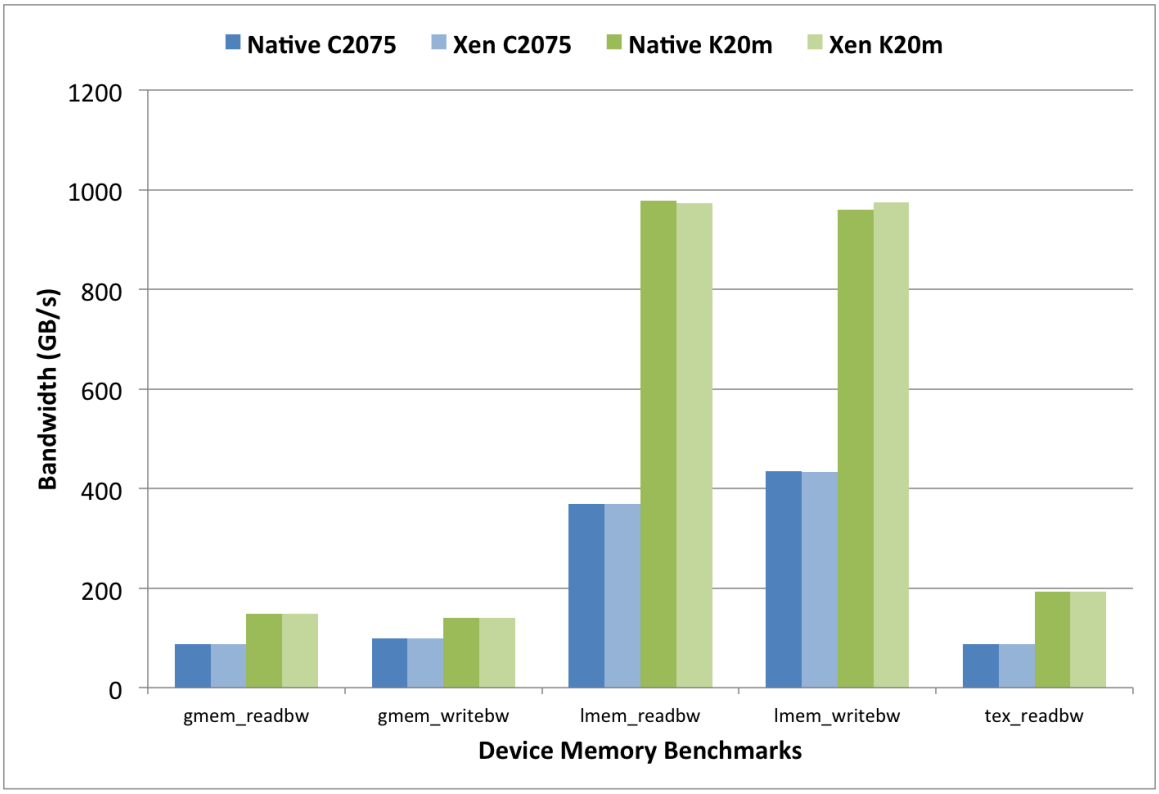


Figure 4.6 GPU Device Memory Bandwidth

thereby ensuring proper function in a multi-guest environment but potentially introducing some overhead. As such, it is imperative to investigate any and all overhead at the PCI Express bus.

Figure 4.8 looks at maximum PCI bus speeds for each experimental implementation. First, we see a consistent overhead in the Fermi C2075 VMs, with a 14.6% performance impact for download (to-device) and a 26.7% impact in readback (from-device). However, the Kepler K20 VMs do not experience the same overhead in the PCI-Express bus, and instead perform at near-native performance. These results indicate that the overhead is minimal in the actual VT-d mechanisms, and instead due to other factors.

Upon further examination, we have identified the performance degradation within the Fermi-based VM experimental setup is due to the testing environment's NUMA

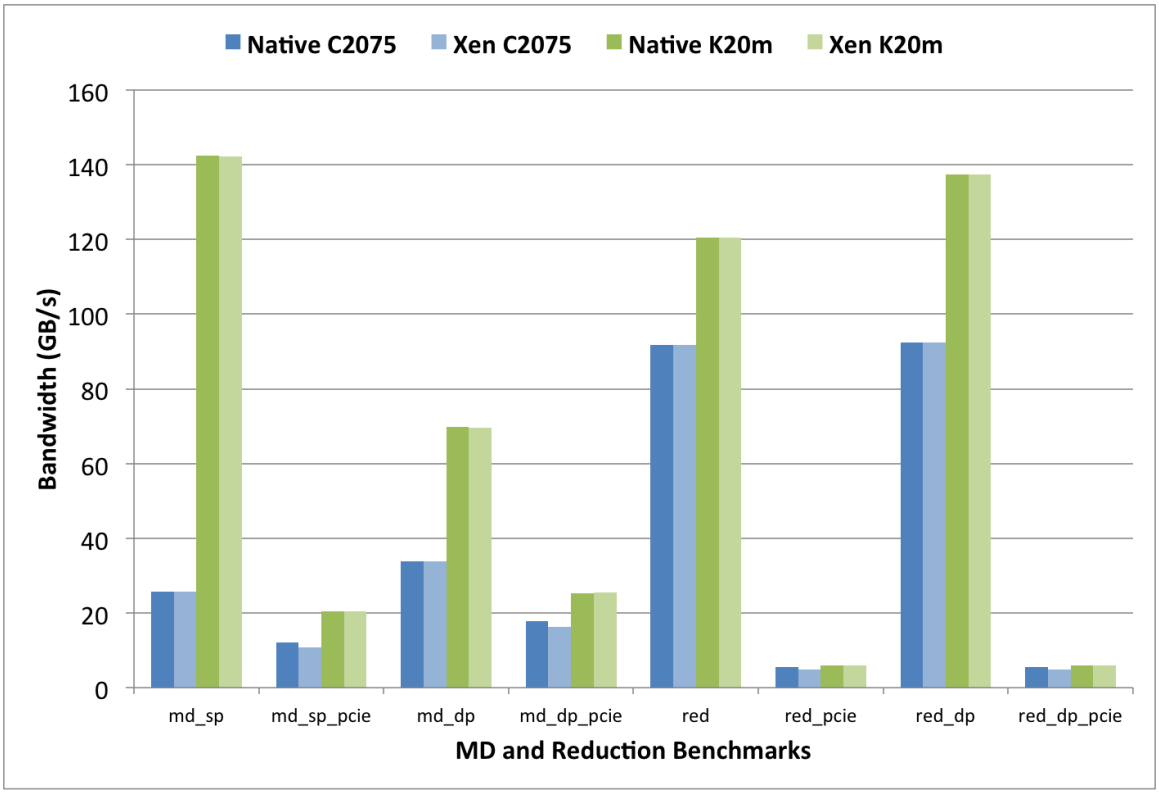


Figure 4.7 GPU Molecular Dynamics and Reduction

node configuration with the Westmere-based CPUs. With the Fermi nodes, the GPUs are connected through the PCI-Express bus from CPU Socket #1, yet the experimental GPU VM was run using CPU Socket #0. This meant that the VM's PCI-Express communication involved more host involvement with Socket #1, leading to an notable decrease in read and write speeds across the PCI-Express Bus. Such architectural limitations seem to be relieved in the next generation with a Sandy-Bridge CPU architecture and the Kepler K20m experimental setup. In summary, GPU PCI-Express performance in a VM is sensitive to the host CPU's NUMA architecture and care is needed to mitigate the impact, either by leveraging new architectures or by proper usage of Xen's VM core assignment features. Furthermore, the overhead in this system diminishes significantly when using the new Kepler GPUs by Nvidia.

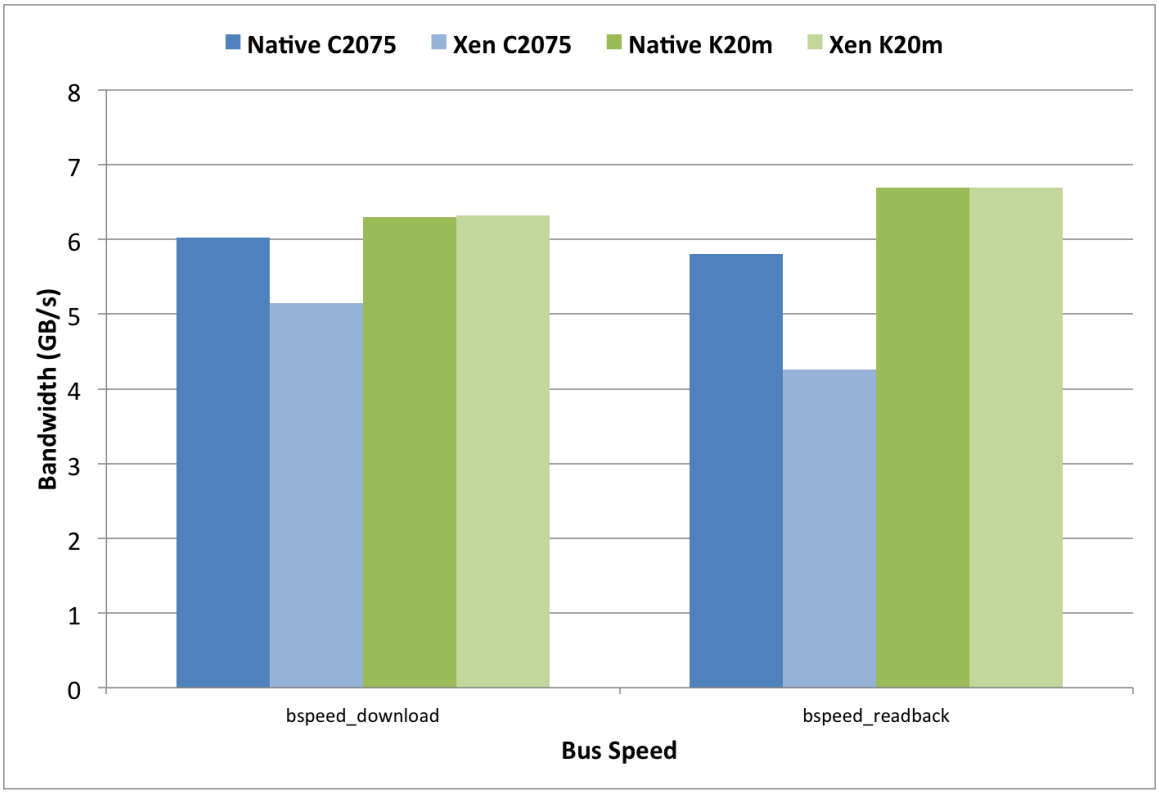


Figure 4.8 GPU PCI Express Bus Speed

4.7 Discussion

This chapter evaluates the use of general purpose GPUs within cloud computing infrastructure, primarily targeted towards advanced scientific computing. The method of PCI passthrough of GPUs directly to a guest virtual machine running on a tuned Xen hypervisor shows initial promise for an ubiquitous solution in cloud infrastructure. In evaluating the results in the previous section, a number of points become clear.

First, we can see that there is a small overhead in using PCI passthrough of GPUs within VMs, compared to native bare-metal usage, which represents the best possible use case. As with all abstraction layers, some overhead is usually inevitable as a necessary trade-off to added feature sets and improved usability. The same is true

for GPUs within Xen VMs. The Kepler K20m GPU-enabled VMs operated at near-native performance for all runs, with a 1.2% reduction at worst in performance. The Fermi based C2075 VMs experience more overhead due to the NUMA configuration that impacted PCI-Express performance. However, when the overhead of the PCI-Express bus is not considered, the C2075 VMs perform at near-native speeds.

GPU PCI passthrough also has the potential to perform better than other front-end API solutions that are applicable within VMs, such as gVirtus or rCUDA. This is because such solutions are designed to communicate via a network interconnect such as 10Gb Ethernet or QDR InfiniBand [181], which introduces an inherent bottleneck. Even with the theoretically optimal configuration of rCUDA using QDR InfiniBand, the maximum theoretical bus speed is 40Gbs, which is comparably less than the measured 54.4Gps real-world performance measured between host-to-device transfers with GPU-enabled Kepler VMs.

Overall, it is our hypothesis that the overhead in using GPUs within Xen VMs as described in this chapter will largely go unnoticed by most mid-level scientific computing applications. This is especially true when using the latest Sandy-Bridge CPUs with the Kepler series GPUs. We expect many mid-tier scientific computing groups to benefit the most from the ability to use GPUs in a scientific cloud infrastructure. Already this has been confirmed in [182], where similar a methodology has been leveraged specifically for Bioinformatics applications in the cloud.

4.8 Chapter Summary and Future Work

The ability to use GPUs within virtual machines represents a leap forward for supporting advanced scientific computing within cloud infrastructure. The method of direct PCI passthrough of Nvidia GPUs using the Xen hypervisor offers a clean, reproducible solution that can be implemented within many Infrastructure-as-a-Service

(IaaS) deployments. Performance measurements indicate that the overhead of providing a GPU within Xen is minimal compared to the best-case native use, however NUMA inconsistencies can impact performance. The New Kepler-based GPUs operate with a much lower overhead, making those GPUs an ideal choice when designing a new GPU IaaS system.

Next steps for this work could involve providing GPU-based PCI passthrough within the OpenStack nova IaaS framework. This will enable research laboratories and institutions to create new private or national-scale cloud infrastructure that have the ability to support new scientific computing challenges. Other hypervisors could also leverage GPU PCI passthrough techniques and warrant in-depth evaluation in the future. Furthermore, we hope to integrate this work with advanced interconnects and other heterogeneous hardware and provide a parallel high performance cloud infrastructure to enable mid-tier scientific computing.

Chapter 5

GPU-Passthrough Performance: A Comparison of KVM, Xen, and LXC for CUDA and OpenCL Applications

5.1 Abstract

As more scientific workloads are moved into the cloud, the need for high performance accelerators increases. Accelerators such as GPUs offer improvements in both performance and power efficiency over traditional multi-core processors; however, their use in the cloud has been limited. Today, several common hypervisors support GPU-passthrough, but their performance has not been systematically characterized.

In this chapter we show that low overhead PCI passthrough is achievable across 3 major hypervisors and two processor microarchitectures. We compare the performance of two generations of NVIDIA GPUs within the Xen and KVM hypervisors, and we also compare the performance to that of Linux Containers (LXC). We show that GPU passthrough to KVM achieves 98–100% of the base system's performance across two architectures, while Xen achieve 96% of the base systems performance. In addition, we describe several valuable lessons learned through our analysis and share the advantages and disadvantages of each hypervisor/PCI passthrough solution.

5.2 Introduction

As scientific workloads continue to demand increasing performance at greater power efficiency, high performance architectures have been driven towards heterogeneity and specialization. Intel's Xeon Phi, and GPUs from both NVIDIA and AMD represent some of the most common accelerators, with each capable of delivering improved performance and power efficiency over commodity multi-core CPUs.

Infrastructure-as-a-Service (IaaS) clouds have the potential to democratize access to the latest, fastest, and most powerful computational accelerators. This is true of both public and private clouds. Yet today's clouds are typically homogeneous without access to even the most commonly used accelerators. Historically, enabling virtual machine access to GPUs and other PCIe devices has proven complex and error-prone, with only a small subset of GPUs being certified for use within a few commercial hypervisors. This is especially true for NVIDIA GPUs, likely the most popular for scientific computing, but whose drivers have always been closed source.

Given the complexity surrounding the choice of GPUs, host systems, and hypervisors, it is perhaps no surprise that Amazon is the only major cloud provider offering customers access to GPU-enabled instances. All of this is starting to change, however, as open source and other freely available hypervisors now provide sufficiently robust PCI passthrough functionality to enable GPU and other accelerator access whether in the public or private cloud.

Today, it is possible to access GPUs at high performance within all of the major hypervisors, merging many of the advantages of cloud computing (e.g. custom images, software defined networking, etc.) with the accessibility of on-demand accelerator hardware. Yet, no study to date has systematically compared the performance of PCI passthrough across all major cloud hypervisors. Instead, alternative solutions have

been proposed that attempt to virtualize the GPU [183] , but sacrifice performance.

In this chapter, we characterize the performance of both NVIDIA Fermi and Kepler GPUs operating in PCI passthrough mode in Linux KVM, Xen, and Linux Containers (LXC). Through a series of microbenchmarks as well as scientific and Big Data applications, we make two contributions:

1. We demonstrate that PCI passthrough at high performance is possible for GPUs across 3 major hypervisors.

2. We describe the lessons learned through our performance analysis, as well as the relative advantages and disadvantages of each hypervisor for GPU support.

5.3 Related Work & Background

GPU virtualization and GPU-passthrough are used within a variety of contexts, from high performance computing to virtual desktop infrastructure. Accessing one or more GPUs within a virtual machine is typically accomplished by one of two strategies: 1) via API remoting with device emulation; or 2) using PCI passthrough.

5.3.1 GPU API Remoting

rCUDA, vCUDA, GViM, and gVirtuS are well-known API remoting solutionsFundamentally, these approaches operate similarly by splitting the driver into a front-end/back-end model, where calls into the interposed CUDA library (front-end) are sent via shared memory or a network interface to the back-end service that executes the CUDA call on behalf of the virtual machine. Notably, this technique is not limited to CUDA, but can be used to decouple OpenCL, OpenGL, and other APIs from their local GPU or accelerator.

The performance of API-remoting depends largely on the application and the remoting solution's implentation. Bandwidth and latency-sensitive benchmarks and applications will tend to expose performance bottlenecks more than compute-intensive applications. Moreover, solutions that rely on high speed networks, such as Infiniband, will compete with application-level networking for bandwidth.

5.3.2 PCI Passthrough

Input/Output Memory Management Units, or IOMMUs, play a fundamental roll in the PCI-passthrough virtualization mechanism. Like traditional MMUs that provide a virtual memory address space to CPUs [184], an IOMMU serves the fundamental purpose of connecting a direct memory access (DMA) capable I/O bus to main memory. The IOMMU unit, typically within the chipset, maps device virtual addresses to physical memory addresses. This process also has the added improvement of guaranteeing device isolation by blocking rogue DMA and interrupt requests [185], with a slight overhead, especially in early implementations [186].

Currently two major IOMMU implementations exist, VT-d and AMD-Vi by Intel and AMD, respectively. Both specifications provide DMA remapping to enable PCI-passthrough as well as other features such as interrupt remapping, hypervisor snooping, and security control mechanisms to ensure proper and efficient hardware utilization. PCI passthrough has been studied within the context of networking [187], storage [188], and other PCI-attached devices; however, GPUs have historically lagged behind other devices in their support for virtual machine passthrough.

5.3.3 GPU Passthrough, a Special Case of PCI Passthrough

While generic PCI passthrough can be used with IOMMU technologies to pass through many PCI-Express devices, GPUs represent a special case of PCI devices, and a spe-

cial case of PCI passthrough. In traditional usage, GPUs usually serve as VGA devices primarily to render screen output, and while the GPUs used in this study do not render screen out, the function still exists in legacy. In GPU-passthrough, another VGA device (such as onboard graphics built into the motherboard, or a baseboard management controller) is necessary to serve as the primary display for the host, as well as providing emulated VGA devices for each guest VM. Most GPUs also have a video BIOS that requires full initialization and reset functions, which is often difficult due to the proprietary nature of the cards and their drivers.

Nevertheless, for applications that require native or near-native GPU performance across the full spectrum of applications with immediate access to the latest GPU drivers and compilers, GPU passthrough solutions are preferrable to API remoting. Today, Citrix Xenserver, open source Xen [189], and VMWare ESXi [190], and most recently KVM all support GPU passthrough. To our knowledge, no one has systematically characterized the performance of GPU passthrough across a range of hypervisors, across such a breadth of benchmarks, and across multiple GPU generations as we do.

5.4 Experimental Methodology

5.4.1 Host and Hypervisor Configuration

We used two hardware systems, named Bespin and Delta, to evaluate hypervisors. The Bespin system at USC/ISI represents Intel's Sandy Bridge microarchitecture with a Kepler class K20 GPU. The Delta system, provided by the FutureGrid project [191], represents the Westmere microarchitecture with a Fermi class C2075 GPU. Table 5.1 provides the major hardware characteristics of both systems. Note that in addition, both systems include 10 gigabit Ethernet, gigabit Ethernet, and either FDR or QDR

Table 5.1 Host hardware configurations

	Bespin	Delta
CPU (cores)	2x E5-2670 (16)	2x X5660 (12)
Clock Speed	2.6 GHz	2.6 GHz
RAM	48 GB	192 GB
NUMA Nodes	2	2
GPU	1x K20m	2x C2075

Infiniband. Our experiments do not emphasize networking, and we use the gigabit ethernet network for management only.

A major design goal of these experiments was to reduce or eliminate NUMA effects (non-uniform memory access) on the PCI passthrough results in order to facilitate fair comparisons across hypervisors and to reduce experimental noise. To this end, we configured our virtual machines and containers to execute only on the NUMA node containing the GPU under test. We acknowledge that the NUMA effects on virtualization may be interesting in their own right, but they are not the subject of this set of experiments.

We use Bespin and Delta to evaluate three hypervisors and one container-based approach to GPU passthrough. The hypervisors and container system, Xen, KVM, and LXC, are summarized in Table 5.2. Note that each virtualization solution imposes its own unique requirements on the base operating system. Hence, Xen's Linux kernel refers to the Domain-0 kernel, whereas KVM's Linux kernel represents the actual running kernel hosting the KVM hypervisor. Linux Containers share a single kernel between the host and guests.

It is worth noting that these experiments were originally set up, configured and

Table 5.2 Host Hypervisor/Container Configuration

Hypervisor	Linux Kernel	Linux Version
KVM	3.12	Arch 2013.10.01
Xen 4.3.0-7	3.12	Arch 2013.10.01
LXC	2.6.32-358.23.2	CentOS 6.4

tested with the VMWare ESXi hypervisor as well. However, we've found that fundamental limitations with the reliability and accuracy of the VMWare virtual Time Stamp Counter (TSC) [192] can lead to significant deviations in performance reporting. Not feeling comfortable with VMWare's ability to accurately report fair timing measurements, we decided to omit the VMWare results from this manuscript. For more information on time keeping in virtual machines, please refer to Section 7.3 for more details.

Similarly, hypervisor requirements prevented us from standardizing on a single host operating system. For Xen and KVM, we relied on the Arch Linux 2013.10.01 distribution because it provides easy access to the mainline Linux kernel. For our LXC tests, we use CentOS 6.4 because its shared kernel was identical to the base CentOS 6.4 kernel used in our testing. All of this makes comparison challenging, but as we describe in Section 5.4.2, we are running a common virtual machine across all experiments.

5.4.2 Guest Configuration

We treat each hypervisor as its own system, and compare virtual machine guests to a base CentOS 6.4 system. The base system and the guests are all composed of CentOS 6.4 installation with a 2.6.32-358.23.2 stock kernel and CUDA version 5.5.

Each guest is allocated 20 GB of RAM and a full CPU socket (either 6 or 8 CPU cores). Bespin experiments received 8 cores and Delta experiments received 6 cores. VMs were restricted to a single NUMA node. On the Bespin system, the K20m GPU was attached to NUMA node 0. On the Delta system, the C2075 GPU was attached to NUMA node 1. Hence VMs ran on NUMA node 0 for the Bespin experiments, and node 1 for the Delta experiments.

5.4.3 Microbenchmarks

Our experiments are composed of a mix of microbenchmarks and application-level benchmarks, as well as a combination of CUDA and OpenCL benchmarks. The SHOC benchmark suite provides a series of microbenchmarks in both OpenCL and CUDA [193]. For this analysis, we focus on the OpenCL benchmarks in order to exercise multiple programming models. Benchmarks range from low-level peak Flops and bandwidth measurements, to kernels and mini-applications.

5.4.4 Application Benchmarks

For our application benchmarks, we have chosen the LAMMPS molecular dynamics simulator [194], the GPU-LIBSVM [195], and the LULESH shock hydrodynamics simulator [196]. These represent a range of computational characteristics, from computational physics to big data analytics, and are representative of GPU-accelerated applications in common use.

LAMMPS The Large-scale Atomic/Molecular Parallel Simulator (LAMMPS), is a parallel molecular dynamics simulator [194, 197] used for production MD simulation on both CPUs and GPUs [198]. LAMMPS has two packages for GPU support, the USER-CUDA and GPU packages. With the USER-CUDA package, each GPU is used

by a single CPU, whereas the GPU package allows multiple CPUs to take advantage of a single GPU. There are performance trade-offs with both approaches, but we chose to use the GPU package in order to stress the virtual machine by exercising multiple CPUs. Consistent with the existing GPU benchmarking approaches, our results are based on the Rhodopsin protein.

GPU-LIBSVM LIBSVM is a popular implementation [199] of the machine learning classification algorithm support vector machine (SVM). GPU-accelerated LIBSVM [195] enhances LIBSVM by providing GPU-implementations of the kernel matrix computation portion of the SVM algorithm for radial basis kernels. For benchmarking purposes we use the NIPS 2003 feature extraction gisette data set. This data set has a high dimensional feature space and large number of training instances, and these qualities are known to be computational intensive to generate SVM models. The GPU-accelerated SVM implementation shows dramatic improvement over the CPU-only implementation.

LULESH Hydrodynamics is widely used to model continuum properties and interactions in materials when there is an applied force [200]. Hydrodynamics applications consume approximately one third of the runtime of data center resource throughout the U.S. DoD (Department of Defense). The Livermore Unstructured Lagrange Explicit Shock Hydro (LULESH) was developed by Lawrence Livermore National Lab as one of five challenge problems in the DARPA UHPC program. LULESH is widely used as a proxy application in the U.S. DOE (Department of Energy) co-design effort for exascale applications [196].

