

ARCHITECTURAL PRINCIPLES AND EXPERIMENTATION OF
DISTRIBUTED HIGH PERFORMANCE VIRTUAL CLUSTERS

by

Andrew J. Younge

Submitted to the faculty of

Indiana University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Indiana University

September 2016

Copyright © 2016 Andrew J. Younge

All Rights Reserved

INDIANA UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Andrew J. Younge

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date

Geoffrey C. Fox, Ph.D, Chair

Date

Judy Qiu, Ph.D

Date

Thomas Sterling, Ph.D

Date

D. Martin Swany, Ph.D

INDIANA UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Andrew J. Younge in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Geoffrey C. Fox, Ph.D
Chair, Graduate Committee

Accepted for the Department

Department Chair Name, Chair
Computer Science Program

Accepted for the College

Dean Name, Associate Dean
School of Informatics and Computing

ABSTRACT

ARCHITECTURAL PRINCIPLES AND EXPERIMENTATION OF DISTRIBUTED HIGH PERFORMANCE VIRTUAL CLUSTERS

Andrew J. Younge

Department of Computer Science

Doctor of Philosophy

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 HPC's advanced hardware resources.

Concurrently, we are at the forefront of a convergence in infrastructure between Big Data and HPC, in which a unified distributed computing archi-

tecture could provide computing and storage capabilities for both differing distributed systems use cases. There is the potential to leverage performance and advanced hardware from the HPC community, and provide 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.

This work proposes to bridge the gap between HPC and cloud infrastructure and enable infrastructure convergence through a framework for virtual clusters. It begins with an evaluation of current hypervisors and their viability to run HPC workloads within current infrastructure, which helps define existing performance gaps. Next, we uncover mechanisms to enable the use of specialized hardware available in many HPC resources, such as 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 virtualization using KVM, supports advanced Molecular Dynamics simulations at near-native performance. 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.

ACKNOWLEDGMENTS

Thanks, Mom! (Acknowledgements TBD)

Contents

Table of Contents	viii
List of Figures	xi
1 Introduction	1
1.1 Overview	1
1.2 Research Statement	6
1.3 Research Challenges	11
1.4 Outline	15
2 Related Research	18
2.1 Virtualization	18
2.1.1 Hypervisors and Containers	21
2.2 Cloud Computing	23
2.2.1 Infrastructure-as-a-Service	28
2.2.2 Virtual Clusters	38
2.2.3 The FutureGrid Project	42
2.3 High Performance Computing	45
2.3.1 Brief History of Supercomputing	45
2.3.2 Distributed Memory Computation	47
2.3.3 Exascale	49
3 Analysis of Virtualization Technologies for High Performance Computing Environments	52
3.1 Abstract	52
3.2 Introduction	53
3.3 Related Research	55
3.4 Feature Comparison	57
3.4.1 Usability	59
3.5 Experimental Design	61
3.5.1 The FutureGrid Project	61
3.5.2 Experimental Environment	63
3.5.3 Benchmarking Setup	63

3.6	Performance Comparison	66
3.7	Discussion	71
4	Evaluating GPU Passthrough in Xen for High Performance Cloud Computing	75
4.1	Abstract	75
4.2	Introduction	76
4.3	Virtual GPU Directions	78
4.3.1	Front-end Remote API invocation	79
4.3.2	Back-end PCI passthrough	81
4.4	Implementation	83
4.4.1	Feature Comparison	84
4.5	Experimental Setup	85
4.6	Results	86
4.6.1	Floating Point Performance	87
4.6.2	Device Speed	88
4.6.3	PCI Express Bus	90
4.7	Discussion	92
4.8	Chapter Summary and Future Work	95
5	GPU-Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications	96
5.1	Introduction	96
5.2	Related Work & Background	98
5.2.1	GPU API Remoting	98
5.2.2	PCI Passthrough	98
5.2.3	GPU Passthrough, a Special Case of PCI Passthrough	99
5.3	Experimental Methodology	100
5.3.1	Host and Hypervisor Configuration	100
5.3.2	Guest Configuration	102
5.3.3	Microbenchmarks	102
5.3.4	Application Benchmarks	103
5.4	Performance Results	104
5.4.1	SHOC Benchmark Performance	106
5.4.2	GPU-LIBSVM Performance	111
5.4.3	LAMMPS Performance	113
5.4.4	LULESH Performance	114
5.5	Lessons Learned	116
5.6	Directions for Future Work	118
6	Supporting High Performance Molecular Dynamics in Virtualized Clusters using IOMMU, SR-IOV, and GPUDirect	119
6.1	Introduction	119

6.2	Background and Related Work	122
6.2.1	GPU Passthrough	123
6.2.2	SR-IOV and InfiniBand	124
6.2.3	GPUDirect	126
6.3	A Cloud for High Performance Computing	126
6.4	Benchmarks	127
6.5	Experimental Setup	129
6.5.1	Node configuration	129
6.5.2	Cluster Configuration	131
6.6	Results	131
6.6.1	LAMMPS	132
6.6.2	HOOMD	133
6.7	Discussion	135
6.8	Chapter Summary	136
7	Virtualization advancements to support HPC applications	138
7.1	Memory Page Table Optimizations	139
7.2	Live Migration Mechanisms	142
7.2.1	RDMA-enabled VM Migration	144
7.2.2	Moving the Compute to the Data	147
7.3	Fast VM Cloning	148
7.4	Virtual Cluster Scheduling	151
7.5	Chapter Summary	154
8	Conclusion	155
8.1	Impact	157
	Bibliography	158

List of Figures

1.1	Data analytics and computing ecosystem compared (from [21]), with virtualization included	5
1.2	High Performance Virtual Cluster Framework	9
1.3	Architectural diagram for High Performance Virtual Clusters	10
2.1	Virtual Machine Abstraction	19
2.2	Hypervisors and Containers	22
2.3	View of the Layers within a Cloud Infrastructure	27
2.4	Eucalyptus Architecture	32
2.5	OpenNebula Architecture	36
2.6	Virtual Clusters on Cloud Infrastructure	39
2.7	FutureGrid Participants, Network, and Resources	43
2.8	Top 500 Development over time from 2002 to 2016 [8]	48
3.1	A comparison chart between Xen, KVM, VirtualBox, and VMWare ESX	57
3.2	FutureGrid Participants and Resources	62
3.3	Linpack performance	68
3.4	Fast Fourier Transform performance	69
3.5	Ping Pong bandwidth performance	70
3.6	Ping Pong latency performance (lower is better)	71
3.7	Spec OpenMP performance	72
3.8	Benchmark rating summary (lower is better)	73
4.1	GPU PCI passthrough within the Xen Hypervisor	84
4.2	GPU Floating Point Operations per Second	88
4.3	GPU Fast Fourier Transform	89
4.4	GPU Matrix Multiplication	90
4.5	GPU Stencil and S3D	91
4.6	GPU Device Memory Bandwidth	92
4.7	GPU Molecular Dynamics and Reduction	93
4.8	GPU PCI Express Bus Speed	94

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.	107
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.	108
5.3	SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.	109
5.4	SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.	110
5.5	GPU-LIBSVM relative performance on Bespin system. Higher is better.	112
5.6	GPU-LIBSVM relative performance on Delta showing improved performance within virtual environments. Higher is better.	113
5.7	LAMMPS Rhodopsin benchmark relative performance for Bespin system. Higher is better.	114
5.8	LAMMPS Rhodopsin benchmark relative performance for Delta system. Higher is better.	115
5.9	LULESH relative performance on Bespin. Higher is better.	116
6.1	Node PCI Passthrough of GPUs and InfiniBand	130
6.2	LAMMPS LJ Performance	132
6.3	LAMMPS RHODO Performance	134
6.4	HOOMD LJ Performance with 256k Simulation	135
7.1	Transparent Huge Pages with KVM	141
7.2	Adding extra specs to a VM flavor in OpenStack	152

¹ Chapter 1

² Introduction

³ 1.1 Overview

⁴ For years visionaries in computer science have predicted the advent of utility-based
⁵ computing. This concept dates back to John McCarthy's vision stated at the MIT
⁶ centennial celebrations in 1961.

⁷ “If computers of the kind I have advocated become the computers of the
⁸ future, then computing may someday be organized as a public utility just
⁹ as the telephone system is a public utility... The computer utility could
¹⁰ become the basis of a new and important industry.“

¹¹ Only recently has the hardware and software become available to support the concept
¹² of utility computing on a large scale.