Table 5.3 SHOC overheads for Bespin (K20) expressed as geometric means of scaled values within a level, while maximum overheads are expressed as a percentage.

	Bespin (K20)					
	KVM		Xen		LXC	
	Mean	Max	Mean	Max	Mean	Max
L0	0.999	1.57	0.997	3.34	1.00	1.77
L1	0.998	1.23	0.998	1.39	1.00	1.47
L2	0.998	0.48	0.995	0.846	0.999	1.90
	Bespin PCIe-only					
	KVM		Xen		LXC	
	Mean	Max	Mean	Max	Mean	Max
L0	0.997	0.317	0.999	0.143	0.995	0.981
L1	0.998	0.683	0.997	1.39	1.00	0.928
L2	0.998	0.478	0.996	0.846	1.00	0.247

Table 5.4 SHOC overheads for Delta (c2075) expressed as geometric means of scaled values within a level, while maximum overheads are expressed as a percentage.

	Delta (C2075)					
	KVM		Xen		LXC	
	Mean	Max	Mean	Max	Mean	Max
L0	1.01	0.031	0.969	12.7	1.00	0.073
L1	1.00	1.45	0.959	24.0	1.00	0.663
L2	1.00	0.101	0.982	4.60	1.00	0.016
	Delta PCIe-only					
	KVM		Xen		LXC	
	Mean	Max	Mean	Max	Mean	Max
L0	1.04	0.029	0.889	12.7	1.00	0.01
L1	1.00	1.45	0.914	20.5	0.999	0.380
L2	1.00	0.075	0.918	4.60	1.00	N/A

5.5 Performance Results

We characterize GPGPU performance within virtual machines across two hardware systems, 4 hypervisors, and 3 application sets. We begin with the SHOC benchmark suite before describing the GPU-LIBSVM, LAMMPS, and LULESH results. All benchmarks are run 20 times and averaged. Results are scaled with respect to a base CentOS 6.4 system for both systems. That is, we compare virtualized Bespin performance to non-virtualized Bespin performance, and virtualized Delta performance to non-virtualized Delta performance. Values less than 1 indicate that the base system outperformed the virtual machine, while values greater than 1 indicate that the virtual machine outperformed the base system. In cases where we present geometric means across multiple benchmarks, the means are taken over these scaled values, and the semantics are the same: less than 1 indicates overhead in the hypervisor, greater than 1 indicates a performance increase over the base system.

5.5.1 SHOC Benchmark Performance

SHOC splits its benchmarks into 3 Levels, named Levels 0 through 2. Level 0 represents device-level characteristics, peak Flop/s, bandwidth, etc. Level 1 characterizes computational kernels: FFT and matrix multiplication, among others. Finally, Level 2 includes "mini-applications," in this case an implementation of the S3D, a computational chemistry application.

Because the SHOC OpenCL benchmarks report more than 70 individual microbenchmarks, space does not allow us to show each benchmark individually. Instead, we start with a broad overview of SHOC's performance across all benchmarks, hypervisors, and systems. We then discuss in more detail those benchmarks that either outperformed or underperformed the Bespin (K20) system by 0.50% or more.

We call these benchmarks outliers. As we will show, those outlier benchmarks identified on the Bespin system, also tend to exhibit comparable characteristics on the Delta system as well, but the overhead is typically higher.

In Tables 5.3 and 5.4, we provide geometric means for each SHOC level across each hypervisor and system. We also include the maximum overhead for each hypervisor at each level to facilitate comparison across hypervisors and systems. Finally, we provide a comparable breakdown of only the PCIe SHOC benchmarks, based on the intuition that PCIe-specific benchmarks will likely result in higher overhead.

At a high level, we immediately notice that in the cases of KVM and LXC, both perform very near native across both the Bespin and Delta platforms. On average, these systems are almost indistinguishable from their non-virtualized base systems. So much so, that experimental noise occasionally boosts performance slightly above their base systems.

This is in sharp contrast to the Xen , which performs well on the Bespin system, but poorly on the Delta system in some cases. This is particularly evident when looking at the maximum overheads for Xen across both systems. In this case, we see that on the Bespin system, Xen's maximum overhead of 3.34% is dwarfed by Delta's maximum Xen overhead of 24.0%. We provide a more in-depth discussion of these overheads below.

Of the Level 0 benchmarks, only four exhibited notable overhead in the case of the Bespin system: bspeed_download, lmem_readbw, tex_readbw, and ocl_queue. These are shown in Figure 5.1. These benchmarks represent device-level characteristics, including host-to-device bandwidth, onboard memory reading, and OpenCL kernel queue delay. Of the four, only bspeed_download incurs a statistically significant overhead. The remainder perform within one standard deviation of the base, despite an overhead of greater than 0.5%.

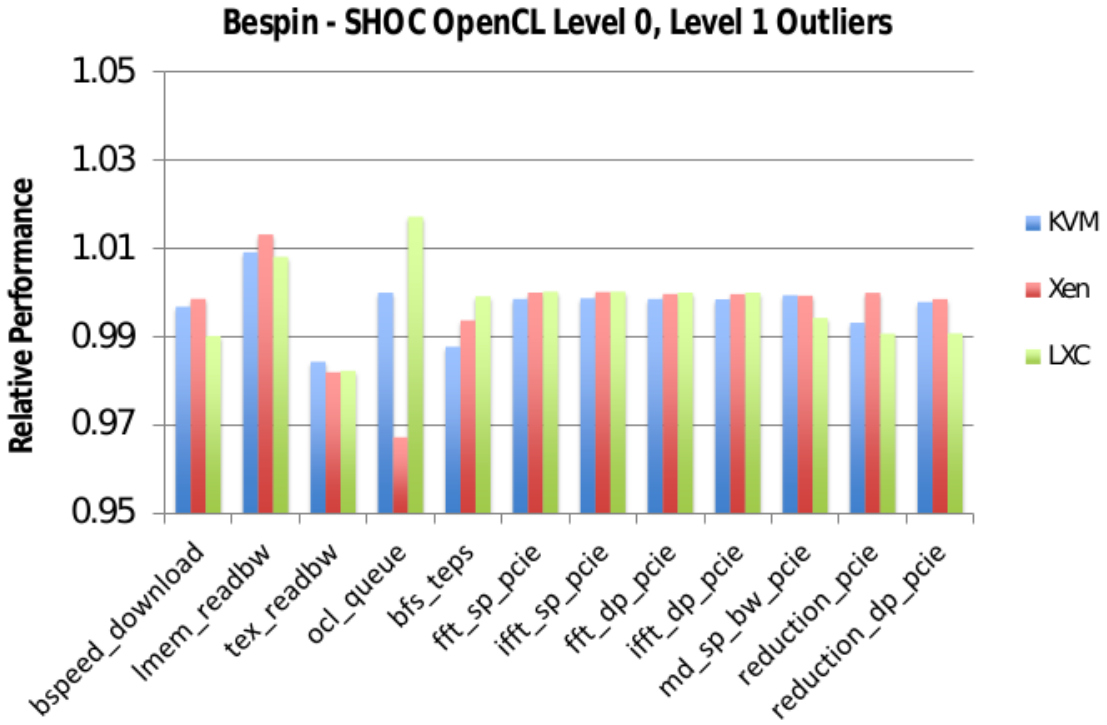


Figure 5.1 SHOC Levels 0 and 1 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

bspeed_download is representative of the most likely source of virtualization overhead, data movement across the PCI-Express bus. The PCIe lies at the boundary of the virtual machine and the physical GPU, and requires interrupt remapping, IOMMU interaction, etc. in order to enable GPU passthrough into the virtual machine. Despite this, in Figure 5.1 we see a maximum of less than 0.5% overhead for bspeed_download for both KVM and Xen.

The remainder of Figure 5.1 includes a series of SHOC's Level 1 benchmarks, representing computational kernels. This includes BFS, FFT, molecular dynamics, and reduction kernels. Notably, nearly all of the benchmarks exhibiting overhead are the PCIe portion of SHOC's benchmarks. This is unsurprising, since the Level 0 benchmarks suggest PCIe bandwidth as the major source overhead. Still, results remain

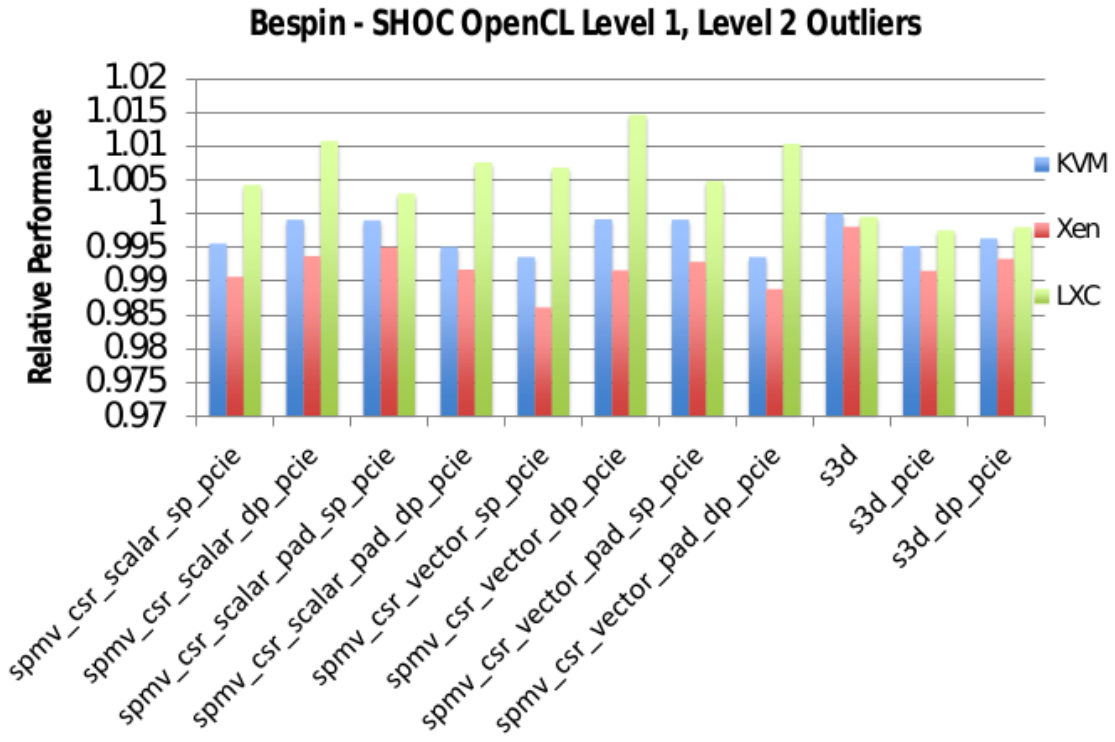


Figure 5.2 SHOC Levels 1 and 2 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

consistent with the bspeed_download overhead observed in the Level 0 benchmarks, further suggesting that host/device data movement is the major source of overhead. In Figure 5.2 we present the remaining SHOC Level 1 results as well as the SHOC Level 2 results (S3D).

Turning to the Delta system, in Figures 5.3 and 5.4, we show the same benchmarks for the Delta system as was shown in Figures 5.1 and 5.2. Again, we find that the same benchmarks are responsible for most of the overhead on the Delta system. This is unsurprising, since PCIe was shown to be the source of the bulk of the overhead. A major difference in the case of the Delta system, however, is the amount of overhead. While the Bespin system saw overheads of approximately 1%, Delta's overhead routinely jumps higher with Xen.

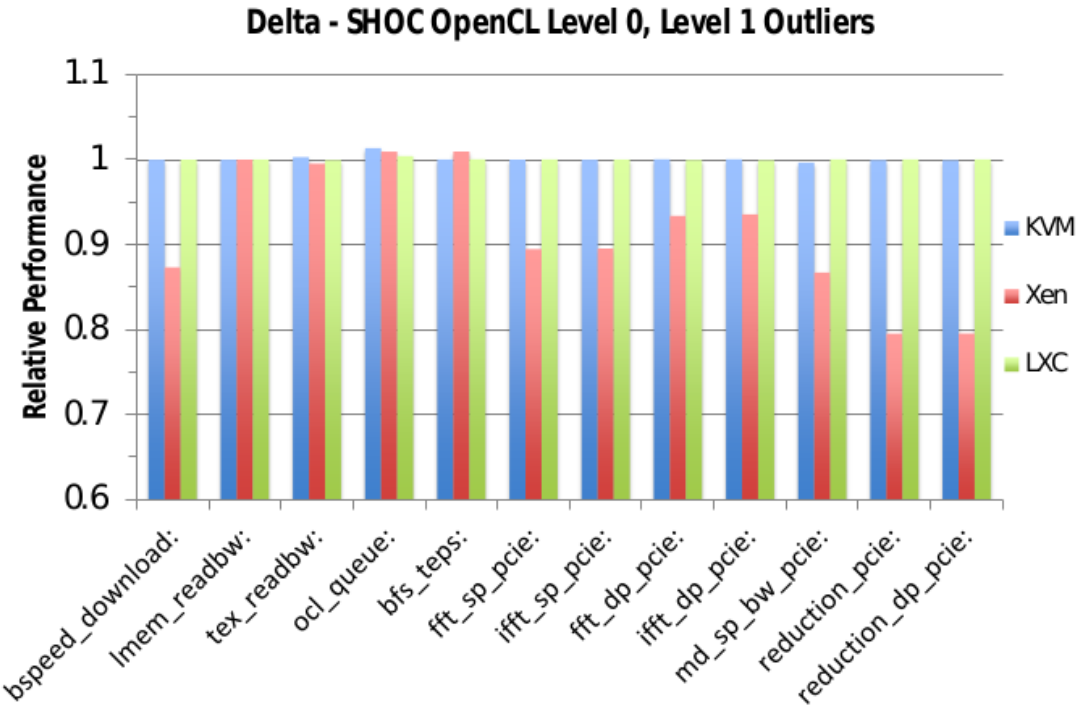


Figure 5.3 SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.

On further examination, we determined that Xen was unable to activate IOMMU large page tables on the Delta system. KVM successfully allocated 4k, 2M, and 1G page table sizes, while Xen was limited to size 4k page tables. The Bespin system was able to take advantage of 4k, 2M, and 1G page sizes on both KVM and Xen. It appears that this issue is correctable and does not represent a fundamental limitation to the Xen hypervisor on the Nehalem/Westmere microarchitecture.

In light of this, we broadly find that virtualization overhead across hypervisors and architectures is minimal. Questions remain as to the source of the exceptionally high overhead in the case of Xen on the Delta system, but because KVM shows no evidence of this overhead, we believe the Westmere/Fermi architecture to be suitable for VGA passthrough in a cloud environment. In the case of the Bespin system, it

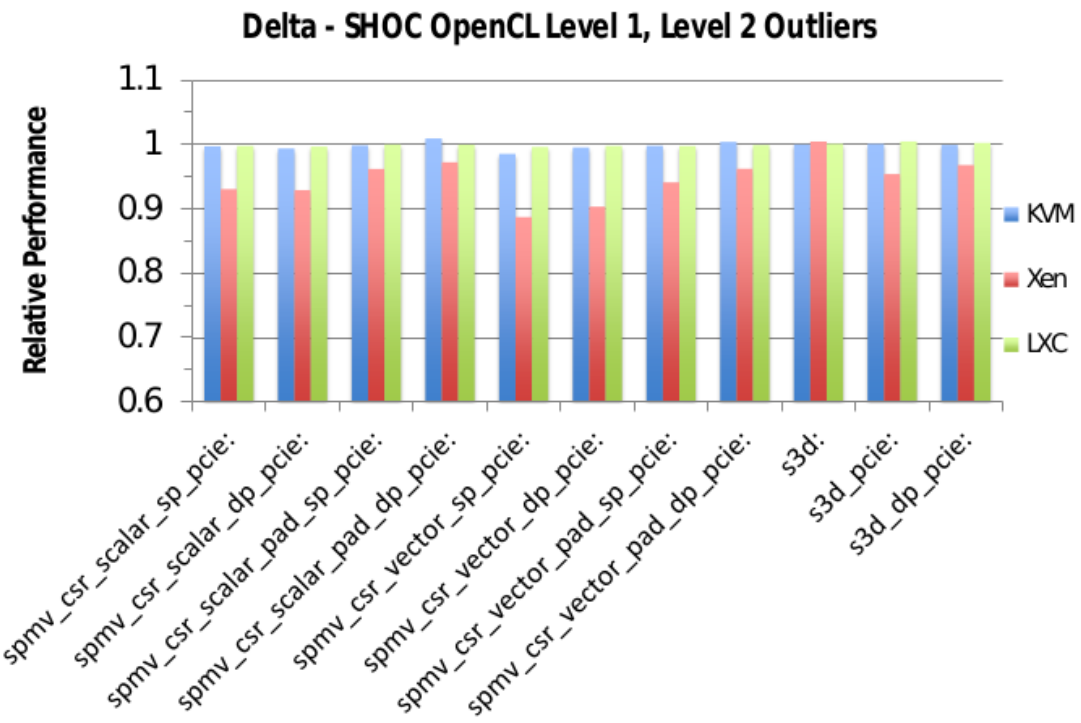


Figure 5.4 SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.

is clear that VGA passthrough can be achieved across hypervisors with virtually no overhead.

A surprising finding is that LXC showed little performance advantage over KVM, Xen, and VWMare. While we expected near-native performance from LXC we did not expect the hardware-assisted hypervisors to achieve such high performance. Still, LXC carries some advantages. In general, its maximum overheads are comparable to or less than KVM or Xen, especially in the case of the Delta system.

5.5.2 GPU-LIBSVM Performance

In Figures 5.5 and 5.6, we display our GPU-LIBSVM performance results for the Bespin (K20m) and Delta (C2075) systems. Using the NIPS 2003 gisette data set,

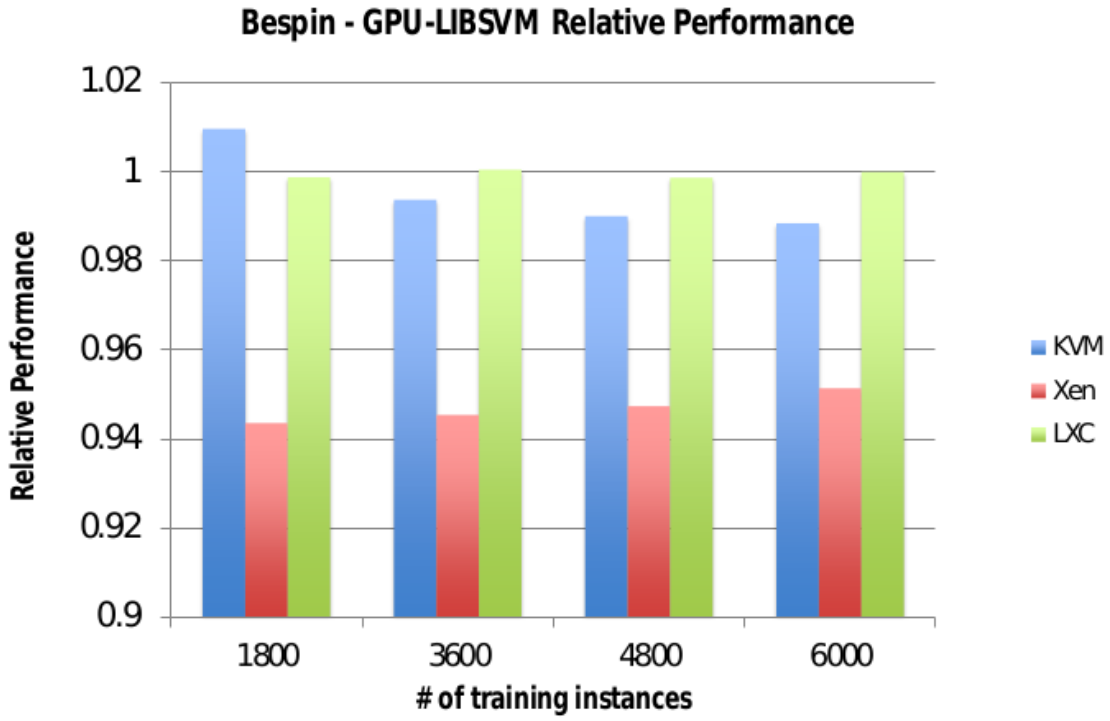


Figure 5.5 GPU-LIBSVM relative performance on Bespin system. Higher is better.

we show the performance across 4 problems sizes, ranging from 1800 to 6000 training instances. The NIPS 2003 gisette dataset is a standard dataset that was used as a part of the NIPS 2003 feature selection challenge.

KVM again performs well across both the Delta and Bespin systems. In the case of the Delta system, in fact, KVM, significantly outperforms the base system. We determined this to be caused by KVM's support for transparent hugepages. When sufficient memory is available, transparent hugepages may be used to back the entirety of the guest VM's memory. Hugepages have previously been shown to improve TLB performance for KVM guests, and have been shown to occasionally boost the performance of a KVM guests beyond its host [201]. After disabling hugepages on the KVM host for the Delta system, performance dropped to 80–87% of the base system, suggesting that memory optimizations such as transparent hugepages can substan-

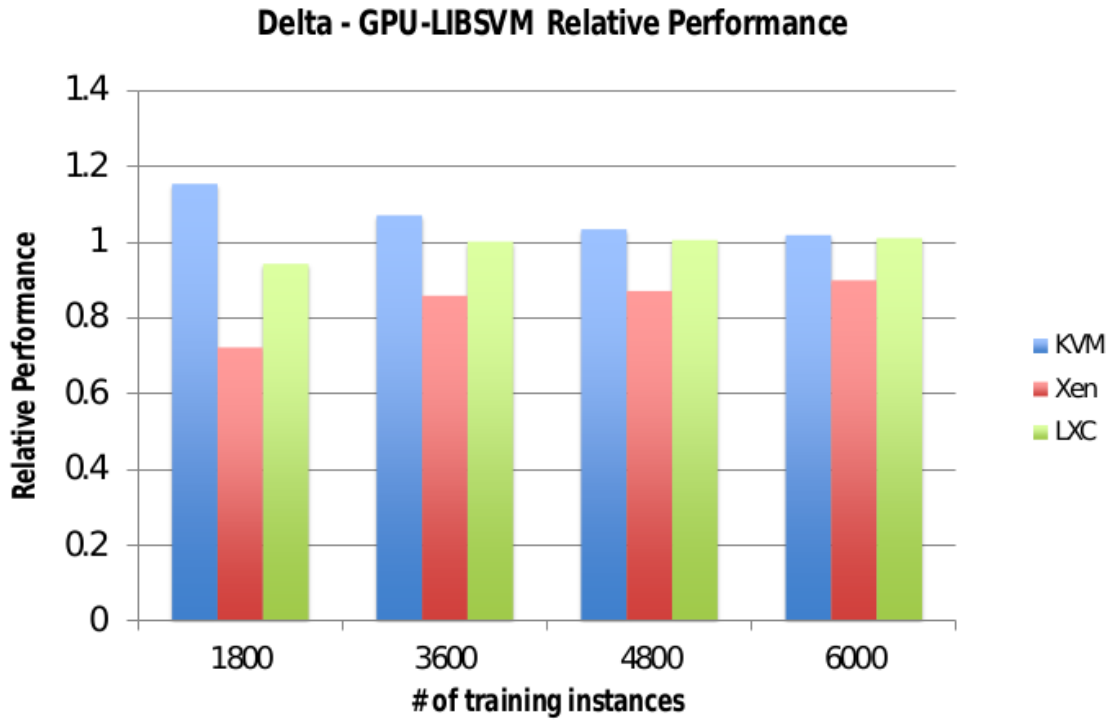


Figure 5.6 GPU-LIBSVM relative performance on Delta showing improved performance within virtual environments. Higher is better.

tially improve the performance of virtualized guests under some circumstances. LXC performs close to the base system, while Xen achieves between 72–90% of the base system's performance. We speculate that this may be related to Xen's inability to enabled page sizes larger than 4k.

5.5.3 LAMMPS Performance

In Figures 5.7 and 5.8, we show the LAMMPS Rhodopsin protein simulation results. LAMMPS is unique among our benchmarks, in that it exercises both the GPU and multiple CPU cores. In keeping with the LAMMPS benchmarking methodology, we execute the benchmark using 1 GPU and 1–8 CPU cores on the Bespin system, selecting the highest performing configuration. In the case of the Delta system, we execute the benchmark on 1–6 cores and 1 GPU, selecting the highest performing

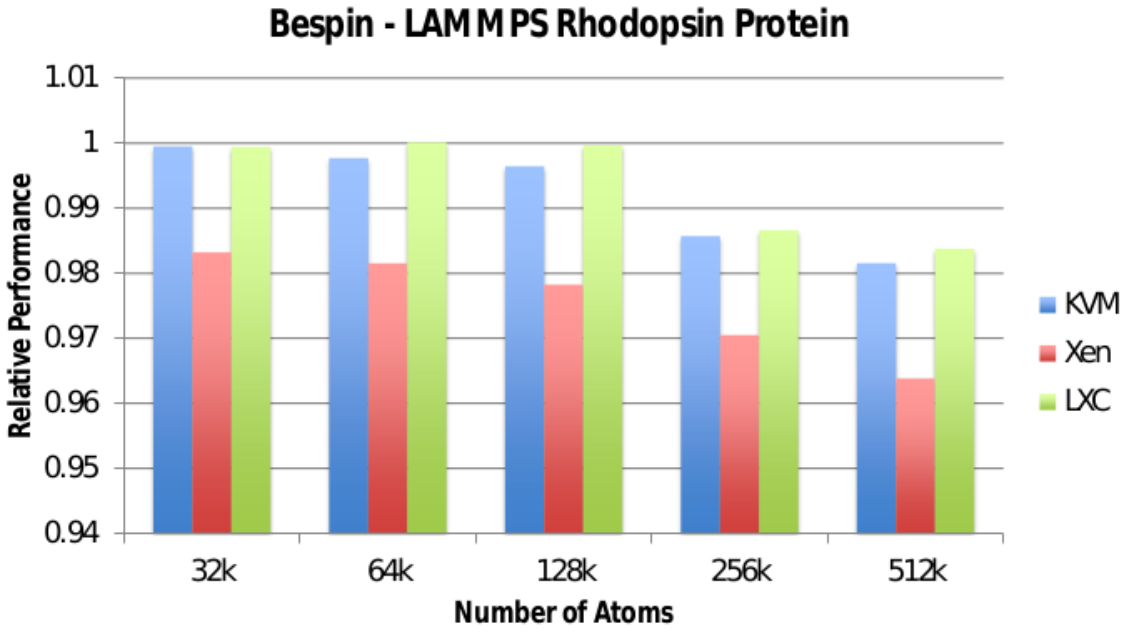


Figure 5.7 LAMMPS Rhodopsin benchmark relative performance for Bespin system. Higher is better.

configuration.

Overall, LAMMPS performs well across both hypervisors and systems. Surprisingly, LAMMPS showed better efficiency on the Delta system than the Bespin system, achieving greater than 98% efficiency across the board, while Xen on the Bespin system occasionally drops as low as 96.5% efficiency.

This performance is encouraging because it suggests that even heterogeneous CPU + GPU code is capable of performing well in a virtualized environment. Unlike SHOC, GPU-LIBSVM, and LULESH, LAMMPS fully exercises multiple host CPUs, splitting work between one or more GPUs and one or more CPU cores. This has the potential to introduce additional performance overhead, but the results do not bear this out in the case of LAMMPS.

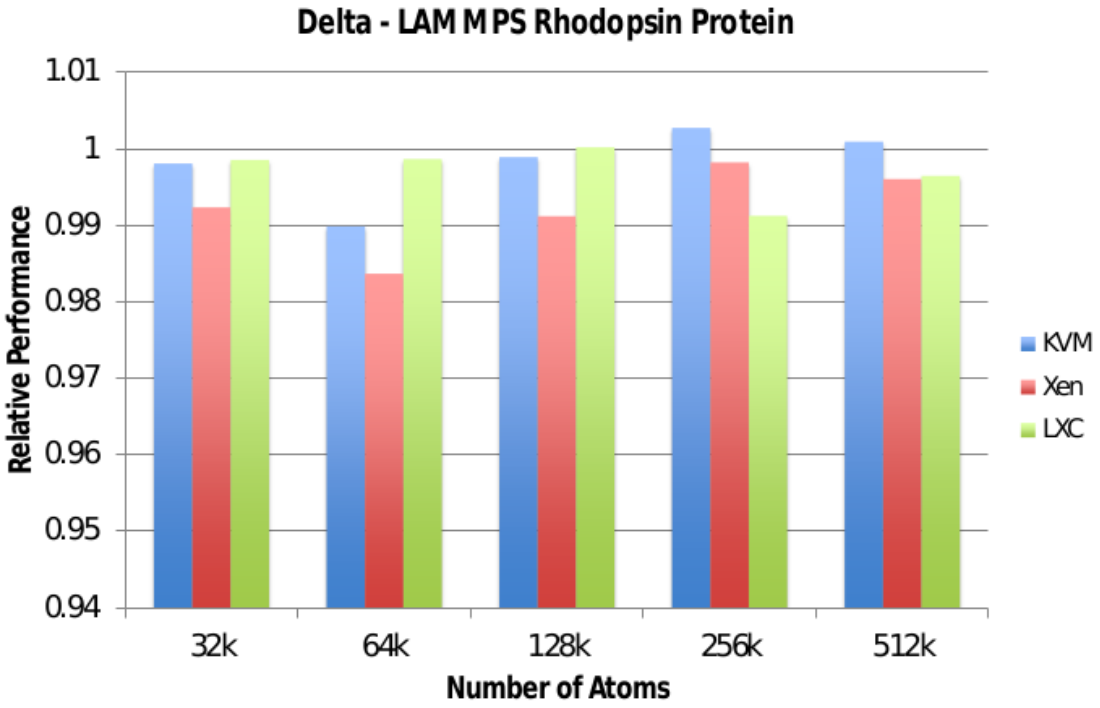


Figure 5.8 LAMMPS Rhodopsin benchmark relative performance for Delta system. Higher is better.

5.5.4 LULESH Performance

In Figure 5.9 we present our LULESH results for problem sizes ranging from mesh resolutions of $N = 30$ to $N = 150$. LULESH is a highly compute-intensive simulation, with limited data movement between the host/virtual machine and the GPU, making it ideal for GPU acceleration. Consequently, we would expect little overhead due to virtualization. We show LULESH results only on the Bespin system, because differences in the code bases between the Kepler and Fermi implementations led to unsound comparisons.

While, overall, we see very little overhead, there is a slight scaling effect that is most apparent in the case of the Xen hypervisor. As the mesh resolution (N^3) increases from $N = 30$ to $N = 150$, we see that the Xen overhead decreases until Xen

performs on-par with KVM and LXC.

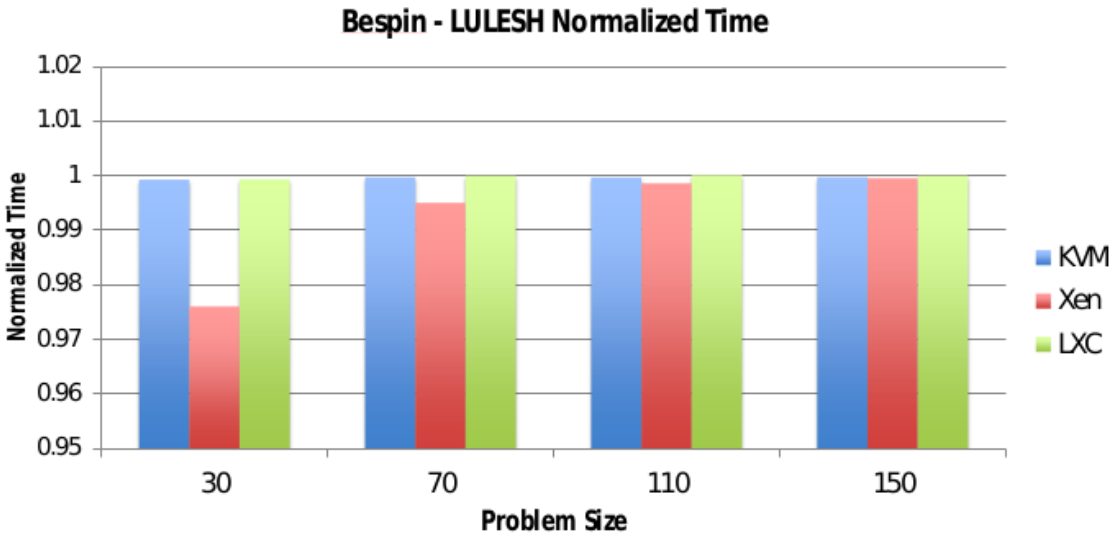


Figure 5.9 LULESH relative performance on Bespin. Higher is better.

5.6 Lessons Learned

Virtualizing performance-critical workloads has always proven controversial, whether the workload is CPU-only [172] or CPU with GPU. From our Westmere results, we can see that this criticism is in part legitimate, at times resulting in a performance penalty of greater than 10%, especially in the case of the Xen hypervisor. We believe much of this to be fixable with further software refinement.

At the same time, however, we have shown that the Sandy Bridge processor generation has nearly erased those performance overheads, suggesting that old arguments against virtualization for performance-critical tasks should be reconsidered. In light of this, the primary lesson from this study is that VGA-passthrough to virtual machines is achievable at low overhead, and across a variety of hypervisors and virtualization platforms. Virtualization performance remains inconsistent across hypervisors for the Westmere generation of processors, but starting with the Sandy Bridge architecture,

performance and consistency increase dramatically. In the case of the Sandy Bridge architecture, even the lowest performing hypervisor, open source Xen, typically performs within 95% of the base case.

This study has also yielded valuable insight into the merits of each hypervisor. KVM consistently yielded near-native performance across the full range of benchmarks. Its support for transparent hugepages resulted in slight performance boosts over-and-above even the base CentOS system in the case of the Delta system.

The Xen hypervisor was consistently average across both architectures, performing neither poorly nor extraordinarily well in any individual benchmark. Xen is the only hypervisor from this study that officially support VGA passthrough. As a result, PCI passthrough support in both Xen is more robust than KVM. We expect that this advantage will not last long, as commercial solutions targeting PCI passthrough in KVM are becoming common, particularly with regard to SR-IOV and networking adapters.

Linux Containers (LXC), consistently performed closest to the native case. This, of course, is not surprising given that LXC guests share a single kernel with their hosts. This performance comes at the cost of both flexibility and security, however. LXC is less flexible than its full virtualization counterparts, offering support for only Linux guests. More importantly, LXC device passthrough has security implications for multi-GPU systems. In the case of a multi-GPU-enabled host with NVIDIA hardware, both GPUs must be passed to the LXC guest in order to initialize the driver. This limitation may be addressable in future revisions to the NVIDIA driver.

5.7 Directions for Future Work

In this chapter we have characterized the performance of common hypervisors across two generations of GPUs and two host microarchitectures, and across 3 sets of bench-

marks. We showed the dramatic improvement in virtualization performance between the Fermi/Westmere and the Kepler/Sandy Bridge system, with the Sandy Bridge system typically performing within 1% of the base system. Finally, this study sought to characterize the GPU and CPU+GPU performance with carefully tuned hypervisor and guest configurations, especially with respect to NUMA. Improvements must be made to today's hypervisors in order to improve virtual NUMA support. Finally, cloud infrastructure, such as OpenStack, must be capable of automatically allocating virtual machines in a NUMA-friendly manner in order to achieve acceptable results at cloud-scale.

The next step in this work is to move beyond the single node to show that clusters of accelerators can be efficiently used with minimal overhead. This will require studies in high speed networking, particularly SR-IOV-enabled ethernet and Infiniband. Special attention is needed to ensure that latencies remain tolerable within virtual environments. Some studies have begun to examine these issues [202], but open questions remain.

Chapter 6

Supporting High Performance Molecular Dynamics in Virtualized Clusters using IOMMU, SR-IOV, and GPUDirect

6.1 Abstract

Cloud infrastructure-as-a-Service paradigms have recently shown their utility for a vast array of computational problems, ranging from advanced web service architectures to high throughput computing. However, many scientific computing applications have been slow to adapt to virtualized cloud frameworks. This is due to performance impacts of virtualization technologies, coupled with the lack of advanced hardware support necessary for running many high performance scientific a:tabnpplications at scale.

By using KVM virtual machines that leverage both Nvidia GPUs and InfiniBand, we show that molecular dynamics simulations with LAMMPS and HOOMD run at near-native speeds. This experiment also illustrates how virtualized environments can support the latest parallel computing paradigms, including both MPI+CUDA and new GPUDirect RDMA functionality. Specific findings show initial promise in scaling of such applications to larger production deployments targeting large scale computational workloads.

6.2 Introduction

At present we stand at the inevitable intersection between High Performance Computing (HPC) and clouds. Various platform tools such as Hadoop and MapReduce, among others, have already percolated into data intensive computing within HPC [26]. In addition, there are efforts to support traditional HPC-centric scientific computing applications in virtualized cloud infrastructure. There are a multitude of reasons for supporting parallel computation in the cloud [59], including features such as dynamic scalability, specialized operating environments, simple management interfaces, fault tolerance, and enhanced quality of service, to name a few. The growing importance of supporting advanced scientific computing using virtualized infrastructure can be seen by a variety of new efforts, including the NSF-funded Comet resource part of XSEDE at San Diego Supercomputer Center [203].

Nevertheless, there exists a past notion that virtualization used in today's cloud infrastructure is inherently inefficient. Historically, cloud infrastructure has also done little to provide the necessary advanced hardware capabilities that have become almost mandatory in supercomputers today, most notably advanced GPUs and high-speed, low-latency interconnects. The result of these notions has hindered the use of virtualized environments for parallel computation, where performance must be paramount.

A growing effort is currently underway that looks to systematically identify and reduce any overhead in virtualization technologies. This effort has, thus far, proven to be a qualified success [172, 204], though further research is needed to address issues of scalability and I/O. Thus, we see a constantly diminishing overhead with virtualization, not only with traditional cloud workloads [205] but also with HPC workloads. While virtualization will almost always include some additional overhead

in relation to its dynamic features, the eventual goal for supporting HPC in virtualized environments is to minimize what overhead exists whenever possible. To advance the placement of HPC applications on virtual machines, new efforts are emerging which focus specifically on key hardware now commonplace in supercomputers. By leveraging new virtualization tools such as IOMMU device passthrough and SR-IOV, we can now support the such advanced hardware as the latest Nvidia Tesla GPUs [206] as well as InfiniBand fabrics for high performance networking and I/O [207, 208].

With the advances in hypervisor performance coupled with the newfound availability of HPC hardware in virtual machines analogous to the most powerful supercomputers used today, we see can see the possibility of a high performance cloud infrastructure using virtualization. While our previous efforts in this area have focused on single-node advancements, it is now imperative to ensure real-world applications can also operate in distributed environments as found in today's cluster and cloud infrastructures.

Efforts to improve power efficiency and performance in data centers has led to more heterogeneous architectures. That move toward heterogeneity has, in turn, led to support for heterogeneity in the cloud. For example, Amazon EC2 supports GPU accelerators in EC2 [209], and OpenStack supports heterogeneity using flavors [210]. These advancements in cloud-level support for heterogeneity combined with better support for high-performance virtualization makes the use of cloud for HPC much more feasible for a wider range of applications and platforms.

In this paper we describe background a related work. Then, we describe a heterogeneous cloud platform, based on OpenStack. This effort has been under development at USC/ISI since 2011 [166]. We describe our work towards integrating GPU and InfiniBand support into OpenStack, and we describe the heterogeneous scheduling additions that are necessary to support not only attached accelerators, but any cloud

composed of heterogeneous elements.

We then demonstrate running two molecular dynamics simulations, LAMMPS and HOOMD, in a virtual infrastructure complete with both Kepler GPUs and QDR InfiniBand. Both HOOMD and LAMMPS are used extensively in some of the world's fastest supercomputers and represent example simulations that HPC supports today. We show that these applications are able to run at near-native speeds within a completely virtualized environment, demonstrating just small performance impacts that are usually acceptable by many users. Furthermore, we demonstrate the ability of such a virtualized environment to support cutting edge software tools such as RDMA GPUDirect, illustrating how cutting-edge HPC technologies are also possible in a virtualized environment.

Following these efforts, we hope to ensure upstream infrastructure projects such as OpenStack [131, 211] are able to make effective and quick use of these features, allowing users to build private cloud infrastructure to support high performance distributed computational workloads.

6.3 Background and Related Work

Virtualization technologies and hypervisors have been seen widespread deployment in support of a vast array of applications. This ranges from public commercial Cloud deployments such as Amazon EC2 [212, 213], Microsoft Azure [214], and Google's Cloud Platform [215] to private deployments within colocation facilities, corporate data centers, and even national scale cyber infrastructure initiatives. All these support look to support various use cases and applications such as web servers, ACID and BASE databases, online object storage, and even distributed systems, to name a few.

The use of virtualization and hypervisors specifically support various HPC so-

lutions has been studied with mixed results. In [172], it is found that there is a great deal of variance between hypervisors when running various distributed memory and MPI applications, finding that KVM overall performed well across an array of HPC benchmarks. Furthermore, some applications may not may fit well into default virtualized environments, such as High Performance Linpack [204]. Other studies have specifically looked at interconnect performance in virtualization and found the best-case scenario to be lacking [216] with up to 60% performance penalties with conventional techniques.

Recently, various CPU architectures have added support for I/O virtualization mechanisms in the CPU ISA through the use of an I/O memory management unit (IOMMU). Often, this is referred to as PCI Passthrough, as it enabled devices on the PCI-Express bus to be passed directly to a specific virtual machine (VM). Specific hardware implementations include Intel's VT-d [217], AMD's IOMMU [218] from x86_64 architectures, and even more recently ARM System MMU [219]. All of these implementations effectively look to aid in the usage of DMA-capable hardware to be used within a specific virtual machine. Using these features, a wide array of hardware can be utilized directly within VMs and enable fast and efficient computation and I/O capabilities.

With PCI Passthrough, a PCI device is handed directly to a running (or booting) VM, thereby relinquishing control of the device within the host entirely. This is different from typical VM usage where hardware is emulated in the host and used in a guest VM, such as with bridged ethernet adapters or emulated VGA devices. Performing PCI Passthrough requires the host to seize the device upon boot using a specialized driver to effectively block normal driver initialization. In the instance of the KVM hypervisor, this is done using the *vfio* and *pci_stub* drivers. Then, this driver relinquishes control to the VM, whereby normal device drivers initiate the hardware

and enable the device for use by the guest OS.

6.3.1 GPU Passthrough

Nvidia GPUs comprise the single most common accelerator in the Nov 2014 Top 500 List [8] and represent an increasing shift towards accelerators for HPC applications. Historically, GPU usage in a virtualized environment has been difficult, especially for scientific computation. Various front-end remote API implementations have been developed to provide CUDA and OpenCL libraries in VMs, which translate library calls to a back-end or remote GPU. One common use case of this is rCUDA [56], which provides a front-end CUDA API within a VM or any compute node, and then sends the calls via Ethernet or InfiniBand to a separate node with 1 or more GPUs. While this method is valid, it has the drawback of relying on the interconnect itself and the bandwidth available, which can be especially problematic on Ethernet. Furthermore, as this method consumes bandwidth, it can leave little remaining for MPI or RDMA routines, thereby constructing a bottleneck for some MPI+CUDA applications that depend on inter-process communication.

Recently efforts have been seen to support such GPU accelerators within VMs using IOMMU technologies, with implementations now available with KVM [206], Xen [220] and VMWare [221]. These efforts have shown that GPUs can achieve up to 99% of their bare metal performance when passed to a virtual machine using PCI Passthrough. VMWare specifically shows how the such PCI Passthrough solutions perform well and are likely to outperform front-end Remote API solutions such as rCUDA within a VM [221]. While these works demonstrate PCI Passthrough performance across a range of hypervisors and GPUs, they have been limited to investigating single node performance until now.

6.3.2 SR-IOV and InfiniBand

With almost all parallel HPC applications, the interconnect fabric which enables fast and efficient communication between processors becomes a central requirement to achieving good performance. Specifically, a high bandwidth link is needed for distributed processors to share large amounts of data across the system. Furthermore, low latency becomes equally important for ensuring quick delivery of small message communications and resolving large collective barriers within many parallelized codes. One such interconnect, InfiniBand, has become the most common implementation used within the Top500 list. However previously InfiniBand was inaccessible to virtualized environments.

Supporting I/O interconnects in VMs has been aided by Single Root I/O Virtualization (SR-IOV), whereby multiple virtual PCI functions are created in hardware to represent a single PCI device. These virtual functions (VFs) can then be passed to a VM and used as by the guest as if it had direct access to that PCI device. SR-IOV allows for the virtualization and multiplexing to be done within the hardware, effectively providing higher performance and greater control than software solutions.

SR-IOV has been used in conjunction with Ethernet devices to provide high performance 10Gb TCP/IP connectivity within VMs [222], offering near-native bandwidth and advanced QoS features not easily obtained through emulated Ethernet offerings. Currently Amazon EC2 offers a high performance VM solution utilizing SR-IOV enabled 10Gb Ethernet adapters. While SR-IOV enabled 10Gb Ethernet solutions offers a big forward in performance, Ethernet still does not offer the high bandwidth or low latency typically found with InfiniBand solutions.

Recently SR-IOV support for InfiniBand has been added by Mellanox in the ConnectX series adapters. Initial evaluation of SR-IOV InfiniBand within KVM VMs

has proven has found point-to-point bandwidth to be near-native, but up to 30% latency overhead for very small messages [207, 223]. However, even with the noted overhead, this still signifies up to an order of magnitude difference in latency between InfiniBand and Ethernet with VMs. Furthermore, advanced configuration of SR-IOV enabled InfiniBand fabric is taking shape, with recent research showing up to a 30% reduction in the latency overhead [208]. However, real application performance has not yet been well understood until now.

6.3.3 GPUDirect

NVIDIA's GPUDirect technology was introduced to reduce the overhead of data movement across GPUs [224, 225]. GPUDirect supports both networking as well as peer-to-peer interfaces for single node multi-GPU systems. The most recent implementation of GPUDirect, version 3, adds support for RDMA over InfiniBand for Kepler-class GPUs.

The networking component of GPUDirect relies on three key technologies: CUDA 5 (and up), a CUDA-enabled MPI implementation, and a Kepler-class GPU (RDMA only). Both MVAPICH and OpenMPI support GPUDirect. Support for RDMA over GPUDirect is enabled by the MPI library, given supported hardware, and does not depend on application-level changes to a user's code.

In this paper, our GPUDirect work focuses on GPUDirect v3 for multi-node RDMA support. We demonstrate scaling for up to 4 nodes connected via QDR InfiniBand and show that GPUDirect RDMA improves both scalability and overall performance by approximately 9% at no cost to the end user.

6.4 A Cloud for High Performance Computing

With support for GPU Passthrough, SR-IOV, and GPUDirect, we have the building blocks for a high performance, heterogeneous cloud. In addition, other common accelerators (e.g. Xeon Phi [226]) have similarly been demonstrated in virtualized environments. Our vision is of a heterogeneous cloud, supporting both high speed networking and accelerators for tightly coupled applications.

To this end we have developed a heterogeneous cloud based on OpenStack [131]. In our previous work, we have demonstrated the ability to rapidly provision GPU, bare metal, and other heterogeneous resources within a single cloud [166]. Building on this effort we have added support for GPU passthrough to OpenStack as well as SR-IOV support for both ConnectX-2 and ConnectX-3 Infiniband devices. Mellanox separately supports an OpenStack InfiniBand networking plugin for OpenStack's Neutron service [227], however the Mellanox plugin depends on the ConnectX-3 adapter. Our institutional requirements depend on ConnecteX-2 SR-IOV support, requiring an independent implementation.

OpenStack supports services for networking (Neutron), compute (Nova), identity (Keystone), storage (Cinder, Swift), and others. Our work focuses entirely on the compute service.

Scheduling is implemented at two levels: the cloud-level and the node-level. In our earlier work, we have developed a cloud-level heterogeneous scheduler for OpenStack, allowing scheduling based on architectures and resources [166]. In this model, the cloud-level scheduler dispatches jobs to nodes based on resource requirements (e.g. Kepler GPU) and node-level resource availability.

At the node, a second level of scheduling occurs to ensure that resources are tracked and not over-committed. Unlike traditional cloud paradigms, devices passed

into VMs cannot be over-committed. We treat devices, whether GPUs or InfiniBand virtual functions, as schedulable resources. Thus, it is the responsibility of the individual node to track resources committed and report availability to the cloud-level scheduler. For reporting, we piggyback on top of OpenStack's existing reporting mechanism to provide a low overhead solution.

6.5 Benchmarks

We selected two molecular dynamics (MD) applications for evaluation in this study: LAMMPS and HOOMD [228, 229]. These MD simulations are chosen to represent a subset of advance parallel computation for a number of fundamental reasons:

- MD simulations provide a practical representation of N-Body simulations, which is one of the major computational *Dwarfs* [230] in parallel and distributed computing.
- MD simulations are one of the most widely deployed applications on large scale supercomputers today.
- Many MD simulations have a hybrid MPI+CUDA programming model, which has often become commonplace in HPC as the use of accelerators increases.

As such, we look to LAMMPS and HOOMD to provide a real-world example for running cutting-edge parallel programs on virtualized infrastructure. While these applications by no means represent all parallel scientific computing efforts (as justified by the 13 Dwarfs defined in [230]), we hope these MD simulators offer a more pragmatic viewpoint than traditional synthetic HPC benchmarks such as High Performance Linpack.

LAMMPS The Large-scale Atomic/Molecular Parallel Simulator is a well understood highly parallel molecular dynamics simulator. It supports both CPU and GPU-based workloads. Unlike many simulators, both MD and otherwise, LAMMPS is heterogeneous. It will use both GPUs and multicore CPUs concurrently. For this study, this heterogeneous functionality introduces additional load on the host, allowing LAMMPS to utilize all available cores on a given system. Networking in LAMMPS is accomplished using a typical MPI model. That is, data is copied from the GPU back to the host and sent over the InfiniBand fabric. No RDMA is used for these experiments.

HOOMD-blue The Highly Optimized Object-oriented Many-particle Dynamics – Blue Edition is a particle dynamics simulator capable of scaling into the thousands of GPUs. HOOMD supports executing on both CPUs and GPUs. Unlike LAMMPS, HOOMD is homogeneous and does not support mixing of GPUs and CPUs. HOOMD supports GPUDirect using a CUDA-enabled MPI. In this paper we focus on HOOMD's support for GPUDirect and show its benefits for increasing cluster sizes.

6.6 Experimental Setup

Using two molecular dynamics tools, LAMMPS [228] and HOOMD [229], we demonstrate a high performance *system*. That is, we combine PCI passthrough for Nvidia Kepler-class GPUs with QDR Infiniband SR-IOV and show that high performance molecular dynamics simulations are achievable within a virtualized environment.