¹³ The concepts inspired by the notion of utility computing have combined with
¹⁴ the requirements and standards of Web 2.0 [1] to create Cloud computing [2–4].
¹⁵ Cloud computing is defined as, “A large-scale distributed computing paradigm that
¹⁶ is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-
¹⁷ scalable, managed computing power, storage, platforms, and services are delivered on

18 demand to external customers over the Internet.” This concept of cloud computing
19 is important to Distributed Systems because it represents a true paradigm shift [5]
20 within the entire IT infrastructure. Instead of adopting the in-house services, client-
21 server model, and mainframes, clouds push resources out into abstracted services
22 hosted *en masse* by larger organizations. This concept of distributing resources is
23 similar to many of the visions of the Internet itself, which is where the “clouds”
24 nomenclature originated, as many people depicted the internet as a big fluffy cloud
25 one connects to.

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

34 While cloud computing is changing IT infrastructure, it also has had a drastic
35 impact on distributed systems itself, which has a different evolution. Gone are the
36 IBM Mainframes of the 1980’s which dominated the enterprise landscape. While
37 some mainframes still exist, they are used only for batch related processing tasks
38 and are relatively unused for scientific applications as they are inefficient at Floating
39 Point Operations. As such, they were replaced with Beowulf Clusters [6], Massively
40 Parallel Processors (MPPs) and Supercomputers of the 90’s and 00’s. A novelty of
41 these distributed memory systems is that instead of just one large machine, many
42 machines are connected together and used to achieve a common goal, thereby maxi-
43 mizing the overall speed of computation. Clusters represent a more commodity-based

44 supercomputer, where off the shelf CPUs are used instead of the highly customized
45 and expensive processors and interconnects found in Supercomputers.

46 Supercomputers and Clusters are best suited for large scale applications. These
47 HPC applications can even include “Grand Challenge” applications [7] and can repre-
48 sent a sizable amount of the scientific calculations done on large-scale Supercomputing
49 resources today. However, there exists a gap of many orders of magnitude between
50 leading-class high performance computing and what is available on the common labo-
51 ratory workshop. This gap, described here as mid-tier scientific computation is a fast
52 growing field that struggles to efficiently harness distributed systems while hoping to
53 minimize extensive development efforts. These mid-tier scientific endeavours need to
54 leverage distributed systems to effectively complete the calculations at hand, however
55 may not require the extreme scale provided by the latest machines at the peak of the
56 Top500 list [8]. It simply may not be feasible for small research teams to effectively
57 handle the development complexity of extreme-scale resources, and instead look to-
58 wards other options. This can include some scientific disciplines such as high energy
59 physics [9], materials science [10], bioinformatics [11], and climate research [12], to
60 name a few.

61 As more domain science turns to the aid of computational resources for con-
62 ducting novel scientific endeavours, there is a continuing and growing need for na-
63 tional cyberinfrastructure initiatives to support an increasingly diverse set of scientific
64 workloads. Substantial growth can be see in the number of computational resource
65 requests [13, 14] from many of the larger computational facilities. Concurrently, there
66 has also been an increase in accelerators and hybrid computing models capable of
67 quickly providing additional resources [15] beyond commodity clusters.

68 Historically, application diversity was separated into High Performance Comput-
69 ing (HPC) and High Throughput Computing (HTC). With HTC, computational

70 problems can be separated into independent tasks that can execute in a pleasingly
71 parallel fashion, happily gaining any available resources and rarely require significant
72 communication or synchronization between tasks. HPC often represents computa-
73 tional problems that require significant communication and coordination to effectively
74 produce results, usually with the use of a communication protocol such as MPI [16].
75 Recently however, many big-data paradigms [17] have been introduced in distributed
76 systems that represent new computational models for distributed computing, such as
77 MapReduce [18] and the corresponding Apache Big Data stack [19, 20]. Supporting
78 these different distributed computational paradigms requires a flexible infrastructure
79 capable of providing computational resources for all possible models in a fast and
80 efficient manner.

81 Currently we are at the forefront of a convergence within scientific computing be-
82 tween HPC and big data computation [21]. This amalgamation of historically differing
83 viewpoints of Distributed Systems looks to combine the performance characteristics
84 of HPC and the pursuit towards Exascale with the data and programmer oriented
85 concurrency models found in Big Data and cloud services.

86 Much of the convergence effort has been focused on applications and platform
87 services. Specifically, significant work towards convergent applications has been out-
88 lined with the Big Data Ogres [22] and the corresponding HPC-ABDS model [23].
89 This convergence can also be seen with efforts in bringing interconnect advances to
90 classically big data platforms such as with InfiniBand and MapReduce [24]. However,
91 the underlying hardware and OS environments are still something to be reconciled,
92 and represents something that virtualization can potentially provide. It is expected
93 that new big data efforts will continue to move in this direction [25], especially if
94 virtualization can make HPC hardware that's traditionally prohibitive in such areas,
95 such as accelerators and high-speed interconnects, readily available to cloud and big

96 data platforms. As the deployment of big data applications and platform services on
 97 virtualized infrastructure is well defined and studied [26], this dissertation has focused
 98 the difficulty of running HPC applications on similar virtualized infrastructure. How-
 99 ever, it is possible and hopeful that research regarding virtualization can also play a
 100 part in bringing advanced hardware and performance-focused considerations to Big
 101 Data applications, effectively cross-cutting the convergence with HPC. In Summary,
 102 success of the research in virtualization could be defined by the ability to support the
 103 convergence between HPC and Big Data.

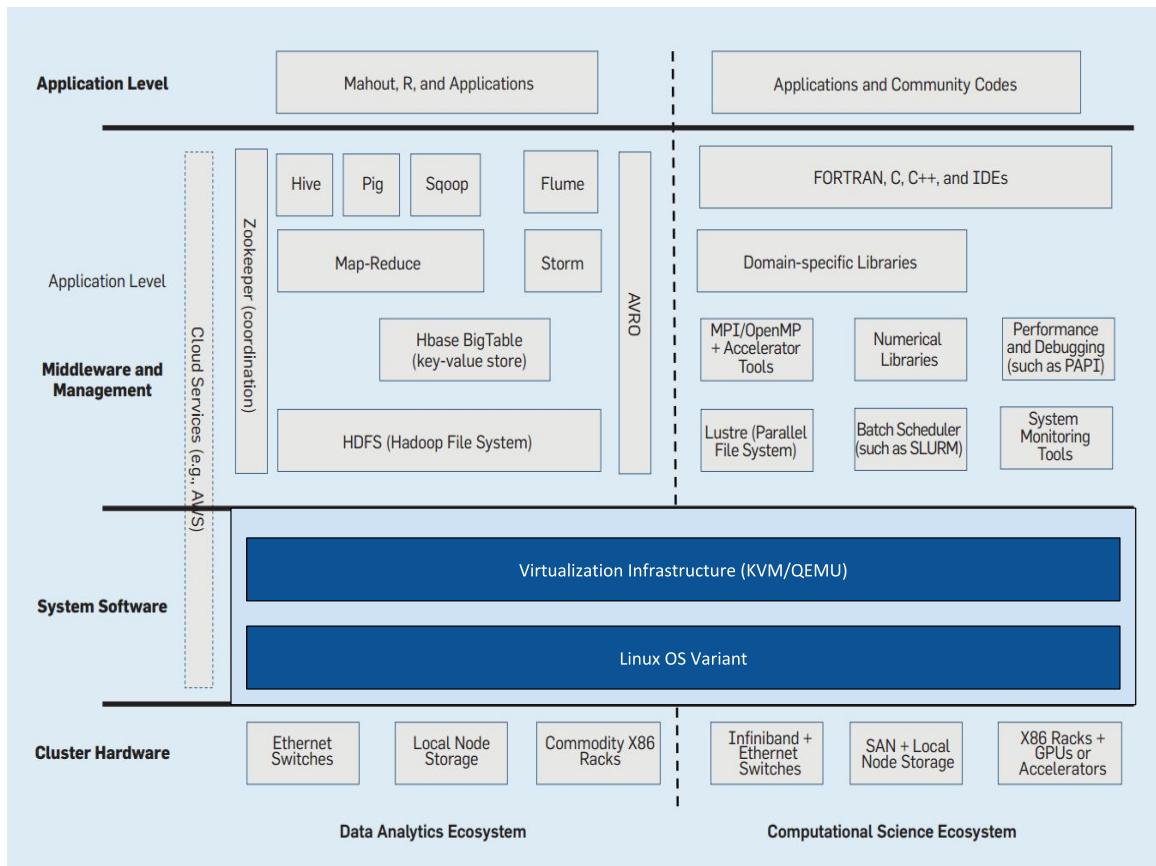


Figure 1.1 Data analytics and computing ecosystem compared (from [21]), with virtualization included