For the first time, we also demonstrate Nvidia GPUDirect technology within such a virtual environment. Thus, we look to not only illustrate that virtual machines provide a flexible high performance infrastructure for scaling scientific workloads in-

cluding MD simulations, but also that the latest HPC features and programming environments are also available in this same model.

6.6.1 Node configuration

To support the use of Nvidia GPUs and InfiniBand within a VM, specific and exact host configuration is needed. This node configuration is illustrated in Figure 6.1. While our implementation is specific to the KVM hypervisor, this setup represents a design that can be hypervisor agnostic.

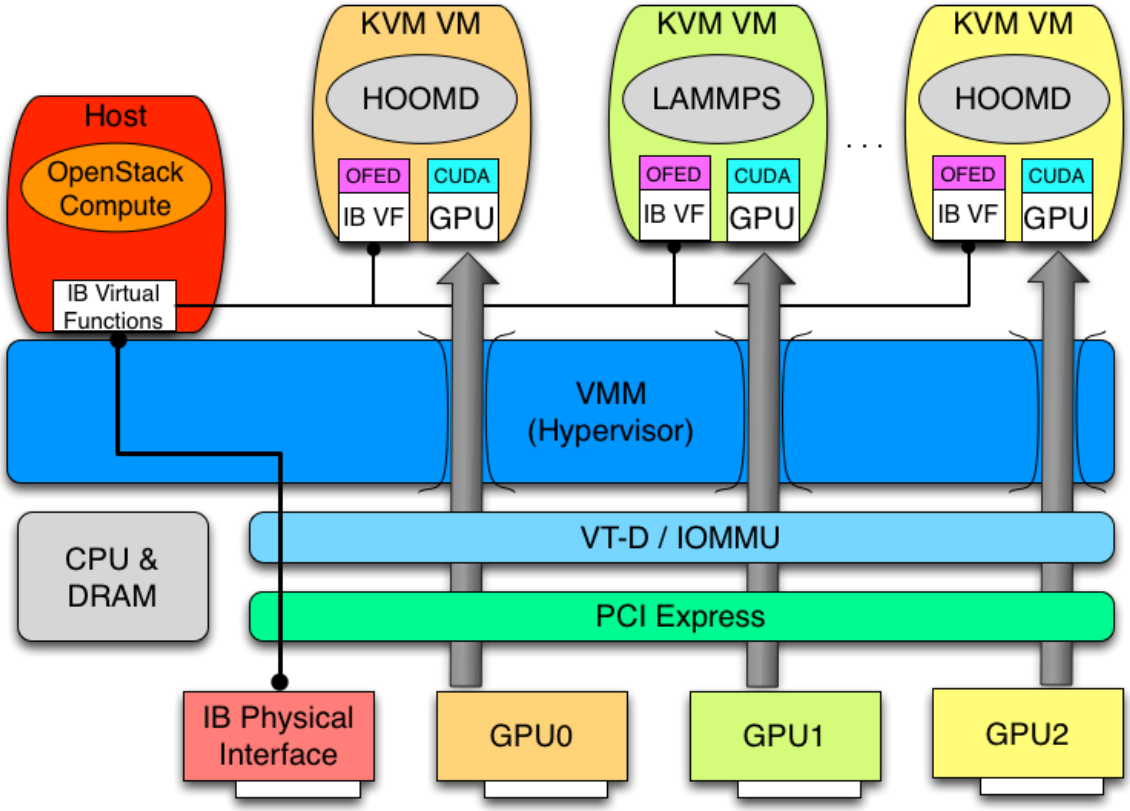


Figure 6.1 Node PCI Passthrough of GPUs and InfiniBand

Each node in the testbed uses CentOS 6.4 with a 3.13 upstream Linux kernel for the host OS, along with the latest KVM hypervisor, QEMU 2.1, and the *vfio* driver. Each Guest VM runs CentOS 6.4 with a stock 2.6.32-358.23.2 kernel. A Kepler GPU

is passed through using PCI Passthrough and directly initiated within the VM via the Nvidia 331.20 driver and CUDA release 5.5. While this specific implementation used only a single GPU, it is also possible to include as many GPUs as one can fit within the PCI Express bus if desired. As the GPU is used by the VM, an on-board VGA device was used by the host and a standard Cirrus VGA was emulated in the guest OS.

With using SR-IOV, the OFED drivers version 2.1-1.0.0 are used with Mellanox ConnectX-3 VPI adapter with firmware 2.31.5050. The host driver initiates 4 VFs, one of which is passed through to the VM where the default OFED mlnx_ib drivers are loaded.

6.6.2 Cluster Configuration

Our test environment is composed of 4 servers each with a single Nvidia Kepler-class GPU. Two servers are equipped with K20 GPUs, while the other two servers are equipped with K40 GPUs, demonstrating the potential for a more heterogeneous deployment. Each server is composed of 2 Intel Xeon E5-2670 CPUs, 48GB of DDR3 memory, and Mellanox ConnectX-3 QDR InfiniBand. CPU sockets and memory are split evenly between the two NUMA nodes on each system. All InfiniBand adapters use a single Voltaire 4036 QDR switch with a software subnet manager for IPoIB functionality.

For these experiments, both the GPUs and InfiniBand adapters are attached to NUMA node 1 and both the guest VMs and the base system utilized identical software stacks. Each guest was allocated 20 GB of RAM and a full socket of 8 cores, and pinned to NUMA node 1 to ensure optimal hardware usage. While all VMs are capable of login via the InfiniBand IPoIB setup, a 1Gb Ethernet network was used for all management and login tasks.

For a fair and effective comparison, we also use a native environment without any virtualization. This native environment employs the same hardware configuration, and like the Guest OS runs CentOS 6.4 with the stock 2.6.32-358.23.2 kernel.

6.7 Results

In this section, we discuss the performance of both the LAMMPS and HOOMD molecular dynamics simulation tools when running within a virtualized environment. Specifically, we scale each application to 32 cores and 4 GPUs, both in a native bare-metal and virtualized environments. Each application set was run 10 times, with the results averaged accordingly.

6.7.1 LAMMPS

Figure 6.2 shows one of the most common LAMMPS algorithms used; the Lennard-Jones potential (LJ). This algorithm is deployed in two main configurations - a 1:1 core to GPU mapping, and a 8:1 core to GPU mapping. With the LAMMPS GPU implementation, a delicate balance between GPUs and CPUs is required to find the optimal ratio for fastest computation, however here we just look at the two most obvious choices. With small problem sizes, the 1:1 mapping outperforms the more complex core deployment, as the problem does not require the additional complexity provided with multi-core solution. As expected the multi-core configuration quickly offers better performance for larger problem sizes, achieving roughly twice the performance with all 8 available cores. This is largely due to the availability of all 8 cores to keep the GPU running 100% with continual computation.

The important factor for this manuscript is the relative performance of the virtualized environment. From the results, it is clear the VM solution performs very well compared to the best-case native deployment. For the multi-core configuration across

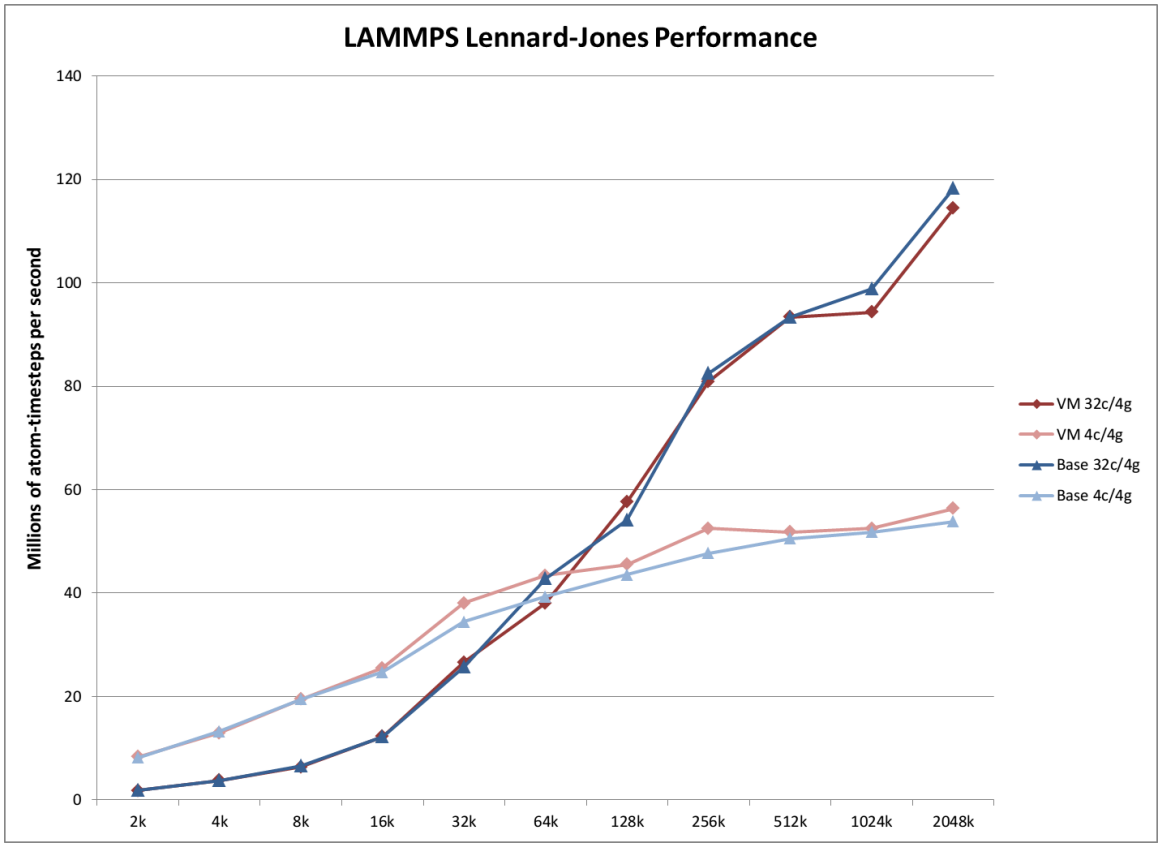


Figure 6.2 LAMMPS LJ Performance

all problem sizes, the virtualized deployment averaged 98.5% efficiency compared to native. The single core per GPU deployment reported better-than native performance at 100% native. This is likely due to caching effects, but further investigation is needed to fully identify this occurrence.

Another common LAMMPS algorithm, the Rhodopsin protein in solvated lipid bilayer benchmark (Rhodo), was also run with results given in Figure 6.3. As with the LJ runs, we see the multi-core to GPU configuration resulting in higher computational performance for the larger problem sizes compared to the single core per GPU configuration, as expected.

Again, the overhead of the virtualized configuration remains low across all configurations and problem sizes, with an average 96.4% efficiency compared to native.

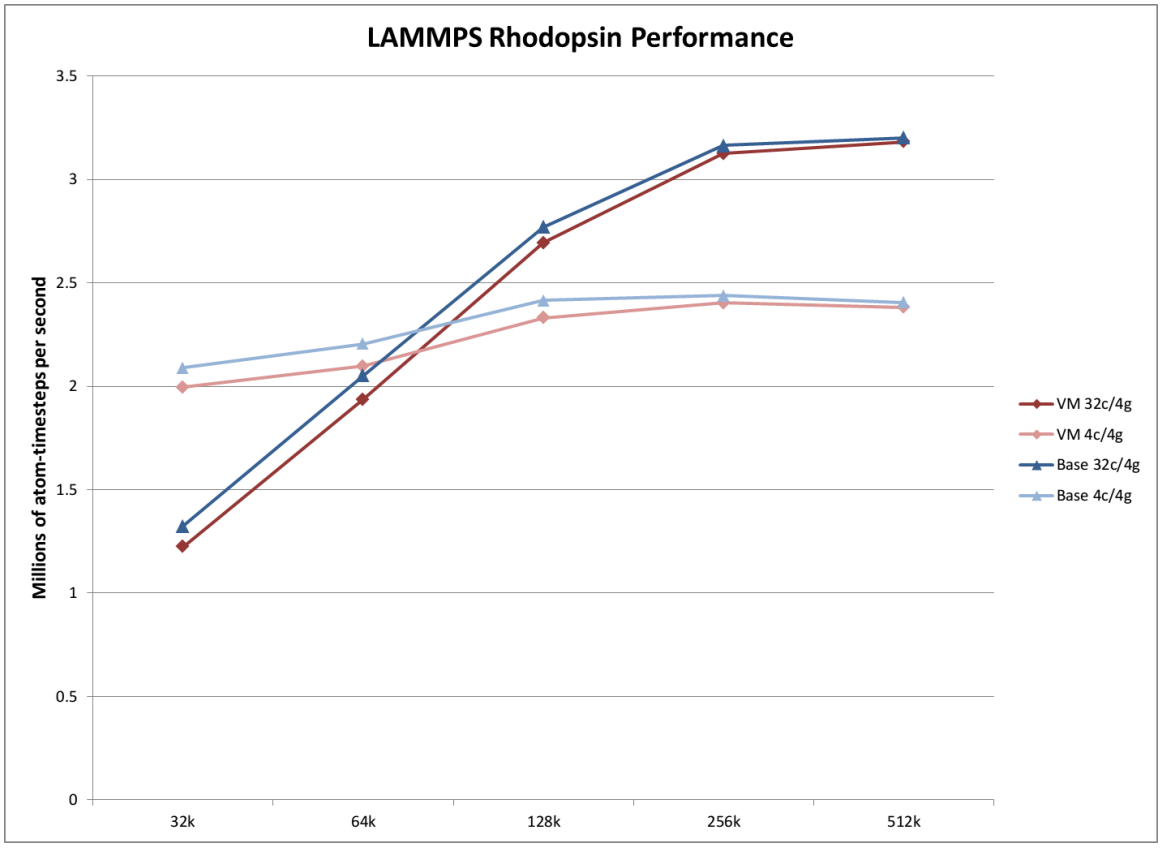


Figure 6.3 LAMMPS RHODO Performance

Interestingly enough, we also see the performance gap decrease as the problem size increases, with the 512k problem size in yielding 99.3% of native performance. This finding leads us to extrapolate that a virtualized MPI+CUDA implementation would scale to a larger computational resource with similar success.

6.7.2 HOOMD

In Figure 6.4 we show the performance of a Lennard-Jones liquid simulation with 256K particles running under HOOMD. HOOMD includes support for CUDA-aware MPI implementations via GPUDirect. The MVAPICH 2.0 GDR implementation enables a further optimization by supporting RDMA for GPUDirect. From Figure 6.4 we can see that HOOMD simulations, both with and without GPUDirect, perform very

near-native. The GPUDirect results at 4 nodes achieve 98.5% of the base system's performance. The non-GPUDirect results achieve 98.4% efficiency at 4 nodes. These results indicate the virtualized HPC environment is able to support such complex workloads. While the effective testbed size is relatively small, it indicates that such workloads may scale equally well to hundreds or thousands of nodes.

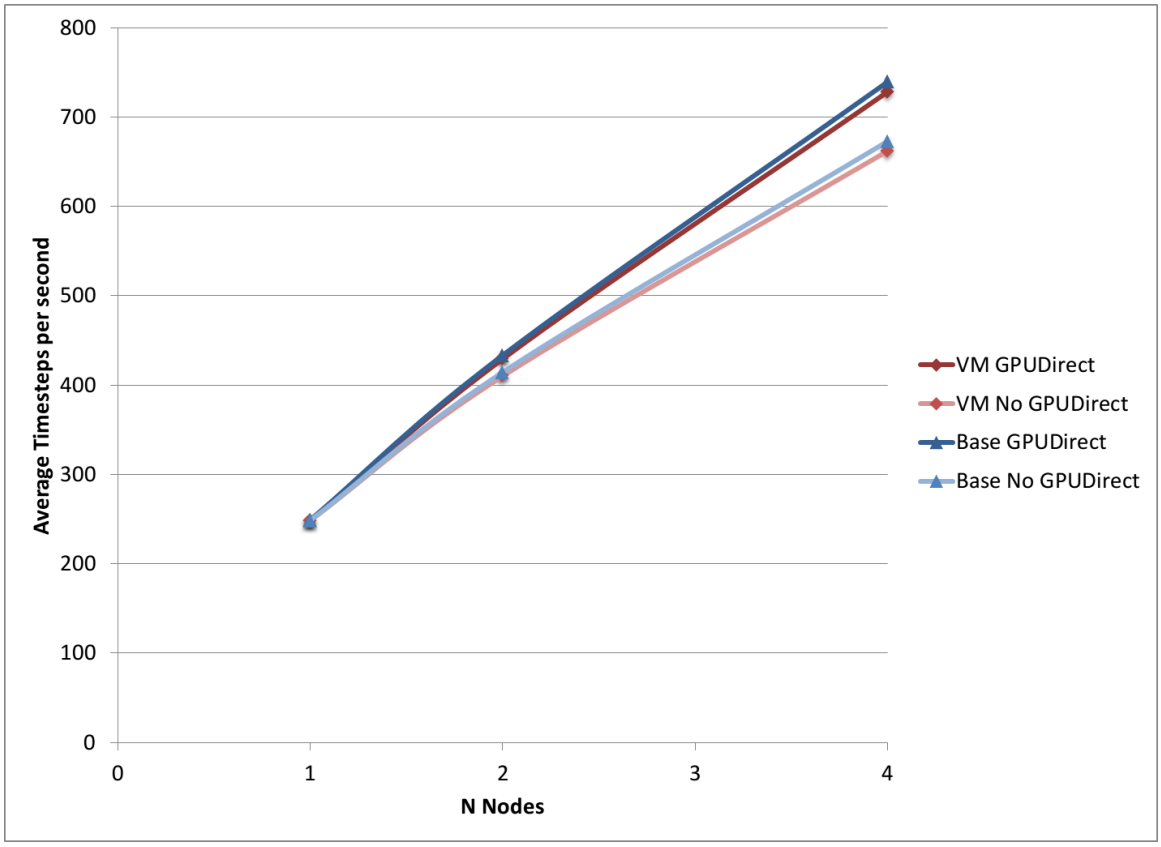


Figure 6.4 HOOMD LJ Performance with 256k Simulation

6.8 Discussion

From the results, we see the potential for running HPC applications in a virtualized environment using GPUs and InfiniBand interconnect fabric. Across all LAMMPS runs with ranging core configurations, we found only a 1.9% overhead between the KVM virtualized environment and native. For HOOMD, we found a similar 1.5%

overhead, both with and without GPU Direct. These results go against conventional wisdom that HPC workloads do not work in VMs. In fact ,we show two N-Body type simulations programmed in an MPI+CUDA implementation perform at roughly near-native performance in tuned KVM virtual machines.

With HOOMD, we see how GPUDirect RDMA shows a clear advantage over the non-GPUDirect implementation, achieving a 9% performance boost in both the native a virtualized experiments. While GPUDirect's performance impact has been well evaluated previously [224], it is the author's belief that this manuscript represents the first time GPUDirect has has been utilized in a virtualized environment.

Another interesting finding of running LAMMPS and HOOMD in a virtualized environment is as workload scales from a single node to 32 cores, the overhead does not increase. These results lend credence to the notion that this solution would also work for a much larger deployment. Specifically, it would be possible to expand such computational problems to a larger deployment in FutureGrid [231], Chameleon Cloud [232], or even the planned NSF Comet machine at SDSC, scheduled to provide up to 2 Petaflops of computational power. Effectively, these results help support the theory that a majority of HPC computations can be supported in virtualized environment with minimal overhead.

6.9 Chapter Summary

With the advent of cloud infrastructure, the ability to run large-scale parallel scientific applications has become possible but limited due to both performance and hardware availability concerns. In this work we show that advanced HPC-oriented hardware such as the latest Nvidia GPUs and InfiniBand fabric are now available within a virtualized infrastructure. Our results find MPI + CUDA applications such as molecular dynamics simulations run at near-native performance compared to tra-

ditional non-virtualized HPC infrastructure, with just an averaged 1.9% and 1.5% overhead for LAMMPS and HOOMD, respectively. Moving forward, we show the utility of GPUDirect RDMA for the first time in a cloud environment with HOOMD. Effectively, we look to pave the way for large-scale virtualized cloud Infrastructure to support a wide array of advanced scientific computation commonly found running on many supercomputers today. Our efforts leverage these technologies and provide them in an open source Infrastructure-as-a-Service framework using OpenStack.

Chapter 7

Virtualization advancements to support HPC applications

Throughout this dissertation, the question of whether cloud infrastructure, using virtualization, can support mid-tier scientific computation has been investigated. These scientific problems are tightly coupled, distributed memory computations, and in the past have failed to perform well on traditional public and private cloud systems. It is our estimate that this is due to a number of reasons, including hypervisor design and implementation, lack of hardware advances, VM placement inefficiencies, and the lack of hardware availability in virtualization. In Chapter 3, we discussed the base case with off-the-shelf single node configurations. This showed that hypervisor selection matters a great deal in performance and reliability, and that some applications can perform well on a single node. This study expressly avoided multi-node configurations due to the previously documented issues with Ethernet interconnects in virtualized environments [48].

With the rise of GPUs in HPC, Chapter 4 offers a first solution in Xen to GPU passthrough as an alternative to either no GPU availability or providing GPUs access across a network by virtualizing the front end API libraries. It was found that our method of GPU passthrough incurred some overhead with Xen on older x86 hardware when transferring data across the PCI-Express bus, but provided the best option at the time. As methods were derived for other hypervisors such as KVM and VMWare, we looked to evaluate all options for a multitude of computational

problems, including both CUDA and OpenCL codes in Chapter 5. This research found that newer hardware without the QPI interconnect between the socket and the PCI-Express bus (Sandy-Bridge and up CPUs) can yield near-native performance for a range of applications. Specifically, it was found that the KVM hypervisor was the most performant and stable hypervisor, and LXC, a container option, also performed very well (although with security limitations).

Chapter 6 combines the lessons learned from KVM tuning, GPU passthrough, and intersects our other related efforts in inserting a high performance interconnect, in this particular case InfiniBand [208], into a virtualized cluster environment. Specifically, a test-bed was created across 4 nodes, each with Kepler GPUs and FDR InfiniBand passed through to VMs spanning the entire CPU set. InfiniBand was specifically set up to utilize SR-IOV, which allows for the multiplexing of the ConnectX-3 VPI card to the guest instances. From here, two Molecular Dynamics simulations were run both in this tuned KVM configuration and on bare-metal. The results indicate that overhead in virtualization is on the order of 1-2% for these HPC applications given a few different configurations and problem sizes.

From this work, a number of observations start to emerge. First, that virtualization may indeed be able to support HPC workloads given the advances. While the scale of the experiments of the virtual cluster is small at only a few dozen cores, the performance trends from scaling up HOOMD-blue in Figure 6.4 look to be very closely correlated between the virtualized and bare metal experiments. While this application, as well as other HPC applications using similar distributed memory models, all need to be replicated at a much higher scale, we have shown there are currently no limitations in doing so, aside from the availability of the infrastructure itself. It is our hope that with this knowledge, future test-beds can be assembled at a larger scale to further move this effort further forward.

Another observation is that the KVM hypervisor continually proves to be the most performance-oriented hypervisor studied in this dissertation. While this is a bit of surprise given it is a Type 2 hypervisor, which introduces additional potential for host noise and overhead, it nonetheless showed the smallest degree of variation between results, both in Chapter 3 experiments and again in Chapter 5. KVM also offers the best performance of hypervisors overall, and with many workloads, such as seen in Figure 5.2, where KVM often performs within 0.5% of native in SHOC benchmarks. Furthermore, KVM has support for CPU pinning, NUMA socket binding, PCI passthrough and SR-IOV, as well as transparent huge pages and advanced migration mechanisms (described in more detail later in this Chapter). While it is possible that other specialized hypervisors, such as Palacios [31] (not studied), could also perform similarly, the KVM hypervisor has a large community support, industry backing, and production-level integration into the latest private infrastructure, such as OpenStack.

A 3rd observation regarding high performance virtual clusters is that the SR-IOV InfiniBand integration described in Chapter 6 provides a drastic shift in the outlook for distributed memory applications. While SR-IOV InfiniBand does have a 15-30% overhead in latency compared to native implementations for small messages [208], this is still an order of magnitude better than current Ethernet options available from cloud providers such as Amazon. Furthermore, bandwidth of InfiniBand solutions in virtual clusters looks to be near-native, also surpassing current Ethernet deployments. It is also possible for other interconnects, such as Intel's emerging Omni-Path interconnect, to demonstrate similar or better results in a virtualized ecosystem, and future experimentation should try to leverage other interconnection options if the hardware supports it.

While these now smaller and better defined performance differences may not be

satisfactory for extreme-scale distributed memory applications, we expect a large amount of users with mid-tier scientific computational problems to be accepting of these small overheads when considering the value added by working in a virtualized environment. Armed with this knowledge, we find that the outlook for high performance virtual clusters to be promising.

However, next steps are needed to demonstrate the value of virtulaization, providing a more rich user experience while simultaneously further enhancing performance for many users. These value-added techniques, such as efficient tuning, scheduling, VM cloning, and compute-migration may in fact help enable new classes of scientific computations, not only within HPC applications, but also with big data platform services. We specifically focus on leveraging the KVM hypervisor in conjunction with a high speed, low latency interconnect to provide new features that otherwise have yet to be made possible. While much of this work is under construction, this nonetheless gives a glimpse at some future directions in high performance virtualization.