104 To further illustrate where virtualization can play a part in HPC and Big Data
 105 convergence, we look at Figure 1.1 from Reed & Dongarra [21]. While the two ecosys-

106 tems depicted are only representative and in no way exhaustive, they do show how
107 drastically different user environments exist and are reliant on differing hardware. If
108 we insert a performance-oriented virtualization mechanism within the system soft-
109 ware capable of handling the advanced cluster hardware and perform at near-native
110 speeds (at or under 5% overhead, as loosely defined in [27]), it could provide a single,
111 comprehensive *convergent ecosystem* that supports both HPC and Big Data efforts
112 at a critical level.

113 This work proposes the use of virtual clusters [28] to provide distinct, separated
114 environments on similar resources using virtual machines. These virtual clusters,
115 similar in design to the Beowulf clusters and commodity HPC systems, provide a
116 distributed memory environment, usually with a local resource management system
117 [29]. However, past concerns with virtualization have limited the adoption of virtual
118 clusters in many large scale cyberinfrastructure deployments. This has largely been
119 due to the overhead of virtualization, whereby many scientific computations have
120 experienced a significant and notable degradation in performance. In an ecosystem
121 familiarized with HPC systems where performance is paramount, this has been an
122 obstructive hurdle for deploying many tightly coupled applications.

123 1.2 Research Statement

124 With the rise of cloud computing within the greater realm of distributed systems,
125 there have been a number of scientific computing endeavors map well to cloud in-
126 frastructure. This first includes the simple and most common practice of on demand
127 computing where by users can rent-a-workstation [30]. Perhaps these resources are
128 more powerful than a given researcher's laptop and used to run their scientific applica-
129 tions or support greater laboratory collaborative efforts, such as a shared database or

130 Web services. We've also seen virtualized cloud infrastructure support high through-
131 put computing very well. Often times pleasingly parallel applications, be it from high
132 energy physics such as the LHC effort [9, 31] or bioinformatics with BLAST align-
133 ment jobs [11], have proven to run with high efficiency in public and private cloud
134 environments. Furthermore, the rise of public cloud infrastructure has also coincided
135 with increase in big data computation and analytics. Many of these big data plat-
136 form services have evolved complimentary to cloud infrastructure, and as such have
137 a symbiotic relationship with virtualization technologies [32].

138 However, with tightly coupled, high performance distributed memory applications,
139 the same endeavors that support leading class scientific efforts, run very poorly on
140 virtualized cloud infrastructure [33]. This is due to a myriad of addressable reasons
141 ranging from scheduling, abstraction overhead, or a lack of advanced hardware sup-
142 port necessary for tightly coupled communication. This postulates the question on
143 whether virtualization can in fact support such tightly coupled large scale applica-
144 tions without an imposed significant performance penalty. Simply put, *the goal of this*
145 *dissertation is to investigate the viability of mid-tier scientific applications supported*
146 *in virtualized infrastructure.*

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

155 Given the current outlook on virtualization for supporting HPC applications, this

156 dissertation proposes a framework for High Performance Virtual Clusters that enable
157 advanced computational workloads, including tightly coupled distributed memory ap-
158 plications, to run with a high degree of efficiency in virtualized environments. This
159 framework, outlined in Figure 1.2, illustrates the topics to be addressed to provide
160 a supportive virtual cluster environment for high performance mid-tier scientific ap-
161 plications. Areas marked in darker green indicate topics this dissertation may touch
162 upon, whereas light green areas in Figure 1.2 identify outstanding considerations to
163 be investigated. We specifically identify mid-tier distributed memory parallel compu-
164 tations as a focal point for the computational challenges at hand as a way to separate
165 from some of the latest efforts in towards Exascale [34–36] computing. While virtu-
166 alization may in fact be able to play a role towards usable Exascale computing, such
167 efforts fall outside the immediate scope of this dissertation.