In the rest of this chapter, we look to review the methods utilized in this dissertation regarding virtualization, then identify research, design, and future implementation of advanced virtualization techniques to enable a new class of infrastructure with added performance and features that have yet to be realized. We focus on guest memory optimizations, live migration deployments, and integration with an RDMA-enabled interconnect for novel VM migration and cloning. It may be possible for these advancements, once implemented, to have an impact not only on cloud infrastructure, but also on dedicated environments, which support big data applications or other distributed memory applications that focus on both usability and performance.

7.1 PCI Passthrough

As cloud computing's reach into distributed systems increases, so does the requirement of virtualization to perform at near-native speeds and to take full advantage of the underlying hardware. While this does equate to fast and efficient hypervisors, it also alludes to the effective utilization of devices beyond CPU and memory. Such devices can vary widely and include hardware such as graphics processing units, networking adapters, I/O hubs, web cameras, secondary and tertiary storage, to name a few. Within virtualization, oftentimes it is necessary to emulate these devices, where doing so provides a virtual hardware set that the guest VM can interact with using a specialized driver whereby the hypervisor translates requests to the underlying physical hardware.

Emulated devices are often the most common method for device interaction with VMs, and can be a critical component in virtualization. Emulated drivers create a feature set in software, where all I/O requests are intercepted by the hypervisor and emulated on the real underlying hardware. Often times, the emulated hardware provided to the guest OS within a VM is older or more generalized than the given architecture set. This is due to the fact that the effort for constructing emulated devices is large, and the emulation of older hardware often helps with overall compatibility. However, this emulation process can often be slow and lead to significant overhead when utilizing the underlying hardware, not to mention the lack of newer features provided by more recent hardware.

Para-virtualized devices are specially tuned device software implementations of hardware where para-virtualized device drivers are installed in a guest VM that operate on a particular I/O API. While this leads to improvements in performance over emulation methods, it requires the guest to be modified in order to communicate ef-

fectively with the hypervisor. With Xen, the entire guest OS can be para-virtualized, whereas in other solutions such as KVM using `virtio`, device drivers can be para-virtualized. The performance enhancement of para-virtualization is derived from the removal of the hardware compatibility that must be in place for emulated drivers. At the cost of compatibility, it para-virtualization uses a tuned and customized API specific to the hardware at hand as an alternative.

A recent method for guest device interaction arrives out of the use of direct I/O device passthrough, whereby the hypervisor (and controlling host OS) relinquish the entire control of a given device to the guest VM. This allows for the guest to have direct interaction with the physical hardware, removing the need for complicated para-virtualized methods or slow emulated hardware. As the Peripheral Component Interconnect (PCI) bus and the updated PCI-Express bus are the most common hardware interfaces for devices on modern CPU architectures, this method often viewed as PCI passthrough.

An I/O Memory Management Unit is responsible for managing the connection of direct memory access (DMA) capable hardware along an I/O bus to main memory. Just like a CPU MMU, the I/O MMU maps device-visible memory addresses to physical memory addresses. However, utilizing DMA-capable drivers within a guest directly without I/O MMU virtualization technology would result in the guest attempting to perform DMA operations on incorrect guest physical addresses that would not map to proper machine addresses. In order to safely and securely enable PCI passthrough in virtualized settings, such CPU architecture mechanisms are needed.

As the popularity of virtualization increased, CPU architectures have met this demand with I/O MMU virtualization extensions. Intel and AMD have introduced VT-d and AMD-Vi (also named IOMMU) processor support for such operations. These

extensions provide the necessary mechanisms to isolate and restrict I/O devices specific to their owned partition space [233] through the use of I/O device assignment, DMA remapping, interrupt remapping, interrupt posting, and error reporting. DMA remapping ensures device DMAs not only find correct virtual memory addresses, but also act on only those memory addresses that are allowed, which provides the necessary VM isolation. Thus, hardware DMA remapping enables direct device assignment to VMs without device-specific knowledge in the hypervisor.

Using these IOMMU virtualization techniques for direct I/O passthrough with various hypervisors depends greatly on the hypervisor at hand. First, the BIOS must have IOMMU virtualization enabled and the OS kernel must initialize such hardware extensions at boot. This initialization can be done with the `intel_iommu=on` Linux kernel parameter for all Intel VT-d enabled architectures. With Xen, a specialized device driver called `xen-pciback` is used to "grab" the PCIE device on boot to keep the device state uninitialized and ready to be passed through to a VM. This is described in more detail in Section 4. With KVM, a similar method is utilized, whereby either `pci-stub` or `vfio-pci` (the former for pre-3.9 Linux kernels, the latter for current Linux systems) is utilized to bind to the PCI device on boot before the kernel or other add-on modules attempt to initiate the device. Either method is built into the kernel and any device drivers must be blacklisted so as to ensure proper ordering. `vfio-pci`, similar to `xen-pciback`, takes the PCI device ID as a parameter to determine which device to bind to (this can be found from the output of `lspci`), and holds all device initiation until the device is handed to a booting guest.

Generally, PCI passthrough can be utilized for most PCI devices, however GPUs devices can require some added configuration and considerations. This is largely due to the fact that GPUs are VGA devices, which represent a special case. Specifically, VGA devices to be used for PCI passthrough must not be the primary VGA displays.

Second, the VGA BIOS has to be loaded by the guest VM before actual BIOS boot. This can be done using a specific emulated BIOS, which with KVM can be either a modified SeaBIOS or a EUFI-enabled Open Virtual Machine Firmware (OVMF) configuration. For Nvidia devices, only approved devices (often Tesla and QUADRO adapters) can be used for GPU passthrough due to proprietary VGA BIOS configurations. Of further note, some GPU devices come with attached PCI Bridges due to packaging and compatability reasons. This includes add-on GPU servers, such as the Nvidia S2050, or dual-GPU units, such as the Nvidia Tesla K80 GPU. Other devices may be packaged with such PCI Bridges for including onboard sound controllers too. However, these PCI Bridges handle PCIE connections to the devices and yet are often in separate IOMMU domains, which causes GPU passthrough to fail due to Access Controller Services (ACS) errors in the IOMMU hardware. While there are new kernel patches coming available to override the ACS mechanisms with KVM, this obviously provides a significant security vulnerability that may be problematic for deployments outside of the academic realm.

In Chapters 5 and 6, KVM's PCI passthrough is detailed for use with Nvidia GPUs, where we find that this method can, with NUMA placement of VMs and newer hardware on Sandy Bridge architectures, perform at near-native speeds. Without GPU passthrough, GPU usage within a virtualized domain was either not possible, or required the use of front-end API solutions. As discussed in Section 4.3 of this dissertation, such remote API methods are suboptimal due to performance considerations and the lack of full-feature support. Some of the best methods developed thus far for utilizing remote GPUs in a virtualized architecture come from the rCUDA project [56], which sends CUDA commands across an interconnect to a remote device. However, this approach is fundamentally limited by the interconnect itself. Using data from the rCUDA [234] project, we compare GPU passthrough performance of both

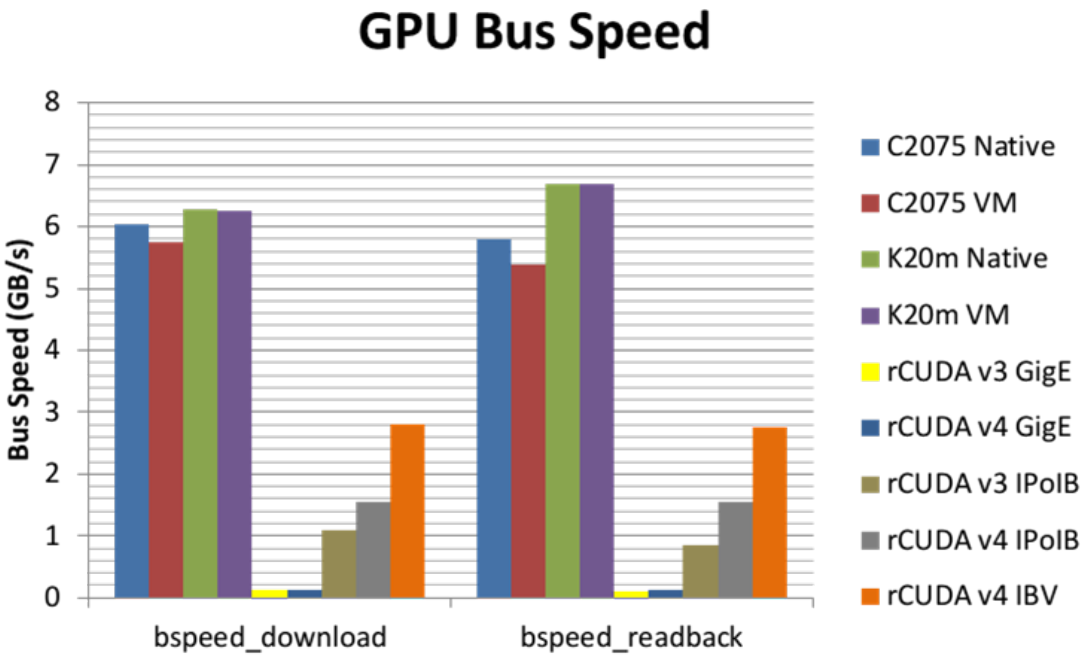


Figure 7.1 Comparison of GPU Passthrough in Xen to rCUDA across various interconnects

C2075 and K20 cards in Xen to various interconnects reliant on rCUDA. In Figure 7.1, it is illustrated that even in the best case with high speed InfiniBand, rCUDA is still limited by the interconnect fabric, which can only operate as fast as the same PCIE bus to which the GPU is attached. In reality, even the top-end IB adapters are not capable of saturating full 16x PCIE lanes. Furthermore, all GPU data transfers saturate the interconnect, leaving no available bandwidth for communication operations as found with many distributed memory HPC applications. Sockets and shared memory approaches found in API-remoting methods, such as gVirtus [176], suffer even worse performance impacts due to the necessity to buffer memory, as seen in related research [235]. In summary, these front-end API methods are suboptimal in comparison to GPU passthrough, as the transfer time between CPU and GPU memory often can have a drastic impact on overall application performance.

While PCI passthrough works well for providing dedicated accelerator resources such as GPUs, a sharing model of PCI passthrough does not hold. This is because PCI passthrough is a simple 1-to-1 relationship between a guest and PCI device. While it is possible to have a 1-to-many relationship (e.g. a single VM with multiple GPUs connected), there is no way to share a single device across multiple guests or the guest and the host. While this is not an issue and in fact a desirable effect with GPUs (GPUs are not designed for multi-application sharing and such a solution would largely lead to significant inefficiencies), high speed networking adapters attached on a PCIE bus do have a necessity to be shared in virtualized environments.

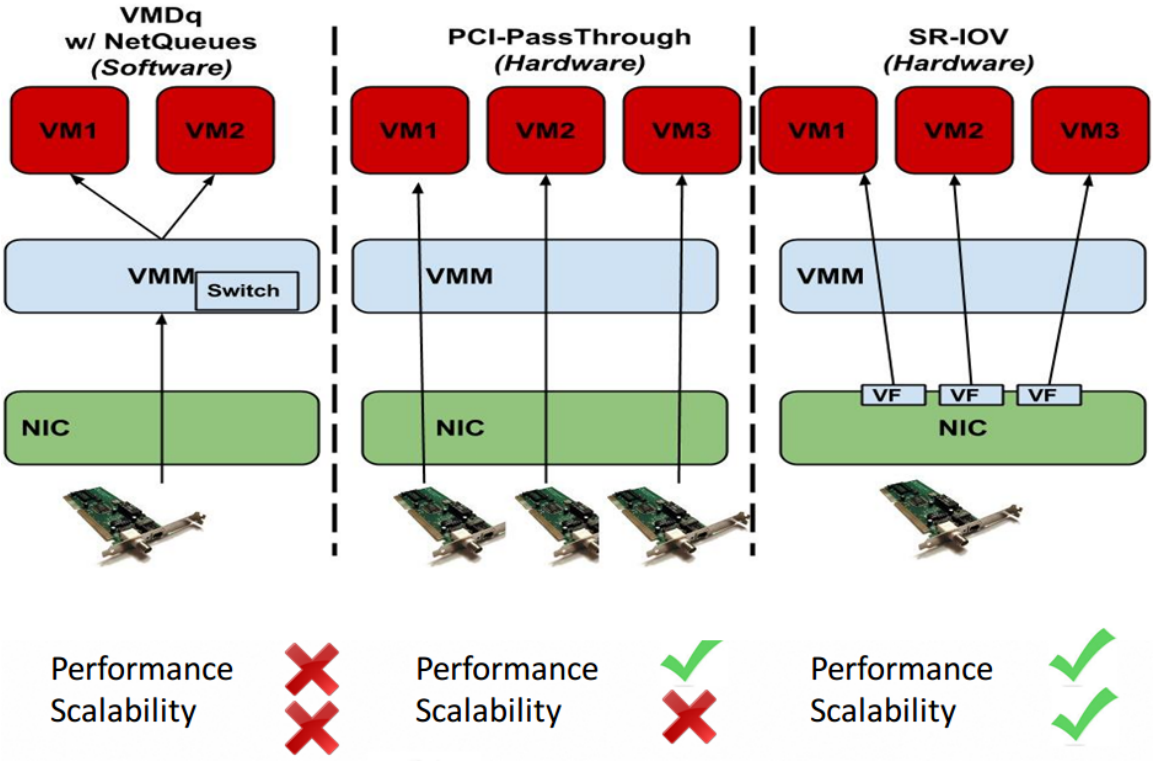


Figure 7.2 Efficient VM networking comparison [208]

With this requirement for sharing PCI networking adapters, a few options exist and are illustrated in Figure 7.2. The first option is to use software multiplexing that sorts inbound and outbound traffic and forwards packets to queues set up for each

VM as necessary, as implemented with Virtual Machine Device Queues (VMDq) [236]. While this solution of software queues has a workable Ethernet solution and allows VMs to share a PCI based NIC, there are performance limitations. This is due to the large amount of hypervisor involvement necessary in sorting the queues, as well as the fact that all data is buffered between the hypervisor and guest memory. The next and simplest option is simply to add more PCI adapters to a given server. While multiple PCI passthrough does work, there is a fundamental limitation to this method, as the number of cores and NUMA sockets is expanding faster than the number of PCI lanes available per socket.

With these limitations in mind, Single Root I/O Virtualization (SR-IOV) has been developed. This standardization effectively enables multiplexing of a PCI based communications adapter (typically Ethernet but also InfiniBand adapters) within the hardware. With SR-IOV, multiple Virtual Functions (VFs) are created and configured in hardware. Each VF has a dedicated resource pool with specific Tx and Rx queues, along with other lightweight PCI resources such as device registers, base address registers, and hardware descriptors. Adapters also maintain a Physical Function (PF), which is a fully-implemented PCI device that acts not only as a standard controller card, but also as a control mechanism for the VFs (given firmware adjustments). These mechanisms are illustrated in Figure 7.3 provided by Intel [237].

To use SR-IOV within a virtualized setting, a VF is given to a guest VM on boot. This happens using the PCI passthrough mechanisms, except the specific VF PCI identifier is used instead of the actual adapter card. Within the guest, the VM loads a specific driver (usually supplied by the hardware vendor) that is able to probe and detect VF functionality. This driver fills in the descriptors and sets up memory allocations for DMA directly within the guest OS, effectively creating dedicated data queues within the VM.

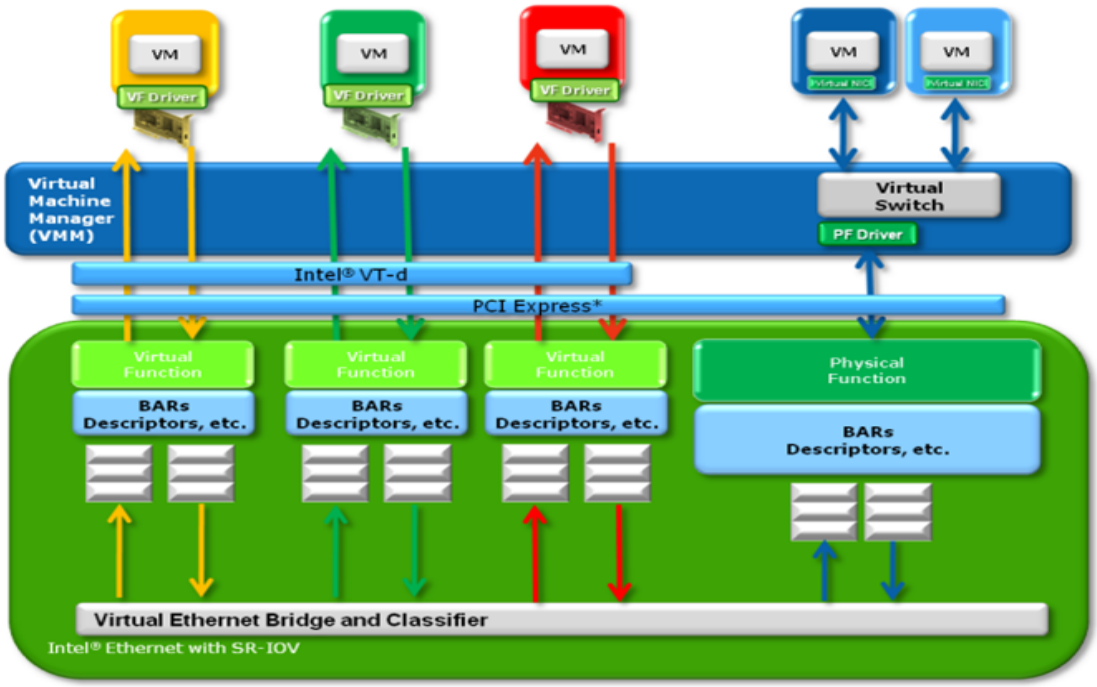


Figure 7.3 SR-IOV Architecture with Virtual Functions, PCI SIG [237]

When a datum (could be a packet or a message, depending on the adapter type) arrives at the physical card, it is sent through a hardware switch, which places the data into a pool specific to the target VF. The data is then immediately DMAed to the guest based on the preallocated memory buffers. This is possible through the use of VT-d mechanisms which enable the translation between device addresses and the guest virtual addresses, bypassing the need for hypervisor memory translation. This entire process happens without any CPU interaction, other than occasional interrupts to provide notification of completed data work queues. This is essentially enabling direct hardware DMA transfer to a guest without any hypervisor involvement.

Using modern 10GbE and InfiniBand adapters, SR-IOV can easily configurable to have up to 64 VFs, enabling a great deal of scalability within a given host. Without any hypervisor interaction or context switching, bandwidth and latency can start to approach near-native levels, which was previously not possible with other methods.

Looking at InfiniBand, research has found that Mellanox adapters with SR-IOV in KVM can achieve near-native bandwidth while incurring only a slight 15-30% overhead on small (1 byte) messages [207, 208]. This latency overhead is due to the extra hardware switching that has to happen between the VFs in the adapter itself, as well as the IOMMU involvement.

7.2 Memory Page Table Optimizations

As we have seen both in Chapter 3, as well as in other supported literature [48], virtualization of memory structures is a point of contention and potential overhead. This is often due to the extensive effort a hypervisor has to perform in order to translate memory addresses from guest-virtual addresses to host-virtual addresses, and then again to machine-physical addresses. Whle this is a necessary function of virtualization and a hypervisor, there are various methods developed both in hardware and software to provide such address translation functionality, each with their own advantages and disadvantages.

Modern computing systems provide a mapping of virtual memory address space to physical machine memory. This memory management technique allows processes to have independent virual address spaces, which provides security and process isolation. Today's x86 CPUs, as well as many other CPU architectures, have a hardware memory management unit (MMU) that provides translation abilities within hardware that often have page table entries (PTE) for storing virtual to machine memory mappings, and a translation lookaside buffer (TLB) that provides an effective cache for the most commonly used virtual addresses. Specifically, x86 page tables are walked iteratively in hardware with their layout specified by the x86 hardware specification where the CR3 register holds the page table base and a 4-level radix tree structure represents the page table hierarchy for 4KB pages.

With virtual machines, a 2-level address translation is needed where guest virtual memory is translated to guest physical memory and then again to physical machine memory. This direct two-level memory mapping is classically handled by the hypervisor, as the guest cannot access machine memory directly. The hypervisor functions in software without hardware support, so it can be expensive for it to update and maintain guest memory translation. With x86 virtualization, memory virtualization is handled using shadow page tables [238]. Shadow page tables eliminate the need for emulation of physical memory inside the VM by creating a page table mapping from guest virtual to machine memory. However, these page tables are not walkable by hardware like a TLB and, as such, guest OS page tables require updating of the shadow page table by the hypervisor. This can be costly not only in the additional management, but also by the cost of VMexit and VMentry calls, which are known to add thousands of CPU cycles of overhead for each call. If there is a memory bound application, continual shadow page table management by the hyperivsor can add a notable overhead, impacting overall application performance.

Recently, Intel and AMD have implemented Extended Page Tables (EPT) and nested paging (NPT), respectively, to cope with the issues of shadow page tables. With nested paging, a guest page table converts guest virtual addresses to get physical addresses, and another second level table converts guest physical addresses to machine addresses. Each address translation in guest mode requires a 2D page walk where the guest page table is traversed and each guest physical address requires a second level page table walk to obtain the machine address. This support means the TLB hardware is able to keep track of both guest pages and hypervisor pages concurrently, effectively removing the need for shadow page tables and resulting hypervisor intervention entirely. Nested paging provides a simple design with no necessary hypervisor traps leading to less overal overhead and swapping, less TLB flushes, and a

reduced memory footprint.

The downside of nested paging for virtual machines occurs when there is a TLB miss. A TLB miss is when a requested page is not in the TLB, and in such cases the cost of a TLB miss is substantially higher in a guest VM. Natively, a TLB miss requires a walk of 4 address entries for 4KB pages. However, in a guest, each of those 4 address entires require another second level walk to find the system physical address for each guest page entry, plus a final nested page walk to translate the final guest physical address to a usable machine address.

The TLB miss can be illustrated in the efforts in Figure 7.4 from Bhargava et. al [239], whereby we can see the steps for handling a TLB miss on an AMD CPU in both native (left) and virtualized (right) modes. With 2D nested or extended page tabling given in Figure 7.4, each `gLn` entry cannot be directly read using a guest physical address, and, as such, the nested page table walk is necessary to translate the guest physical address before the entry can be read. This has to happen recursively for each level, and in this example with 4KB pages, that includes 4 levels. Lastly, a final nested page walk is required to translate the guest physical address of the data to a physical machine address.

The cost of a 2D page table can be evaluated in the number of references, and is easily calculated. If a guest page walk has `n` levels and a nested page walk has `m` levels, the virtualized 2D walk has a cost calculated by:

```
nm + n + m
```

For 4KB pages, this requires 24 references in a virtual TLB miss (4 page waks and 4 nested page walks), compared to a cost of just 4 references for a single TLB walk natively, as indicated in Figure 7.4. While many applications illustrate this TLB

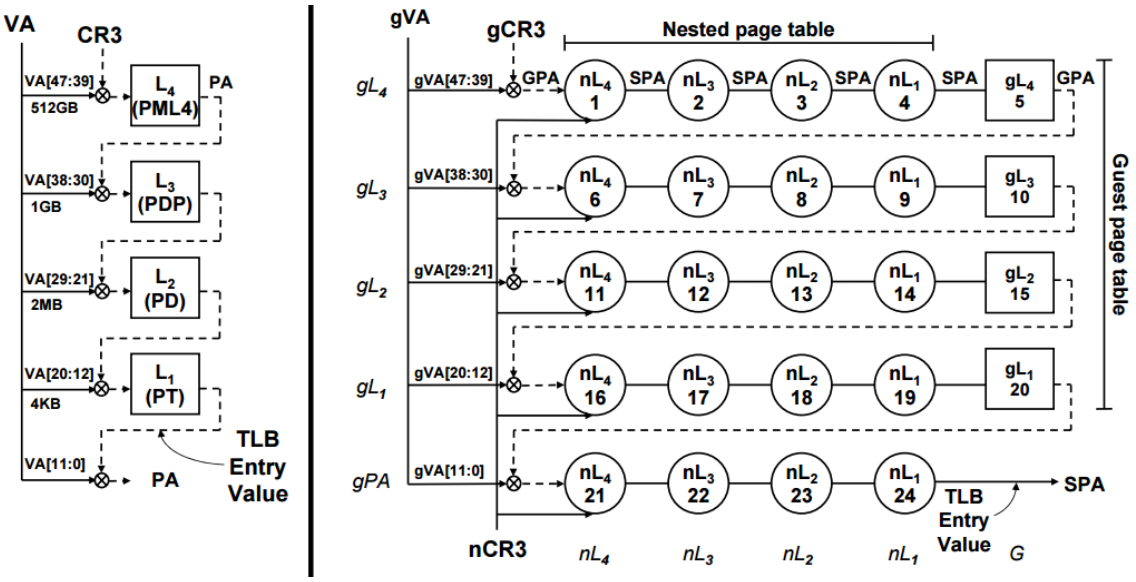


Figure 7.4 Native TLB miss walk compared with 2D virtualized TLB miss walk from [239]. On the left illustrates a native page table walk. On the right illustrates the lengthy 2D nested page table walk for a VM.

miss cost is much less than that of managing shadow page tables, it can still lead to a significant gap in performance between non-virtualized applications, especially as VM count or an application's memory footprint increases.

One potential way to decrease the chance of a TLB miss (and therefore the cost of a miss) is to use a larger page size. By default, x86 hardware uses 4KB pages sizes, but newer hardware can support 2M and 1G page sizes as well, effectively named *transparent huge pages* or THP. Using the KVM hypervisor with transparent huge pages enabled, we can create guest VMs backed entirely by 2M huge pages [201]. We can also enable transparent huge page support within the guest, as well, to have the entire guest OS (including kernel and modules) backed by 2M pages.

The result of THP-enabled guest VMs, along with associated THP in the host to back the guest huge pages can be significant. With huge pages on Intel x86 CPUs with EPT, there exists an entirely separate TLB for huge pages. This will naturally alleviate TLB pressure and therefore reduce TLB contention between guest and host

operating systems (whereby the host is still utilizing 4KB paging for kernel space). Most importantly, the TLB reach is increased, because 2MB pages cover a larger addressable memory space. The size of the page tables themselves also decreases with the use of huge pages.

If huge pages are used in the host, there are then 3 levels of a page table walk, and if huge pages are used in the guest, there is also only a nested page walk of 3 levels. Using the formula, if THP is enabled in the host, there is a TLB miss cost of 19 references. If huge page support is enabled in both the guest and the host, this drops to only 15 references. While this is still more than the native miss cost of 4 or 3 references (for 4KB and 2MB pages, respectively), huge pages provide a 37.5% reduction in the TLB miss cost compared to 4KB pages, still significantly better than the VMexit/entry costs associated with shadow page tables. While it would be possible to only enable huge pages within the guest, this would represent a suboptimal configuration. Here, 2MB pages will be splintered into 4KB pages, thus negating most performance benefits [240]. Most importantly, the overall reduction of the number of TLB misses due to the increased TLB reach with huge pages can have a major impact on application performance.

To evaluate the effect of 2M transparent huge pages on guest performance, we leverage the same KVM setup in Chapter 5 on the Bespin hardware. Specifically, THP is enabled both in the host as well as the guest OS kernels, and the same LibSVM application using GPUs is re-run. The libSVM GPU application can have significant memory requirements, as large chunks of the SVM datasets are stored in CPU memory and transferred in a sudo-random order to and from GPU memory, making it an ideal application to use for evaluating THP.

The results of running libSVM in a THP-enabled VM, a VM with no THP, and natively without virtualization are all outlined in Figure 7.5. Comparing first just

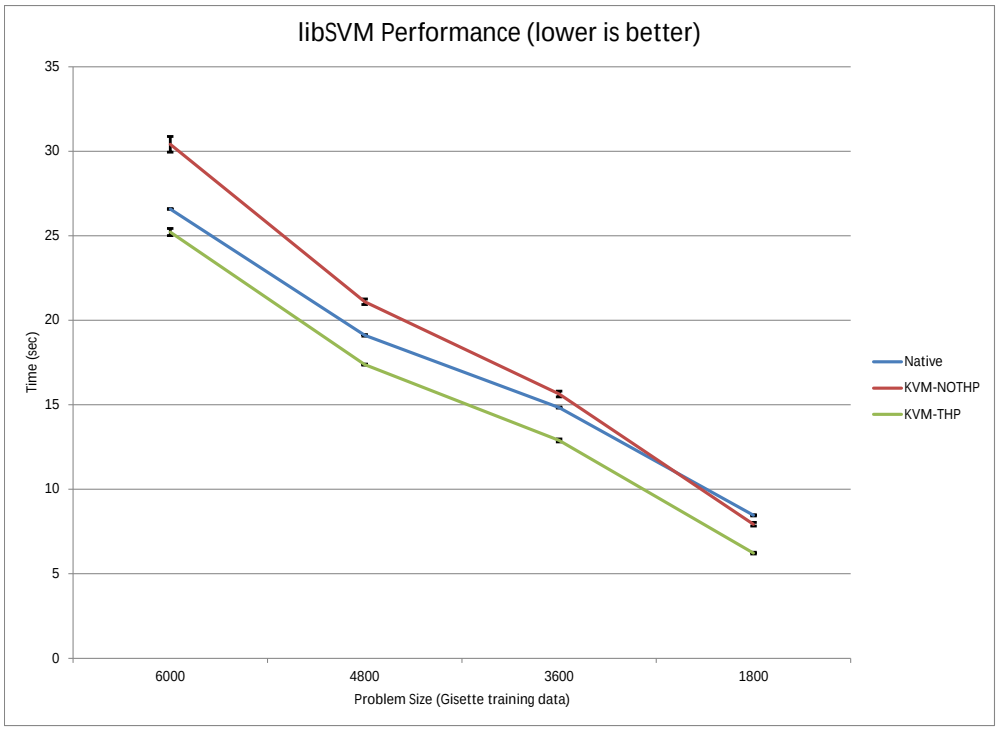


Figure 7.5 Transparent Huge Pages with KVM

the KVM results without THP to the native solution, we can see the impact of THP on the overall application runtime, especially at larger problem sizes (6000 training sets). However, when TLB is enabled in the guest and host, we actually see the KVM VM solution *outperform* the native solution. This is because guest privileged OS memory used to buffer to/from GPU memory is backed by 2MB pages in kernel space, instead of the normal 4k pages, as in the native solution (which has to use 4KB pages). This means data transfers can take advantage of a larger TLB reach, resulting in improved performance. While this is likely a special case for THP usage with the libSVM application due to large linear memory access patterns, the fact that a VM can even occasionally outperform a native runtime is a noteworthy accomplishment.

This also underscores the need for careful tuning and best-practices for hypervisors when supporting advanced scientific tasks where huge page support in both the guest and host environments as a key aspect, especially as the use of big data scientific applications increases.

7.3 Time in Virtual Machines

The measurement of time has been a longstanding consideration within Distributed Systems for many years. Time synchronization has often been at the forefront of that effort, with Lamport clocks [241] often used to illustrate a classic example of how critically important time is within such large scale systems. However, even simply keeping track of time and receiving fine grained resolution can often be difficult.

Within modern CPUs, time is measured a few different ways, each with their own advantages and disadvantages. Classically, time was measured in clock cycles for small operations typically within a OS kernel using a real-time clock (RTC) where an operation's time is measured by simply reading the number of clock cycles before and after an operation. This worked well for timing tasks where actual wall clock time was not needed, but simply a metric to compare operation length between implementations. However, with CPU improvements came dynamic voltage and frequency scaling as well as CPU sleep states, which could and often would confound clock cycle measurements. If a CPU entered a sleep state, it was not guaranteed that the cycle counter would continue to be updated.

Recent x86 CPUs now use a Time Stamp Counter (TSC), first introduced with the Pentium processor. The TSC is incremented at a fixed clock rate, independent of actual frequency and is not halted. The TSC essentially provides time sensitive information within hardware, rather than software, leading to fine granularity. Within Linux, if `constant_tsc` and `nonstop_tsc` are specified, the TSC can generally be

relied upon for accurate measurements independent of frequency or sleep states. In such a case, the TSC can be essentially utilized as a clock source. Reading of the TSC is done through the `rdtsc` instruction, itself generally taking about 24 clock cycles to complete, meaning it is a relatively fine-grained operation for time measurement.

There is also a High Precision Event Timer (HPET) that is used in some x86 systems for the past decade. This provides nanosecond granularity and utilizes multiple hardware sources to provide an accurate time measurement. As such, HPET was a desirable tool for time measurement before an invariant TSC was implemented. Furthermore, HPET is synchronous across multiple cores, whereas a TSC was not until very recently. Disappointingly, recent many-core architectures such as the Intel Knights Landing standalone CPUs still do not provide a core synchronized TSC, and as such the HPET may be more desirable to use for multi-core measurements. However, the overhead for HPET can be much larger than TSC, and implementations vary significantly on the performance, leading to some problems for very fine-grained measurements.

While clock timing and measurements on bare-metal OS and services is now relatively straightforward, virtualized time is not always as straightforward. This is because as the hypervisor virtualizes a given CPU, it also has to provide some virtualized notion of such time counters and clocks to the virtual machines as well. Until recently, many hypervisors virtualized such timing systems in many different ways.

For instance, VMWare provides a virtual TSC by default. This virtual TSC was particularly useful when multi-core TSC were not reliable and there was a need for timing within VMWare VMs. However, there are a number of problems with VMWare's virtual TSC. The VMWare hypervisor has to perform work to virtualize the TSC for each read. For fine-grained measurements that are in close proximity, this can cause a backlog in the hypervisor, and the TSC can fall behind real time.

In effect, for sufficiently small measurements VMWare can report better than native performance when benchmarking. However, this is actually an artifact of VMWare's virtual TSC, not something that is consistent with real time and as such VMWare cannot be relied upon for fine-grained timing experiments. This problem has lead to the removal for VMWare results throughout this manuscript as VMWare. Very recently, the use of x86 invariant TSC by other hypervisors has lead VMWare to try to address the issues with its virtual TSC by providing a mechanism within ESXi to forcibly disable the virtual TSC. However, the results reported within this dissertation have not looked to reevaluate to confirm this method in fact is now reliable.

While VMWare's virtual TSC is problematic, other hypervisors such as KVM have to also wrangle the same situation that lead to VMWare's design of the virtual TSC. However KVM takes a much more simple approach of providing the TSC directly to the guest VM. Intel's VMX instruction set provides conditional trapping of RDTSC, RDMSR, WRMSR and RDTSCP instructions, which is enough for full virtualization of TSC in any manner. VMX also allows for a offset field, to allow for the guest to synchronize TSC of multiple vCPUs. These mechanisms allow KVM to create a simple mapping between guest and host TSC, even if the TSC can be written by either. However as noted earlier with real-time clocks and TSCs on bare metal, the TSCs can get out of sync when power-saving modes are applied, such as with frequency states. As such, KVM best practices include the disabling of all power-saving features to ensure the TSC remains constant and reliable. With `constant_tsc` utilized, this situation is no longer a problem in newer CPU architectures. For all experiments in this dissertation, power saving modes were disabled in the host and guest to ensure no invalid results.

7.4 Live Migration Mechanisms

Migration of VMs represents one of the fundamental advantages to virtualization, and also one of the greatest challenges to efficiency. With VM migration, the complete VM state is copied from a source to an unallocated destination host, where disk, memory, and network connections are kept intact. For disk continuity, a distributed and/or shared filesystem is utilized, most commonly but not exclusively NFS, where both the source and destination hosts have access to the VM disk. Network continuity is preserved as long as the destination guest is within the same LAN and generates an unsolicited ARP reply to maintain the original IP after migration. VM vCPU states and machine states are recorded from the source and are quickly sent to the destination host when the VM is paused. For live migration, the source VM is paused only after all state and memory contents are copied to the destination. The last of the dirtied memory pages are copied over, and the newly formed destination VM is then un-paused. This pause and transfer time represents the entirety of a VM downtime during live migration, and is often at or under 100 milliseconds across commodity Ethernet networks (given a VM with low memory utilization).

The memory transfer stages are often the main performance consideration for overhead during live migration. This is not only due to the potentially large amount of memory to be sent across the network, but also the veracity at which the memory is changed. This is defined directly by the amount of main memory allocated (or in use) by the source VM. However, as a VM's memory is sent, the VM is still running and therefor memory pages can be dirtied, creating the need for any written page to be retransmitted. Given a small network and memory bound processes running within a VM, this can be an infinitely long process of page dirtying. Many live migration strategies provide an iterative timeout mechanism to avoid this infinite state, but this

will lead to increased downtime during migration.

The copying of memory pages for live migration can take multiple implementations. Three common options are summarized below:

- **Pre-copy Migration -** All memory pages are transmitted to the destination before the VM is paused. The hypervisor will note and track all dirtied memory pages, and retransmit those pages in iterative rounds. The rounds end when either no dirtied pages exist or a max iteration count has been reached. The VM state is then transmitted and resumed on the destination. This method was the first live migration technique used in Clark et al [242] and is by far the most common.

- **Post-copy Migration -** The VM state is paused and sent to the destination hypervisor, and immediately resumed. If the new destination VM generates a page fault, the VM is paused, and faulted pages are transmitted across the network on demand from the source and the VM resumed. This methodology is proposed for use in the Xen hypervisor by Hines et al [243].

- **Hybrid-copy Migration -** Provides a compromise solution to memory paging. First, a single copy of the VM memory pages, or a subset of known-necessary memory pages, are sent to the destination. Then the source VM is paused, its VM state sent to the destination, and resumed on the destination. Known dirtied source pages, or missing pages, are then copied to the destination upon a triggered page fault utilizing the same mechanism as post-copy migration. An example of hybrid migration can be found via Lu et al [244].

While pre-copy migration is the traditional and most used live migration technique, there will soon be opportunities to implement other migration techniques to

advance the mobility of distributed computing in high performance virtual clusters with virtualization.

7.4.1 RDMA-enabled VM Migration

Currently, most live migration in production environments occurs over TCP/IP connections due to the prevalence of commodity Ethernet connections within cloud infrastructure. However, even if RDMA-capable interconnects are available in such infrastructure as described with InfiniBand in Chapter 6, live migration still usually occurs over TCP/IP. For the case of InfiniBand, this is via IP over InfiniBand (IPoIB) [245], which can be an inefficient use of the interconnect bandwidth and add extra latency [246].

The time it takes to migrate the memory contents from a source to destination VM can be significantly decreased by using an RDMA based mechanisms. Huang et al first provided a proposed pre-copy method for RDMA-based migration using the Xen hypervisor [247]. Specifically, they found an 80% decrease in migration time with RDMAwrite operations. This speedup is largely due to the removal of overhead necessary for processing TCP/IP communications, largely in CPU utilization for copying buffers, packet processing, and the included context switch overhead when competing for resources in a CPU-bound application (which are common in HPC environments).

The live migration algorithm proposed in [247] uses the standard pre-copy mechanism that sends the entire memory contents across the network as RDMA operations, then iteratively copies dirtied pages before switching the running states. As discussed in the previous section, a post-copy migration strategy may have benefits for quick VM migration in high performance virtual clusters, or even for VM cloning as described later in Section 7.5. Furthermore, efforts in Chapter 3 have found that the Xen hypervisor is not best suited for HPC workloads. As such, there is a need to

redefine the use of RDMA for VM migration using a hybrid post-copy mechanism in a high performance hypervisor. This post-copy migration mechanism is provided defined:

- Transfer initial CPU state, registers
- Start the page table pages translation process: MFN to PFN and use copy-base approach
- Concurrently, allocate remote memory on destination VM.
- Set up other machine state settings in destination
- Start destination VM, pause source VM.
- Initiate RDMAwrite of entire memory contents from source to destination.
- As page faults occur in destination VM, catch faults and perform RDMAread requesting pages.

Currently efforts are underway to provide post-copy live migration in KVM/QEMU, using the `migrate_set_capability x-postcopy-ram on` mode within KVM [248]. This method uses the Linux *userfaultfd* kernel mechanisms from a kernel 4.3 or newer. At the start, all memory blocks are registered as *userfaultfd*, so all faults cause the running thread to pause. In kernel space, the missing page is requested from the sender, which is prioritized over other pages being sent and is returned and mapped to the destination guest memory space and the thread or process is un-paused. This mechanism operates asynchronously, so that multiple outstanding page faults will not stop other executables within the VM. The `xpostcopy-ram` extension has been identified as an opportune place to implement an RDMA based implementation within KVM.

To provide RDMA functionality, the InfiniBand interconnect could first be used as a proof-of-concept. This choice is due to InfiniBand's rise in popularity, increased prevalence in virtualized systems with SR-IOV, as noted in Chapter 6, and RMDA functionality. However, other interconnect options exist that may be better suited for enhanced functionality and performance, such as Intel's new Omnipath [249], or an Ethernet solution such as RoCE [250], to name a few. While RDMA can be managed through the kernel level, it is best used in user-level APIs, such as MPI, or in the case of InfiniBand, ibVerbs. However, it may be ideal to select a interconnect-agnostic middleware for RDMA implementations to add future support for other interconnects, such as Photon [251].

RDMA semantics can be used to either read or write contents of remote memory, in this case VM guest memory pages. However before such operations can take place, the target side of the operation must register the remote memory buffers and send the remote key to the initiator, effectively providing the DMA addressing to be used. Beyond the increased bandwidth and decreased latency benefits provided by an advanced interconnect with InfiniBand, RDMA also allows VM memory to be sent without involving the OS. This is due to InfiniBand's zero-copy, kernel bypass mechanisms, and asynchronous operations. This keeps the CPU load down, allowing for the CPU to spend time on the computation at hand rather than the I/O transfer, as it often has to when utilizing a TCP/IP stack.

Selecting the proper RMDA based communication operations to use can make a notable difference in the overall performance of the migration. Some RDMA operations are largely a one-sided involvement, which can have performance impact and should be carefully considered for use in post-copy live migration. The RDMAread operation requires more effort at the destination host, while RDMAwrite operation puts burden on the source host. One way to determine which method is best is by

evaluating both source and sink CPU loads. However, this method likely will not be as obvious when we consider VMs will be not be running in an over-subscribed virtualized environment, but rather a highly optimized one.

When the destination VM page faults, it is necessary to retrieve the page with as little latency as possible, as the running thread is paused. This is one of the advantages of using RDMA, but implementation details can also effect performance. Using the method developed in [248], the *userfaultfd* will trigger an RDMAread to retrieve the missing page. RDMAread requires no interaction from the source VM, as RDMAread is one sided and the memory buffers have been passed in the setup phase. To further enable quick return time for the missing page, enabling polling on the source host may further reduce latency, however verification of this will be necessary. When the RDMAread operation is finished, the running thread will resume and execution will continue.

7.4.2 Moving the Compute to the Data

While pre-copy migration is dominant in the live migration techniques of nearly every mainstream hypervisor today, the proposed post-copy migration could provide some key new advances for HPC and big data applications. One particular use case would be to send a lightweight VM to act directly on a large or set of large datasets and return a slimmed down result set. This would reduce the requirement of transmitting the data across a network entirely, and could potentially speed up data access latency drastically, as the returning information is transmitted in the form of memory pages and VM state. Essentially, the proposal is to send the compute to the data, instead of visa versa.

With post-copy migration, one could move the computation at hand close to a data source in significantly less time than full pre-copy live migration. This data

source, and lightweight VM sink, could potentially be something similar to a Burst Buffer system [252,253] or a classic HPC I/O node with a distribute filesystem such as Lustre or GPFS [254]. This data source could even potentially be a remote scientific instrument completely separate from the HPC infrastructure itself, especially if the network at hand is capable of RoCE [250] or iWARP [255]. A VM would initiate post-copy live migration, transmitting only the necessary CPU state, registers, and non-paged memory, rather than the full VM memory state. Once migrated, the VM could connect to a (now local) I/O or storage device, accessing data fast and performing the necessary calculations. The VM could potentially even forgo the copy of the majority of its memory. During this time, only the necessary memory pages required to complete the immediate calculation would generate a fault and trigger their transmission from the source. The VM could even return the result (rather than a very large dataset) to the original source VM.

This post-copy live migration technique for remote data computation avoids the cost of spawning a whole new job and/or process with associated running parameters, as well as the extremely high cost of a full VM live migration using the pre-copy method. However, one potential downside of post-copy live migration would be the non-deterministic runtime, as it would be unknown how much remote memory paging would be required. This could lead to more time spent with the destination VM in a paused state awaiting remote memory pages, rather than if the entire VM memory contents were transmitted completely. Careful analysis of memory usage, or a hybrid copy method based on predetermined memory sections, could help overcome this issue, but may require a more advanced migration architecture.

7.5 Fast VM Cloning

In many distributed system environments, concurrency is achieved through the use of homogeneous compute nodes that handle the bulk of the computational load in parallel. This can take many forms, including master/slave configurations, or even a traditional HPC cluster with identical compute nodes, as are often used to support Single Process Multiple Data (SPMD) computational models [256]. With high performance virtual clusters, there is a need to efficiently deploy and manage near identical virtual machines for distributed computation.

In Snowflock [257,258], the notion of VM cloning is given. Specifically, Lagar et al. define the notion of VM Fork, where VMs are treated similarly to a fork system call for processes. This process is conceptually similar to VM migration, with the exception being that the source VM is not destroyed after the migration. Starting with a master VM, an impromptu cluster can be created across a network using the Xen hypervisor. Snowflock specifically uses Multicast to linearly scale out VM creation to many hosts, only coping a minimal state and then remotely coping memory pages when requested. This fetched memory on-demand is similar in principal to the post-copy live migration technique described in the previous section. This is further augmented with blocktap-based virtual disks with Copy-on-Write (CoW) functionally, delivering CoW slices for each child VM.

While Snowflock provides an excellent framework for VM cloning, it is not suitable for the current implementation. First, it uses Xen, which in previous research described in Chapter 3 has been found to have limited performance for HPC workloads [172]. Second, SnowFlock is designed for Ethernet and IP based networks, which have significantly higher latency and lower bandwidth when compared to InfiniBand solutions. Developing analogous mechanisms, like what is listed below, with

a high performance hypvervisor such as KVM or Palacios [31] to use RDMA for VM memory paging could have an effect on the way in which virtual cluster environments are deployed.

- Prepare parent VM state, including registers and info.
- Prepare CoW disk images.
- Create large buffer for all VM memory on child hosts.
- Send vmstate via RDMAwrite or IB Send to N child nodes, where N is the clone size.
- Resume/start child VMs in tandem.
- Parent VM set up RDMA multicast (unreliable connection) and initiiate sending of memory pages to all child VMs
- Child clone VM joins RDMA multicast.
- If child page faults, perform RDMAread operation for specific page.

This method allows for efficient cloning of VMs based on a running parent VM. First, the VM state and images are copied to all child nodes to receive the VM, and blank memory is allocated. Then, each child VM is started, and page faults are handled via RDMA read. This will result in an initial slowdown, but as pages are received the child VMs will start to run. The rest of the memory is eventually sent via multicast to all VMs simultaneously. As multicast is unreliable, a delivery failure will just trigger a page fault and subsequent page retransmission via RDMA. It allows for direct page fault handling, while still allowing child VMs to start and run immediately. As RDMA multicast mechanisms are relatively questionable, more investigation is needed to evaluate the feasibility of this situation.

It is expected that post-copy VM cloning will work most efficiently if used in conjunction with guest VMs backed with huge pages. Transferring memory in 2M chunks will more effectively utilize network bandwidth by eliminating send/receive overhead. This also will hide the latency found in SR-IOV enabled interconnects for small messages. Furthermore, it will reduce the overhead of page fault handling mechanisms, as less overall pages will fault and be transferred. While huge pages are expected to improve VM cloning efficiency, empirical testing will still be necessary to properly evaluate their viability.

While a VM fork mechanism leveraging post-copy live migration in KVM will quickly spool up cloned VMs, the eventual memory transfer will eventually fail to scale past the network's capacity. This could happen if hundreds or thousands of child clone VMs are started simultaneously, as is likely in large scale deployments. As such, a hierarchical distribution may be necessary. One possible method for this would be a two-stage cloning mechanism, where child VMs are cloned one to each cabinet, and the entire memory contents copied using the pre-copy migration mechanism. From there, further cloning occurs to deploy many cloned VMs to individual nodes within the cabinet. Organization of mid-tier cloned VMs would likely be determined based on RDMA fabric configurations within cabinets, as this is a network-bound process.

7.6 Virtual Cluster Scheduling

Historically, cloud infrastructure has taken a simplistic approach when it comes to VM and workload scheduling. Often, round-robin or greedy [259] scheduling algorithms are naively applied. With round robin scheduling, a simple list of host machines are used and iterated over as VM requests are made. This essentially scatters the VMs without regard to their locality, and over-subscription becomes commonplace regardless of the number of requested VMs or their interconnection. A greedy algorithm can

help keep spacial locality, but focuses specifically on over-subscription as well to help consolidate VM allocations. While this over-subscription aspect is advantageous for public cloud providers such as Amazon EC2, it becomes counterproductive for high performance virtual clusters.

With high performance virtual clusters, VM instances that can gain near-native performance are needed. Furthermore, these VMs will be running tightly coupled applications, and, as such, must be allocated in a way in which communication latency is minimized and bandwidth is maximized for all VMs. This will help insure the entire virtual clusters can perform optimally. However, given off-the-shelf private cloud providers or, worse still, public cloud infrastructure, these requirements are at best opaque to the user, and at worst extremely suboptimal.

One way in which high performance virtual clusters can operate efficiently is by defining specific instance types, or *flavors* within OpenStack, that define what resources a VM has allocated. Given previous research on the NUMA effects of VMs [260], these specialized flavors should be defined to fit within a NUMA socket. Furthermore, CPU pinning should be used to specifically keep the VM within the NUMA socket itself. Using Libvirt, a common API utilized in many cloud infrastructure deployments (including OpenStack), we can specify CPU pinning directly in the XML configuration.

```
<cputune>
 <vcpupin vcpu="0" cpuset="0"/>
 <vcpupin vcpu="1" cpuset="1"/>
 <vcpupin vcpu="2" cpuset="4"/>
 <vcpupin vcpu="3" cpuset="5"/>
 <vcpupin vcpu="4" cpuset="6"/>
 <emulatorpin cpuset="2"/>
```

```
</cputune>
```

Figure 7.6 Adding extra specs to a VM flavor in OpenStack

With OpenStack, this NUMA configuration can be executed through the KVM Nova plugin and a scheduling filter, which defines how to place VMs effectively. Furthermore, the instance flavor can also specify the addition of `instance_type_extra_specs` within Nova, whereby specialized hardware such as GPUs and InfiniBand interconnects (as described in Chapters 5 and 6) can be passed through to the VMs directly. Once defined, the same specialized high performance flavors in OpenStack simply have to add the labels (such as 'gpu' or 'infiniband') to the flavor to gain the requested hardware, as illustrated in Figure 7.6 with OpenStack Horizon's UI interface. The implementation put forth in the OpenStack Havana build has since been upgraded by Intel with SR-IOV support and additional scheduling filter additions, and is available in the latest OpenStack releases [261].

```
pci_passthrough_devices=[{"label":"gpu","address":"0000:08:00.0"},
 {"label":"infiniband","address":"0000:21:00.0"}]
instance_type_extra_specs={'pci_passthrough:labels':'["gpu"]'}
instance_type_extra_specs={'pci_passthrough:labels':'["infiniband"]'}
```

To provide a space for high performance virtual clusters, we need a scheduling mechanism within a cloud infrastructure to support the effective and proper placement beyond controlling for NUMA characteristics. While there are many effective

scheduling algorithms for workload placement within an environment, a Proximity Scheduler [41], as defined in OpenStack, may work well. Specifically, one could either define or compute the underlying cloud infrastructure network topology. This could be as simple as a YAML file that defines an underlying InfiniBand interconnect 2-1 Fat Tree topology, or a more complex solution that utilizes network performance metrics to measure bandwidth and latency between disjoint nodes to build a weighted proximity graph. As it is possible that such network parameters could change due to other usage or to reconfiguration, a method of periodically monitoring and updating this proximity network using measurement tools such as PerfSonar [262] may also help keep an effective proximity metric between hosts. With a metric, one can then apply a proximity scheduler to handle high performance virtual cluster allocation requests effectively and in a way that will be far more optimal than a simple round robin scheduling mechanism. The OpenStack private cloud IaaS framework is proposed for this effort, however, further development is needed to bring such a scheduling mechanism to fruition.

The use of service level agreements (SLAs) within cloud infrastructure allocation is a well studied aspect [263]. It is also possible that certain SLAs could be incorporated into a private cloud infrastructure such as OpenStack to simultaneously guarantee performance for virtual clusters with the above defined methods while concurrently offering "classical" VM workload scheduling for HTC, big data, or other cloud usage models. This would provide the same user experience yet support diverse workloads and performance expectations as defined by a given SLA. As it is likely such performance-tuned cloud infrastructure deployments as proposed in this dissertation will likely still be used for traditional cloud workloads, it is advantageous to leverage SLAs in this way.

7.7 Chapter Summary

In summary, we expect the combination of transparent huge pages, an RDMA-capable interconnect multiplexed in hardware for use by both guest and hosts, a high performant hypervisor, and post-copy migration and cloning mechanisms, to enable a novel architecture for high performance virtual clusters. These mechanisms, if implemented and properly managed within a performance oriented scheduling system, could help change how cloud infrastructure supports HPC and big data applications, where performance, usability, and reconfigurability all are available. These migration and specialized environment support capabilities may also help enable new runtime systems for large scale scientific applications.

Chapter 8

Conclusion

With the advent of virtualization and the availability of virtual machines through the use of cloud infrastructure, a paradigm shift in distributed systems has occurred. Many services and applications once deployed on workstations, private servers, personal computers, and even some supercomputers have migrated to a cloud infrastructure. The reasons for this trend toward cloud infrastructure are vast, with advantages including increased application flexibility and scalability, the ability for providers to leverages economies of scale, and customized, on-demand user environments. However, these reasons may not be enough to support all computational challenges within such a virtualized infrastructure.

One example where cloud infrastructure has historically been insufficient is with the support for distributed memory applications. The use of tightly coupled, parallel tasks common in High Performance Computing communities has seen a number of problems and complications when deployed in virtualized infrastructure. While the reasons for lack of integration can be numerous, many challenges stem from two aspects, the performance impact and overhead associated with virtualization, and the lack of hardware necessary to support tightly coupled concurrent tasks. While virtualized cloud infrastructure may not be able to aid in all HPC related activities, it is possible that if these limitations are either overcome or mitigated, virtualization can offer benefits to many in the HPC community. These benefits could include

dynamic resource allocation, enhanced migration and data management capabilities, or even bursting capabilities for rare event simulations, to name a few.

This dissertation looks to evaluate virtualization's ability to support mid-tier scientific HPC applications. From the beginning, this dissertation proposes the advent of high performance virtual clusters to support advanced scientific computation, including tightly coupled HPC applications. This dissertation's framework for building such an environment aims to identify virtualization overhead and to find solutions and best practices with performant hypervisors. This effort also includes defining the methodology needed for supporting advanced accelerators and interconnects common in HPC environments in order to enable a new class of applications in virtualized infrastructure. The framework has built into it cases for evaluating potential deployments as discussed throughout using benchmarks and real-world applications. Furthermore, it is proposed to use the OpenStack IaaS project to encompass these components together in a unified private cloud architecture.

Chapter 2 studied the related research necessary for defining not only the context for virtualization and cloud computing, but also virtual clusters and the history of supercomputing. Chapter 3 looked to study the applicability of various hypervisors for supporting common HPC workloads through the use of benchmarks from a single-node aspect. This found challenges and some solutions to these workloads, and identified missing gaps that exist.

Chapter 4 started the investigation of the utility of GPUs to support mid-tier scientific applications using the Xen hypervisor. This chapter provided a proof-of-concept that, with proper configuration by utilizing the latest in hardware support, GPU passthrough is a viable model for supporting CUDA-enabled applications, a fast-growing application set. Chapter 5 provides an in-depth comparison of multiple hypervisors using the SHOC GPU benchmark suite, as well as a few GPU-enabled

HPC applications. Here we discover our KVM implementation performs at near-native speeds and allows for effective GPU utilization, even outperforming our previous work with the Xen hypervisor.

Chapter 6 takes the lessons learned with KVM in GPU passthrough and adds in SR-IOV InfiniBand support. This high speed, low latency interconnect represents a critical tool for supporting tightly coupled distributed memory applications. With this, a high performance virtual cluster is created. This environment supports two class-leading Molecular Dynamics simulations, LAMMPs and HOOMD-blue, and shows how both applications can not only perform at near-native speeds, but also leverage the latest HPC technologies, such as GPUDirect, for efficient GPU-to-GPU communication across distributed memory resources.

Chapter 7 is an introspective look at other advancements that can be made in virtualization to support high performance virtual clusters. This chaper details the mechanisms whereby virtual clusters leverage PCI passthrough for added hardware and I/O support, along with the reduction of overhead through the use of huge pages. With this, methods are outlined for advanced virtualization techniques, such as live migration leveraging high speed RDMA-capable interconnects, possibilities for moving VMs closer to data sources, VM cloning for fast deployment of virtual clusters themselves, and scheduling considerations for integration of high performance virtual clusters. These tools are details with the consideration of integration within the OpenStack IaaS cloud for private cyberinfrastructure deployments.

Given the efforts of this dissertation, we must evaluate the original hypothesis; can virtualized infrastructure, leveraging the practices of high performance virtual clusters as defined herein, support mid-tier scientific computing endeavors? In the beginning of Chapter 3, looking at the base case when this research started, the answer was a murky *no*. When considering the advances made in hypervisors, the ability to

leverage accelerators such as Nvidia GPUs in Chapter 4 and 5, the addition of a high performance, low-latency interconnect in Chapter 6, and the advanced tuning and configuration with KVM, answer to the hypothesis changes to an optimistic but skeptical *yes*. This conjecture is made by evaluating the support for a common and convoluted HPC application set of various molecular dynamics applications and observing virtualization performance overhead under 2% when compared to native, bare metal configurations.

While more work is needed to evaluate the potential for high performance virtual clusters to scale out, the current body of research does not illuminate any architectural barriers at this time. As such, we expect this work with KVM high performance virtualization to extend significantly beyond the current efforts described herein, perhaps up to supporting high end HPC applications at Petascale. If Petaflop computing is possible within an on-demand cloud infrastructure, this may in fact had a drastic change on the way the community views and utilizes mid-range scientific computing. Perhaps more importantly, the availability of such high-end resources to current and future big data analytics software toolkits and services may also have a drastic impact on overall time-to-solution for big data problems. This convergence between HPC and big data analytics could have significant cost savings for large-scale supercomputing facilities, as it makes it possible to support such diverse workloads with a single, massively parallel, advanced capability hardware platform. Furthermore, allowing simulation an analytics codes to run on the same advanced hardware platform can enable new research in in-situ workflow composition and orchestration, potentialy decreasing the overall experiment wall clock time, leading to scientific discoveries quicker and in greater frequency.

8.1 Impact

This dissertation has illustrated how virtualization can be used to support HPC applications using virtual clusters. While the example applications used herein are in relation to molecular dynamics simulations, it is anticipated that this work is also equally applicable to other fields including astronomy, high energy physics, bioinformatics, and computational chemistry, to name a few. In this, it is also desirous to have these high performance virtual clusters provide a feature-rich yet performance infrastructure to new and emerging big data science applications and platform services. While further work will likely be necessary in storage and I/O efforts related to virtualization, the potential exists today to meet the demands of a convergent infrastructure with virtualization.

It is also possible and likely that such applications can scale with future infrastructure deployments. Next steps in this direction to scale out could see virtual cluster sizes increase to thousands of nodes. Virtualization efforts using the Palacios VMM were able to scale a Cray XT4 system to 4096 nodes, reporting only a 5% overhead [264]. While so far we have seen slightly better results with KVM at smaller scales, it is hopeful that the same scaling may be possible with KVM on newer hardware supporting a more diverse set of user environments.

The model for PCI passthrough illustrated throughout the dissertation may also be able to impact other hardware. First, this could include the Intel Xeon Phi (coprocessor models, not the Knights Landing CPU), or some emerging FPGA implementations like the Stillwater Knowledge Processing Unit (KPU), a distributed data flow processor. The PCI-Express SIG specification has been upgraded in version 4.0 to support larger I/O devices and accelerators with on-bus power delivery, leading to the assumptions that, for at least commodity x86 systems, there could be an increase

in PCIE device utilization. The caveat to this will be if the PCIE bus is abandoned or superseded by other methods, such as System-on-Chip designs or Nvidia's NVLink effort [265], where such new designs do not take into account IOMMU virtualization. While HPC accelerator usage could very well wane in the wake of novel many-core architectures, such as Knights Landing CPUs [266], these movements have still yet to take hold within the HPC community.

As some of the advances described in the dissertation have already made their way to the OpenStack cloud platform. It is expected that OpenStack's usage will only grow in both academic research and industry over time. Considering this, it may be possible to build such a cloud infrastructure to run high performance virtual clusters at a larger scale. Applying this infrastructure, along with high level experiment management and support services, could lead to a new national scale cyberinfrastructure deployment. In time and with further development, this could be deployed within the NSF-funded XSEDE project, for example. Perhaps, instead of having separated hardware for HPC systems like the XSEDE Stampede resource at UT Austin and a separate cloud infrastructure deployment for HTC computing and science gateways like IU's Jetstream, a single, unified high performance cloud infrastructure could be utilized to provide all such needs concurrently. Finally, because the advantages provided by vitualizaton are only now integrating with HPC, new distributed computing paradigms and runtime systems may not yet have been realized.

Bibliography

[1] B. Alexander, "Web 2.0: A New Wave of Innovation for Teaching and Learning," *Learning*, vol. 41, no. 2, pp. 32–44, 2006.

[2] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos, CA, USA)*, 2008, pp. 5–13.

[3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop, 2008. GCE'08*, 2008, pp. 1–10.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California at Berkeley, Tech. Rep., February 2009. [Online]. Available: http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html

[5] T. Kuhn, *The structure of scientific revolutions*. University of Chicago press Chicago, 1970.

[6] T. Sterling, *Beowulf cluster computing with Linux*. The MIT Press, 2001.

[7] T. Hoare and R. Milner, "Grand challenges for computing research," *The Computer Journal*, vol. 48, no. 1, pp. 49–52, 2005.

[8] J. Dongarra, H. Meuer, and E. Strohmaier, "Top 500 supercomputers," website, November 2013.

[9] P. Buncic, C. A. Sanchez, J. Blomer, L. Franco, A. Harutyunian, P. Mato, and Y. Yao, "Cernvm–a virtual software appliance for lhc applications," in *Journal of Physics: Conference Series*, vol. 219, no. 4. IOP Publishing, 2010, p. 042003.

[10] L.-W. Wang, "A survey of codes and algorithms used in nersc material science allocations," *Lawrence Berkeley National Laboratory*, 2006.

[11] K. Menon, K. Anala, S. G. Trupti, and N. Sood, "Cloud computing: Applications in biological research and future prospects," in *Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on*. IEEE, 2012, pp. 102–107.

[12] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running hpc applications in public clouds," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 395–401. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851535

[13] A. S. Bland, J. Wells, O. E. Messer, O. Hernandez, and J. Rogers, "Titan: Early experience with the cray xk6 at oak ridge national laboratory," *Cray User Group*, 2012.

[14] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson *et al.*, "Xsede: accelerating scientific discovery," *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, 2014.

[15] S. A. Goff, M. Vaughn, S. McKay, E. Lyons, A. E. Stapleton, D. Gessler, N. Matasci, L. Wang, M. Hanlon, A. Lenards *et al.*, "The iplant collaborative: cyberinfrastructure for plant biology," *Frontiers in plant science*, vol. 2, p. 34, 2011.

[16] K. Antypas, "Nersc-6 workload analysis and benchmark selection process," *Lawrence Berkeley National Laboratory*, 2008.

[17] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally, J. Meredith, J. Rogers, P. Roth, K. Spafford *et al.*, "Keeneland: Bringing heterogeneous gpu computing to the computational science community," *Computing in Science and Engineering*, vol. 13, no. 5, pp. 90–95, 2011.

[18] P. Pacheco, *Parallel Programming with MPI*, ser. ISBN. Morgan Kaufmann, October 1996, no. 978-1-55860-339-4.

[19] G. C. Fox, S. W. Otto, and A. J. Hey, "Matrix algorithms on a hypercube i: Matrix multiplication," *Parallel computing*, vol. 4, no. 1, pp. 17–31, 1987.

[20] D. Agrawal, S. Das, and A. El Abbadi, "Big data and cloud computing: current state and future opportunities," in *Proceedings of the 14th International Conference on Extending Database Technology*. ACM, 2011, pp. 530–533.

[21] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] S. Kamburugamuve, G. Fox, D. Leake, and J. Qiu, "Survey of apache big data stack," Ph.D. dissertation, Ph. D. Qualifying Exam, Dept. Inf. Comput., Indiana Univ., Bloomington, IN, 2013.

[23] M. Chen, S. Mao, and Y. Liu, "Big data: a survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[24] G. Bell, T. Hey, and A. Szalay, "Beyond the data deluge," *Science*, vol. 323, no. 5919, pp. 1297–1298, 2009.

[25] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.

[26] S. Jha, J. Qiu, A. Luckow, P. K. Mantha, and G. C. Fox, "A tale of two data-intensive paradigms: Applications, abstractions, and architectures," in *Proceedings of the 3rd International Congress on Big Data*, 2014.

[27] J. Qiu, S. Jha, A. Luckow, and G. C. Fox, "Towards hpc-abds: An initial high-performance big data stack," *Building Robust Big Data Ecosystem ISO/IEC JTC 1 Study Group on Big Data*, pp. 18–21, 2014.

[28] M. W. ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. K. D. Panda, "High-performance rdma-based design of hadoop mapreduce over infiniband," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1908–1917.

[29] S. Ekanayake, S. Kamburugamuve, and G. Fox, "Spidal: High performance data analytics with java and mpi on large multicore hpc clusters," in *Proceedings of 24th High Performance Computing Symposium (HPC 2016)*, 2016.

[30] F. Tian and K. Chen, "Towards optimal resource provisioning for running mapreduce programs in public clouds," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 155–162.

[31] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke, S. Jaconette, M. Levenhagen *et al.*, "Palacios and kitten: New high performance operating systems for scalable virtualized and native supercomputing," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[32] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang, "Virtual clusters for grid communities," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 513–520, 2006.

[33] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1998, pp. 62–82.

[34] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, "Cost-benefit analysis of cloud computing versus desktop grids," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–12.

[35] T. Bell, B. Bompastor, S. Bukowiec, J. C. Leon, M. Denis, J. van Eldik, M. F. Lobo, L. F. Alvarez, D. F. Rodriguez, A. Marino *et al.*, "Scaling the cern openstack cloud," in *Journal of Physics: Conference Series*, vol. 664, no. 2. IOP Publishing, 2015, p. 022003.

[36] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "Mapreduce in the clouds for science," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 565–572.

[37] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "A performance analysis of ec2 cloud computing services for scientific computing," in *International Conference on Cloud Computing*. Springer, 2009, pp. 115–131.

[38] J. Dongarra *et al.*, "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, p. 1094342010391989, 2011.

[39] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. K. and Dean Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "Exascale computing study: Technology challenges in achieving exascale systems," *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15, 2008.

[40] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *International Conference on High Performance Computing for Computational Science*. Springer, 2010, pp. 1–25.

[41] J. Suh, "Proximimity scheduler in openstack," Webpage. [Online]. Available: https://wiki.openstack.org/wiki/ProximityScheduler

[42] S. Kamburugamuve, S. Ekanayake, M. Pathirage, and G. Fox, "Towards high performance processing of streaming data in large data centers," in *HPBDC 2016 IEEE International Workshop on High-Performance Big Data Computing in conjunction with The 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2016), Chicago, Illinois USA, Friday*, 2016.

[43] A. J. Younge, C. Reidy, R. Henschel, and G. C. Fox, "Evaluation of smp shared memory machines for use with in-memory and openmp big data applications," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 1597–1606.

[44] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, and G. Pike, "Design of the FutureGrid Experiment Management Framework," in *Proceedings of Gateway Computing Environments 2010 at Supercomputing 2010*. New Orleans, LA: IEEE, Nov 2010. [Online]. Available: http://grids.ucs.indiana.edu/ptliupages/publications/vonLaszewski-10-FG-exp-GCE10.pdf

[45] S. Drake and O. development team, "Heat: OpenStack Orchestration," Webpage. [Online]. Available: https://wiki.openstack.org/wiki/Heat

[46] G. Von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, "Accessing multiple clouds with cloudmesh," in *Proceedings of the 2014 ACM international workshop on Software-defined ecosystems*. ACM, 2014, pp. 21–28.

[47] "The magellan project," Webpage. [Online]. Available: http://magellan.alcf.anl.gov/

[48] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon, "The Magellan Report on Cloud Computing for Science," U.S. Department of Energy Office of Science Office of Advanced Scientific Computing Research (ASCR), Tech. Rep., Dec. 2011.

[49] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman, and N. Wright, "Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud," in *2nd IEEE*

International Conference on Cloud Computing Technology and Science. IEEE, 2010, pp. 159–168.

[50] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[51] D. M. Jacobsen and R. S. Canon, "Contain this, unleashing docker for hpc," *Proceedings of the Cray User Group*, 2015.

[52] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE, 2013, pp. 233–240.

[53] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.

[54] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, New York, U. S. A., Oct. 2003, pp. 164–177.

[55] "Vmware esx server," [Online], http://www.vmware.com/de/products/vi/esx/.

[56] J. Duato, A. J. Pena, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-Orti, "Enabling CUDA acceleration within virtual machines using rCUDA," in *High Performance Computing (HiPC), 2011 18th International Conference on*. IEEE, 2011, pp. 1–10.

[57] L. Shi, H. Chen, J. Sun, and K. Li, "vCUDA: GPU-accelerated high-performance computing in virtual machines," *Computers, IEEE Transactions on*, vol. 61, no. 6, pp. 804–816, 2012.

[58] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," in *Proceedings of the Linux Symposium*, vol. 1, 2007, pp. 225–230.

[59] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[60] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Intl. J. Supercomputer Applications*, vol. 15, no. 3, 2001.

[61] I. Foster, C. Kesselman *et al.*, "The Physiology of the Grid:An Open Grid Services Architecture for Distributed Systems Integration," Argonne National Laboratory, Chicago, Tech. Rep., Jan. 2002.

[62] D. DiNucci, "Fragmented future," *AllBusiness–Champions of Small Business*, 1999. [Online]. Available: http://www.cdinucci.com/Darcy2/articles/Print/Printarticle7.html

[63] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek, "Web Services Choreography Interface," Jun. 2002, version 1.0. [Online]. Available: http://wwws.sun.com/software/xml/developers/wsci/index.html

[64] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2004.

[65] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, and J. Tao, "Cloud Computing: a Perspective Study," *New Generation Computing*, vol. 28, pp. 63–69, Mar 2010. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/10-cloudcomputing-NGC/vonLaszewski-10-NGC.pdf

[66] G. von Laszewski, A. J. Younge, X. He, K. Mahinthakumar, and L. Wang, "Experiment and Workflow Management Using Cyberaide Shell," in *Proceedings of the 4th International Workshop on Workflow Systems in e-Science (WSES 09) with 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 09)*. IEEE, May 2009. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/09-gridshell-ccgrid/vonLaszewski-ccgrid09-final.pdf

[67] G. von Laszewski, F. Wang, A. J. Younge, X. He, Z. Guo, and M. Pierce, "Cyberaide JavaScript: A JavaScript Commodity Grid Kit," in *Proceedings of the Grid Computing Environments 2007 at Supercomputing 2008*. Austin, TX: IEEE, Nov 2008. [Online]. Available: http://cyberaide.googlecode.com/svn/trunk/papers/08-javascript/vonLaszewski-08-javascript.pdf

[68] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, "Comparison of multiple cloud frameworks," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, June 2012, pp. 734–741.

[69] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," *INC, IMS and IDC*, pp. 44–51, 2009.

[70] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[71] S. A. Baset, "Cloud slas: present and future," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 2, pp. 57–66, 2012.

[72] "Amazon Elastic Compute Cloud." [Online]. Available: http://aws.amazon.com/ec2/

[73] S. Krishnan and J. L. U. Gonzalez, "Google compute engine," in *Building Your Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 53–81.

[74] "Nimbus Project," http://www.nimbusproject.org. Last access Mar. 2011.

[75] K. Keahey, I. Foster, T. Freeman, and X. Zhang, "Virtual workspaces: Achieving quality of service and quality of life in the grid," *Scientific Programming Journal*, vol. 13, no. 4, pp. 265–276, 2005.

[76] "Amazon Web Services S3 REST API," [Online]. [Online]. Available: http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf

[77] "Jets3t project," http://bitbucket.org/jmurty/jets3t/wiki/Home. Last Access Mar. 2011.

[78] "Boto project," http://code.google.com/p/boto. Last Access Mar. 2011.

[79] "S3tools project," http://s3tools.org/s3cmd. Last Access Mar. 2011.

[80] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-source Cloud-computing System," *Proceedings of Cloud Computing and Its Applications*, 2008.

[81] "Eucalyptus open-source cloud computing infrastructure," an Overview, Eucalypts Systems, Inc. 2009.

[82] "Eucalyptus," http://www.eucalyptus.com, Last Access Mar. 2011.

[83] I. H. Shaon, "Eucalyptus and its components," Webpage. [Online]. Available: https://mdshaonimran.wordpress.com/2011/11/26/eucalyptus-and-its-components

[84] "Openstack. cloud software," http://openstack.org, Last Access Mar. 2011.

[85] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[86] "Opennebula project," http://www.opennebula.org. Last access Mar. 2011.

[87] I. M. Llorente, R. Moreno-Vozmediano, and R. S. Montero, "Cloud computing for on-demand grid resource provisioning," *Advances in Parallel Computing*, vol. 18, no. 5, pp. 177–191, 2009.

[88] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Capacity leasing in cloud systems using the opennebula engine," *Cloud Computing and its Applications*, 2008.

[89] "Libvirt API webpage," http:// libvirt.org. Last access Mar. 2011.

[90] "Elastichosts webpage," http://www.elastichosts.com. Last Access Mar. 2011.

[91] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao *et al.*, "From virtualized resources

to virtual computing grids: the in-vigo system," *Future Generation Computer Systems*, vol. 21, no. 6, pp. 896–909, 2005.

[92] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle, "Dynamic virtual clusters in a grid site manager," in *12th IEEE International Symposium on High Performance Distributed Computing, 2003. Proceedings*, 2003, pp. 90–100.

[93] I. VMware, "VMware vCloud Air," Webpage. [Online]. Available: http://vcloud.vmware.com

[94] CERN, "LHC Computing Grid Project," Web Page, Dec. 2003. [Online]. Available: http://lcg.web.cern.ch/LCG/

[95] J. Luo, A. Song, Y. Zhu, X. Wang, T. Ma, Z. Wu, Y. Xu, and L. Ge, "Grid supporting platform for AMS data processing," *Lecture notes in computer science*, vol. 3759, p. 276, 2005.

[96] "CMS," Web Page. [Online]. Available: http://cms.cern.ch/

[97] J. Diaz, A. J. Younge, G. von Laszewski, F. Wang, and G. C. Fox, "Grappling Cloud Infrastructure Services with a Generic Image Repository," in *Proceedings of Cloud Computing and Its Applications (CCA 2011)*, Argonne, IL, Mar 2011.

[98] I. Foster, "The anatomy of the grid: Enabling scalable virtual organizations," in *Euro-Par 2001 Parallel Processing: 7th International Euro-Par Conference*. Springer, 2001, pp. 1–5. [Online]. Available: www.globus.org/alliance/publications/papers/anatomy.pdf

[99] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual clusters," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 301–308.

[100] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos, W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni, R. P. Wagner, N. Wilkins-Diehr, and M. L. Norman, "Gateways to discovery: Cyberinfrastructure for the long tail of science," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*, ser. XSEDE '14. New York, NY, USA: ACM, 2014, pp. 39:1–39:8. [Online]. Available: http://doi.acm.org/10.1145/2616498.2616540

[101] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[102] F. Berman, "From TeraGrid to knowledge grid," *Communications of the ACM*, vol. 44, no. 11, pp. 27–28, 2001.

[103] C. Catlett, "The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility," in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002*, 2002, pp. 8–8.

[104] H. H. Goldstine and A. Goldstine, "The electronic numerical integrator and computer (eniac)," *Mathematical Tables and Other Aids to Computation*, vol. 2, no. 15, pp. 97–110, 1946.

[105] J. E. Thornton, "Design of a computer - the control data 6600," 1970.

[106] R. M. Russell, "The cray-1 computer system," *Communications of the ACM*, vol. 21, no. 1, pp. 63–72, 1978.

[107] C. L. Seitz, "The cosmic cube," *Communications of the ACM*, vol. 28, no. 1, pp. 22–33, 1985.

[108] W. D. Hillis, *The connection machine*. MIT press, 1989.

[109] X. Zhang, C. Yang, F. Liu, Y. Liu, and Y. Lu, "Optimizing and scaling hpcg on tianhe-2: early experience," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2014, pp. 28–41.

[110] A. Geist, *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press, 1994.

[111] B. Carpenter, V. Getov, G. Judd, A. Skjellum, and G. C. Fox, "Mpj: Mpi-like message passing for java," 2000.

[112] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*. Springer, 2004, pp. 97–104.

[113] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[114] M. J. Koop, T. Jones, and D. K. Panda, "Mvapich-aptus: Scalable high-performance multi-transport mpi over infiniband," in *IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008.* IEEE, 2008, pp. 1–12.

[115] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes," in *2009 17th Euromicro international conference on parallel, distributed and network-based processing*. IEEE, 2009, pp. 427–436.

[116] D. A. Jacobsen, J. C. Thibault, and I. Senocak, "An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters," in *48th AIAA aerospace sciences meeting and exhibit*, vol. 16, 2010, p. 2.

[117] J. J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: past, present and future," *Concurrency and Computation: practice and experience*, vol. 15, no. 9, pp. 803–820, 2003.

[118] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray Users Group (CUG)*, 2010.

[119] M. A. Heroux and J. Dongarra, "Toward a new metric for ranking high performance computing systems," *Sandia Report, SAND2013-4744*, vol. 312, 2013.

[120] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. E. Bernholdt, "Hobbes: Composition and virtualization as the foundations of an extreme-scale os/r," in *Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers*. ACM, 2013, p. 2.

[121] S. Perarnau, R. Gupta, and P. Beckman, "Argo: An exascale operating system and runtime," in *Poster Proceedings from IEEE/ACM Supercomputing 2015*, 2015.

[122] T. Sterling, D. Kogler, M. Anderson, and M. Brodowicz, "SLOWER: A performance model for Exascale computing," *Supercomputing Frontiers and Innovations*, vol. 1, pp. 42–57, Sep 2014.

[123] E. Ciurana, *Developing with Google App Engine*. Springer, 2009.

[124] D. Chappell, "Introducing windows azure," Microsoft, Inc, Tech. Rep., 2009.

[125] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, "Science clouds: Early experiences in cloud computing for scientific applications," *Cloud Computing and Applications*, vol. 2008, 2008.

[126] R. Creasy, "The origin of the VM/370 time-sharing system," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.

[127] "Amazon elastic compute cloud," [Online], http://aws.amazon.com/ec2/.

[128] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, "Virtual Workspaces in the Grid," *Lecture Notes in Computer Science*, vol. 3648, pp. 421–431, 2005. [Online]. Available: http://workspace.globus.org/papers/VW_EuroPar05.pdf

[129] J. Fontan, T. Vazquez, L. Gonzalez, R. S. Montero, and I. M. Llorente, "OpenNEbula: The Open Source Virtual Machine Manager for Cluster Computing," in *Open Source Grid and Cluster Software Conference*, San Francisco, CA, USA, May 2008.

[130] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, "From virtualized resources to virtual computing Grids: the In-VIGO system," *Future Generation Comp. Syst.*, vol. 21, no. 6, pp. 896–909, 2005.

[131] Rackspace, "Openstack," WebPage, Jan 2011. [Online]. Available: http://www.openstack.org/

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

[133] J. Watson, "Virtualbox: bits and bytes masquerading as machines," *Linux Journal*, vol. 2008, no. 166, p. 1, 2008.

[134] D. Leinenbach and T. Santen, "Verifying the Microsoft Hyper-V Hypervisor with VCC," *FM 2009: Formal Methods*, pp. 806–809, 2009.

[135] I. Parallels, "An introduction to os virtualization and parallels virtuozzo containers," Parallels, Inc, Tech. Rep., 2010. [Online]. Available: http://www.parallels.com/r/pdf/wp/pvc/Parallels_Virtuozzo_Containers_WP_an_introduction_to_os_EN.pdf

[136] D. Bartholomew, "Qemu: a multihost, multitarget emulator," *Linux Journal*, vol. 2006, no. 145, p. 3, 2006.

[137] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *Proceedings of the 2007 workshop on Experimental computer science*, ser. ExpCS '07. New York, NY, USA: ACM, 2007.

[138] Oracle, "Performance evaluation of oracle vm server virtualization software," Oracle, Whitepaper, 2008. [Online]. Available: http://www.oracle.com/us/technologies/virtualization/oraclevm/026997.pdf

[139] K. Adams and O. Agesen, "A comparison of software and hardware techniques for x86 virtualization," in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. ACM, 2006, pp. 2–13, vMware.

[140] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance*

Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on. IEEE, 2007, pp. 200–209.

[141] S. Rixner, "Network virtualization: Breaking the performance barrier," *Queue*, vol. 6, no. 1, p. 36, 2008.

[142] S. Nanda and T. Chiueh, "A survey of virtualization technologies," Tech. Rep., 2005.

[143] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, "Performance implications of virtualizing multicore cluster machines," in *Proceedings of the 2nd workshop on System-level virtualization for high performance computing*. ACM, 2008, pp. 1–8.

[144] R. Harper and K. Rister, "Kvm limits arbitrary or architectural?" IBM Linux Technology Center, Jun 2009.

[145] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, and A. Brinkmann, "Non-intrusive virtualization management using libvirt," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 574 –579.

[146] "FutureGrid," Web Page, 2009. [Online]. Available: http://www.futuregrid.org

[147] J. J. Dujmovic and I. Dujmovic, "Evolution and evaluation of spec benchmarks," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 2–9, 1998.

[148] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC Challenge (HPCC) benchmark suite," in *SC06 Conference Tutorial*. Citeseer, 2006.

[149] J. Dongarra and P. Luszczek, "Reducing the time to tune parallel dense linear algebra routines with partial execution and performance modelling," niversity of Tennessee Computer Science Technical Report, Tech. Rep., 2010.

[150] K. Dixit, "The SPEC benchmarks," *Parallel Computing*, vol. 17, no. 10-11, pp. 1195–1209, 1991.

[151] SPEC, "Standard performance evaluation corporation," Webpage, Jan 2011. [Online]. Available: http://www.spec.org/

[152] R. Henschel and A. J. Younge, "First quarter 2011 spec omp results," Webpage, Mar 2011. [Online]. Available: http://www.spec.org/omp/results/res2011q1/

[153] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. Prentice Hall, 2002, vol. 2.

[154] Amazon, "Elastic Compute Cloud." [Online]. Available: http://aws.amazon.com/ec2/

[155] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[156] "Windows azure platform," [Online], http://www.microsoft.com/azure/.

[157] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, G. G. Pike, W. Smith, J. Vockler, R. J. Figueiredo, J. Fortes *et al.*, "Design of the futuregrid experiment management framework," in *Gateway Computing Environments Workshop (GCE), 2010*. IEEE, 2010, pp. 1–10.

[158] OpenStack, "Openstack compute administration manual," 2013.

[159] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *Proceedings of Cloud Computing and Its Applications*, October 2008. [Online]. Available: http://eucalyptus.cs.ucsb.edu/wiki/Presentations

[160] C. Nvidia, "Programming guide," 2008.

[161] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, p. 66, 2010.

[162] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370 – 1380, 2008, k-means implementation on CUDA with 72x speedup. Compares to 4-threaded CPU version with 30x speedup. [Online]. Available: http://www.sciencedirect.com/science/article/B6WKJ-4SVV8GS-2/2/f7a1dccceb63cbbfd25774c6628d8412

[163] Z. Liu and W. Ma, "Exploiting computing power on graphics processing unit," vol. 2, Dec. 2008, pp. 1062–1065.

[164] S. Hong and H. Kim, "An integrated GPU power and performance model," in *ACM SIGARCH Computer Architecture News*, vol. 38. ACM, 2010, pp. 280–289.

[165] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," 2008.

[166] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. P. Walters, "Heterogeneous cloud computing," in *Proceedings of the Workshop on Parallel Programming on Accelerator Clusters (PPAC). Cluster Computing (CLUSTER), 2011 IEEE International Conference on.* IEEE, 2011, pp. 378–385.

[167] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-purpose GPU programming.* Addison-Wesley Professional, 2010.

[168] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W.-m. Hwu, "GPU clusters for high-performance computing," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on.* IEEE, 2009, pp. 1–8.

[169] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C. Sancho, "Entering the petaflop era: the architecture and performance of Roadrunner," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing.* IEEE Press, 2008, p. 1.

[170] S. Craven and P. Athanas, "Examining the viability of FPGA supercomputing," *EURASIP Journal on Embedded systems*, vol. 2007, no. 1, pp. 13–13, 2007.

[171] G. Shainer, T. Liu, J. Layton, and J. Mora, "Scheduling strategies for HPC as a service (HPCaaS)," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on.* IEEE, 2009, pp. 1–6.

[172] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C. Fox, "Analysis of Virtualization Technologies for High Performance Computing Environments," in *Proceedings of the 4th International Conference on Cloud Computing (CLOUD 2011).* Washington, DC: IEEE, July 2011.

[173] W. Wade, "How NVIDIA and Citrix are driving the future of virtualized visual computing," [Online], http://blogs.nvidia.com/blog/2013/05/22/synergy/.

[174] S. Long, "Virtual machine graphics acceleration deployment guide," VMWare, Tech. Rep., 2013.

[175] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski, "Chromium: a stream-processing framework for interactive rendering on clusters," in *ACM Transactions on Graphics (TOG)*, vol. 21. ACM, 2002, pp. 693–702.

[176] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, "A GPGPU transparent virtualization component for high performance computing clouds," in *Euro-Par 2010 - Parallel Processing*, ser. Lecture Notes in Computer Science, P. DAmbra, M. Guarracino, and D. Talia, Eds. Springer Berlin Heidelberg, 2010, vol. 6271, pp. 379–391. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15277-1_37

[177] K. Diab, M. Rafique, and M. Hefeeda, "Dynamic sharing of GPUs in cloud systems," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 947–954.

[178] C.-T. Yang, H.-Y. Wang, and Y.-T. Liu, "Using pci pass-through for gpu virtualization with cuda," in *Network and Parallel Computing*. Springer, 2012, pp. 445–452.

[179] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (SHOC) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.

[180] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen, "Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models," in *Journal of Physics: Conference Series*, vol. 16. IOP Publishing, 2005, p. 65.

[181] F. Silla, "rCUDA: share and aggregate GPUs in your cluster," Nov. 2012, mellanox Booth Presaentation.

[182] H. Jo, J. Jeong, M. Lee, and D. H. Choi, "Exploiting GPUs in virtual machine for biocloud," *BioMed research international*, vol. 2013, 2013.

[183] J. Duato, A. Pena, F. Silla, R. Mayo, and E. Quintana-Orti, "rCUDA: Reducing the number of gpu-based accelerators in high performance clusters," in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, June 2010, pp. 224–231.

[184] B. L. Jacob and T. N. Mudge, "A look at several memory management units, TLB-refill mechanisms, and page table organizations," in *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1998, pp. 295–306.

[185] B.-A. Yassour, M. Ben-Yehuda, and O. Wasserman, "Direct device assignment for untrusted fully-virtualized virtual machines," IBM Research, Tech. Rep., 2008.

[186] M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, and L. Van Doorn, "The price of safety: Evaluating IOMMU performance," in *Ottawa Linux Symposium*, 2007.

[187] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1–12.

[188] V. Jujjuri, E. Van Hensbergen, A. Liguori, and B. Pulavarty, "VirtFS-a virtualization aware file system passthrough," in *Ottawa Linux Symposium*, 2010.

[189] C. Yang, H. Wang, W. Ou, Y. Liu, and C. Hsu, "On implementation of GPU virtualization using PCI pass-through," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings (CloudCom)*, 2012.

[190] M. Dowty and J. Sugerman, "GPU virtualization on VMware's hosted I/O architecture," *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 73 – 82, 2009.

[191] "FutureGrid," Web page. [Online]. Available: http://portal.futuregrid.org

[192] VMWare, "Timekeeping in vmware virtual machines," VMWare Corporation, Tech. Rep., 2011.

[193] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable Heterogeneous Computing (SHOC) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU '10. ACM, 2010, pp. 63–74.

[194] "LAMMPS molecular dynamics simulator," http://lammps.sandia.gov/, [Online; accessed Jan. 2, 2014].

[195] A. Athanasopoulos, A. Dimou, V. Mezaris, and I. Kompatsiaris, "GPU acceleartion for support vector machines," in *Proc 12th International Workshop*

on Image Analysis for Multimedia Interactive Services (WIAMIS 2001), April 2011.

[196] "LULESH shock hydrodynamics application," https://codesign.llnl.gov/lulesh.php, [Online; accessed Jan. 2, 2014].

[197] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995.

[198] W. M. Brown, "GPU acceleration in LAMMPS," http://lammps.sandia.gov/workshops/Aug11/Brown/brown11.pdf, [Online; accessed Jan. 2, 2014].

[199] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.

[200] "Hydrodynamics Challenge Problem," https://codesign.llnl.gov/pdfs/spec-7.pdf, [Online; accessed Jan. 2, 2014].

[201] A. Arcangeli, "Transparent hugepage support," http://www.linux-kvm.org/wiki/images/9/9e/2010-forum-thp.pdf, 2010, [Online; accessed Jan. 29, 2014].

[202] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, "SR-IOV support for virtualization on infiniband clusters: Early experience," in *Cluster Computing and the Grid, IEEE International Symposium on*. IEEE Computer Society, 2013, pp. 385–392.

[203] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos, W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni *et al.*, "Gateways to discovery: Cyberinfrastructure for the long tail of science," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014, p. 39.

[204] P. Luszczek, E. Meek, S. Moore, D. Terpstra, V. M. Weaver, and J. Dongarra, "Evaluation of the hpc challenge benchmarks in virtualized environments," in *Proceedings of the 2011 International Conference on Parallel Processing - Volume 2*, ser. Euro-Par'11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 436–445.

[205] N. Huber, M. von Quast, M. Hauck, and S. Kounev, "Evaluating and modeling virtualization performance overhead for cloud environments." in *CLOSER*, 2011, pp. 563–573.

[206] J. P. Walters, A. J. Younge, D.-I. Kang, K.-T. Yao, M. Kang, S. P. Crago, and G. C. Fox, "GPU-Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014)*. Anchorage, AK: IEEE, 2014.

[207] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, "Sr-iov support for virtualization on infiniband clusters: Early experience," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 385–392.

[208] M. Musleh, V. Pai, J. P. Walters, A. J. Younge, and S. P. Crago, "Bridging the Virtualization Performance Gap for HPC using SR-IOV for InfiniBand," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014)*. Anchorage, AK: IEEE, 2014.

[209] "Aws high performance computing," http://aws.amazon.com/hpc/, last Access Nov. 2014.

[210] "Openstack flavors," http://docs.openstack.org/openstack-ops/content/flavors.html, last Access Nov. 2014.

[211] K. Pepple, *Deploying OpenStack*. O'Reilly Media, Inc., 2011.

[212] S. Hazelhurst, "Scientific computing using virtual high-performance computing: a case study using the amazon elastic computing cloud," in *Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology*. ACM, 2008, pp. 94–103.

[213] E. Amazon, "Amazon elastic compute cloud (amazon ec2)," *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[214] R. Jennings, *Cloud Computing with the Windows Azure Platform*. John Wiley & Sons, 2010.

[215] "Google cloud platform," https://cloud.google.com/, last Access Nov. 2014.

[216] L. Ramakrishnan, R. S. Canon, K. Muriki, I. Sakrejda, and N. J. Wright, "Evaluating interconnect and virtualization performance forhigh performance computing," *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 2, pp. 55–60, Oct. 2012. [Online]. Available: http://doi.acm.org/10.1145/2381056.2381071

[217] M. Righini, "Enabling intel virtualization technology features and benefits," Intel Corporation, Tech. Rep., 2010.

[218] AMD, "AMD i/o virtualization technology (IOMMU) specification," AMD Corporation, Tech. Rep., 2009.

[219] A. Limited, "Arm system memory management unit architecture specification," ARM Limited, Tech. Rep., 2013.

[220] A. J. Younge, J. P. Walters, S. Crago, and G. C. Fox, "Evaluating GPU Passthrough in Xen for High Performance Cloud Computing," in *High-Performance Grid and Cloud Computing Workshop at the 28th IEEE International Parallel and Distributed Processing Symposium*, IEEE. Pheonix, AZ: IEEE, 05/2014 2014.

[221] L. Vu, H. Sivaraman, and R. Bidarkar, "Gpu virtualization for high performance general purpose computing on the esx hypervisor," in *Proceedings of the High Performance Computing Symposium*, ser. HPC '14. San Diego, CA, USA: Society for Computer Simulation International, 2014, pp. 2:1–2:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2663510.2663512

[222] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 gbe nics with sr-iov support," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1–12.

[223] T. P. P. D. L. Ruivo, G. B. Altayo, G. Garzoglio, S. Timm, H. Kim, S.-Y. Noh, and I. Raicu, "Exploring infiniband hardware virtualization in opennebula towards efficient high-performance computing." in *CCGRID*, 2014, pp. 943–948.

[224] "NVIDIA GPUDirect," https://developer.nvidia.com/gpudirect, [Online; accessed Nov. 24, 2014].

[225] G. Shainer, A. Ayoub, P. Lui, T. Liu, M. Kagan, C. R. Trott, G. Scantlen, and P. S. Crozier, "The development of mellanox/nvidia gpudirect over infinibanda new model for gpu to gpu communications," *Computer Science-Research and Development*, vol. 26, no. 3-4, pp. 267–273, 2011.

[226] "Getting Xen working for Intel(R) Xeon Phi(tm) Coprocessor," https://software.intel.com/en-us/articles/

getting-xen-working-for-intelr-xeon-phitm-coprocessor, [Online; accessed Nov. 24, 2014].

[227] "Mellanox Neutron Plugin," https://wiki.openstack.org/wiki/Mellanox-Neutron, [Online; accessed Nov. 24, 2014].

[228] S. Plimpton, P. Crozier, and A. Thompson, "Lammps-large-scale atomic/molecular massively parallel simulator," *Sandia National Laboratories*, 2007.

[229] J. Anderson, A. Keys, C. Phillips, T. Dac Nguyen, and S. Glotzer, "Hoomd-blue, general-purpose many-body dynamics on the gpu," in *APS Meeting Abstracts*, vol. 1, 2010, p. 18008.

[230] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, "The landscape of parallel computing research: A view from berkeley," Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Tech. Rep., 2006.

[231] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo, S. Smallen, W. Smith, and A. Grimshaw, "Futuregrida reconfigurable testbed for cloud, hpc and grid computing," *Contemporary High Performance Computing: From Petascale toward Exascale, Computational Science. Chapman and Hall/CRC*, 2013.

[232] K. Keahey, J. Mambretti, D. K. Panda, P. Rad, W. Smith, and D. Stanzione, "Nsf chameleon cloud," website, November 2014. [Online]. Available: http://www.chameleoncloud.org/

[233] 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. 3, 2006.

[234] F. Silla, "rcuda: towards energy-efficiency in gpu computing by leveraging low-power processors and infiniband interconnects," in *HPC Advisory Council Spain Conference*, 2013.

[235] J. P. Walters, "Achieving Near-Native GPU Performance in the Cloud," in *Nvidia GPU Technology Conference*, 2015.

[236] Y. Luo, "Network i/o virtualization for cloud computing," *IT Professional Magazine*, vol. 12, no. 5, p. 36, 2010.

[237] P. Kutch, "Pci-sig sr-iov primer: An introduction to sr-iov technology," *Intel application note*, pp. 321 211–002, 2011.

[238] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: Current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39–47, 2005.

[239] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne, "Accelerating two-dimensional page walks for virtualized systems," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 1. ACM, 2008, pp. 26–35.

[240] B. Pham, J. Vesely, G. H. Loh, and A. Bhattacharjee, "Using tlb speculation to overcome page splintering in virtual machines," Rutgers University Technical Report DCS-TR-713, Tech. Rep., 2015.

[241] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[242] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[243] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*. ACM, 2009, pp. 51–60.

[244] P. Lu, A. Barbalace, and B. Ravindran, "Hsg-lm: Hybrid-copy speculative guest os live migration without hypervisor," in *Proceedings of the 6th International Systems and Storage Conference*, ser. SYSTOR '13. New York, NY, USA: ACM, 2013, pp. 2:1–2:11. [Online]. Available: http://doi.acm.org/10.1145/2485732.2485736

[245] J. Chu and V. Kashyap, "Transmission of IP over InfiniBand (IPoIB)," IETF, Tech. Rep., 2006.

[246] W. Yu, N. S. Rao, P. Wyckoff, and J. S. Vetter, "Performance of rdma-capable storage protocols on wide-area network," in *2008 3rd Petascale Data Storage Workshop*. IEEE, 2008, pp. 1–5.

[247] W. Huang, Q. Gao, J. Liu, and D. K. Panda, "High performance virtual machine migration with rdma over modern interconnects," in *2007 IEEE International Conference on Cluster Computing*. IEEE, 2007, pp. 11–20.

[248] D. Gilbert, "Post copy live migration," Webpage, 2015. [Online]. Available: http://wiki.qemu.org/Features/PostCopyLiveMigration

[249] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Intel omni-path architecture: Enabling scalable, high performance fabrics," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, Aug 2015, pp. 1–9.

[250] M. Beck and M. Kagan, "Performance evaluation of the rdma over ethernet (roce) standard in enterprise data centers infrastructure," in *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*. International Teletraffic Congress, 2011, pp. 9–15.

[251] E. Kissel and M. Swany, "Photon: Remote memory access middleware for high-performance runtime systems," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 1736–1743.

[252] J. Lofstead, I. Jimenez, and C. Maltzahn, "Consistency and fault tolerance considerations for the next iteration of the doe fast forward storage and io project," in *2014 43rd International Conference on Parallel Processing Workshops*, Sept 2014, pp. 61–69.

[253] C. G. Wright Jr, "Trinity burst buffer-architecture and design," Los Alamos National Laboratory (LANL), Tech. Rep., 2015.

[254] F. B. Schmuck and R. L. Haskin, "Gpfs: A shared-disk file system for large computing clusters." in *FAST*, vol. 2, 2002, pp. 231–244.

[255] M. J. Rashti and A. Afsahi, "10-gigabit iwarp ethernet: comparative performance analysis with infiniband and myrinet-10g," in *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.

[256] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, "Image processing on a simd/spmd architecture: Opsila," in *Pattern Recognition, 1988., 9th International Conference on*, Nov 1988, pp. 430–433 vol.1.

[257] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan, "Snowflock: rapid virtual machine cloning for cloud computing," in *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 2009, pp. 1–12.

[258] H. A. Lagar-Cavilla, J. A. Whitney, R. Bryant, P. Patchin, M. Brudno, E. de Lara, S. M. Rumble, M. Satyanarayanan, and A. Scannell, "Snowflock: Virtual machine cloning as a first-class cloud primitive," *ACM Transactions on Computer Systems (TOCS)*, vol. 29, no. 1, p. 2, 2011.

[259] A. J. Younge, G. von Laszewski, L. Wang, and G. C. Fox, "Providing a Green Framework for Cloud Based Data Centers," in *The Handbook of Energy-Aware Green Computing*, I. Ahmad and S. Ranka, Eds. Chapman and Hall/CRC Press, 2011, ch. 17, in press.

[260] D. P. Berrange, "Openstack performance optimization," in *KVM Forum 2014*, 2014.

[261] Y. Jiang and Y. He, "Openstack pci passthrough," Webpage, Intel, Inc, 2016. [Online]. Available: https://wiki.openstack.org/wiki/Pci_passthrough

[262] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Łapacz, D. M. Swany, S. Trocha, and J. Zurawski, "Perfsonar: A service oriented architecture for multi-domain network monitoring," in *International Conference on Service-Oriented Computing*. Springer, 2005, pp. 241–254.

[263] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena, L. Arantes, and P. Sens, "Towards qos-oriented sla guarantees for online cloud services," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 50–57.

[264] J. R. Lange, K. Pedretti, P. Dinda, P. G. Bridges, C. Bae, P. Soltero, and A. Merritt, "Minimal-overhead virtualization of a large scale supercomputer," in *ACM SIGPLAN Notices*, vol. 46, no. 7. ACM, 2011, pp. 169–180.

[265] N. Agarwal, D. Nellans, M. O'Connor, S. W. Keckler, and T. F. Wenisch, "Unlocking bandwidth for gpus in cc-numa systems," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 354–365.

[266] K. S. Hemmert, M. W. Glass, S. D. Hammond, R. Hoekstra, M. Rajan, S. Dawson, M. Vigil, D. Grunau, J. Lujan, D. Morton *et al.*, "Trinity: Architecture and early experience," in *Cray Users Group*, 2016.