168 In order to provide comprehensive high performance virtual clusters, we need to
169 first look at a key area itself, the virtualized infrastructure itself. At the core, we
170 have to consider the hypervisor or virtual machine monitor and the overhead and
171 performance characteristics associated with it. This includes performance tuning
172 considerations, NUMA effects, and advanced hardware device passthrough. Specifi-
173 cally, device passthrough in the context of this manuscript refers to two major device
174 types; GPUs and InfiniBand interconnects (the later using SR-IOV). The virtual in-
175 frastructure also must consider scheduling as a major factor in performing efficient
176 placement of workloads on the underlying host infrastructure, and in particular a
177 Proximity scheduler is of interest [37]. Storage solutions in a virtualized environ-
178 ment is an increasingly important aspect of this framework, as both HPC and big
179 data solutions are continuing to prioritize I/O performance compared to computa-
180 tion. Storage is also likely to be heavily dependent on interconnect considerations as
181 well, as potentially provided by device passthrough, however such I/O considerations

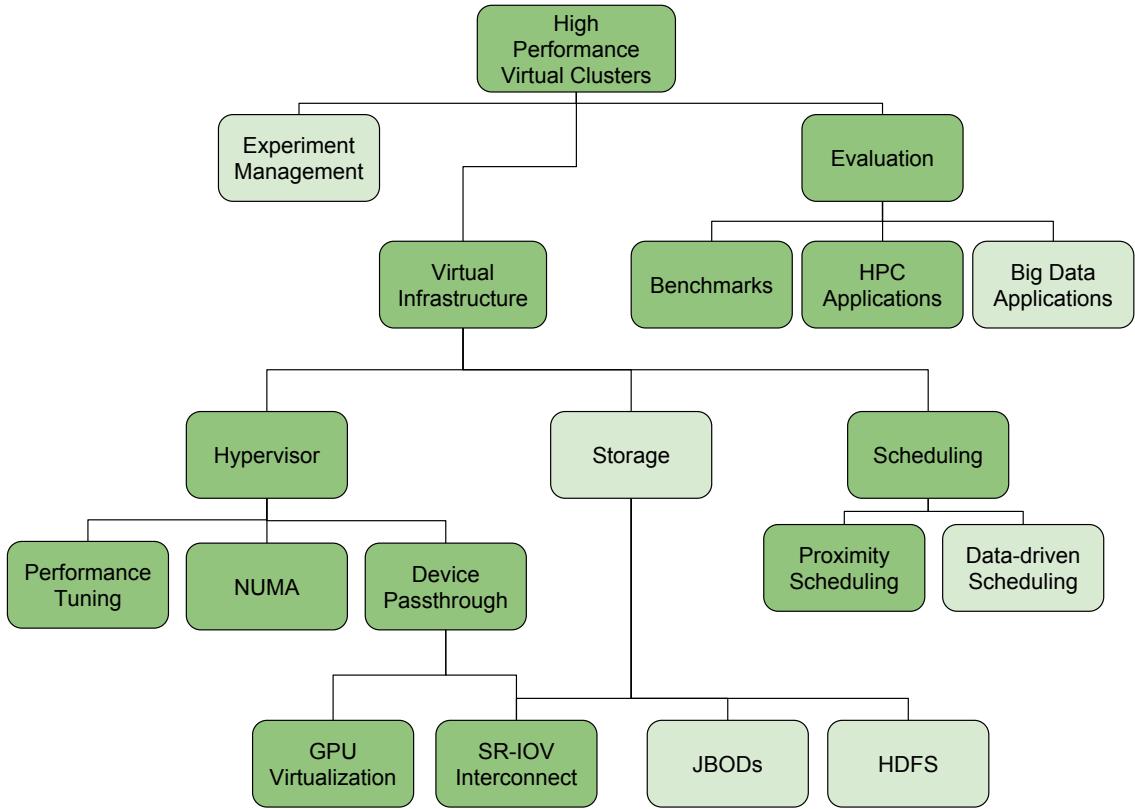


Figure 1.2 High Performance Virtual Cluster Framework

182 lie beyond this dissertation's immediate scope.

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

₁₉₃ viability of virtualization in this context.

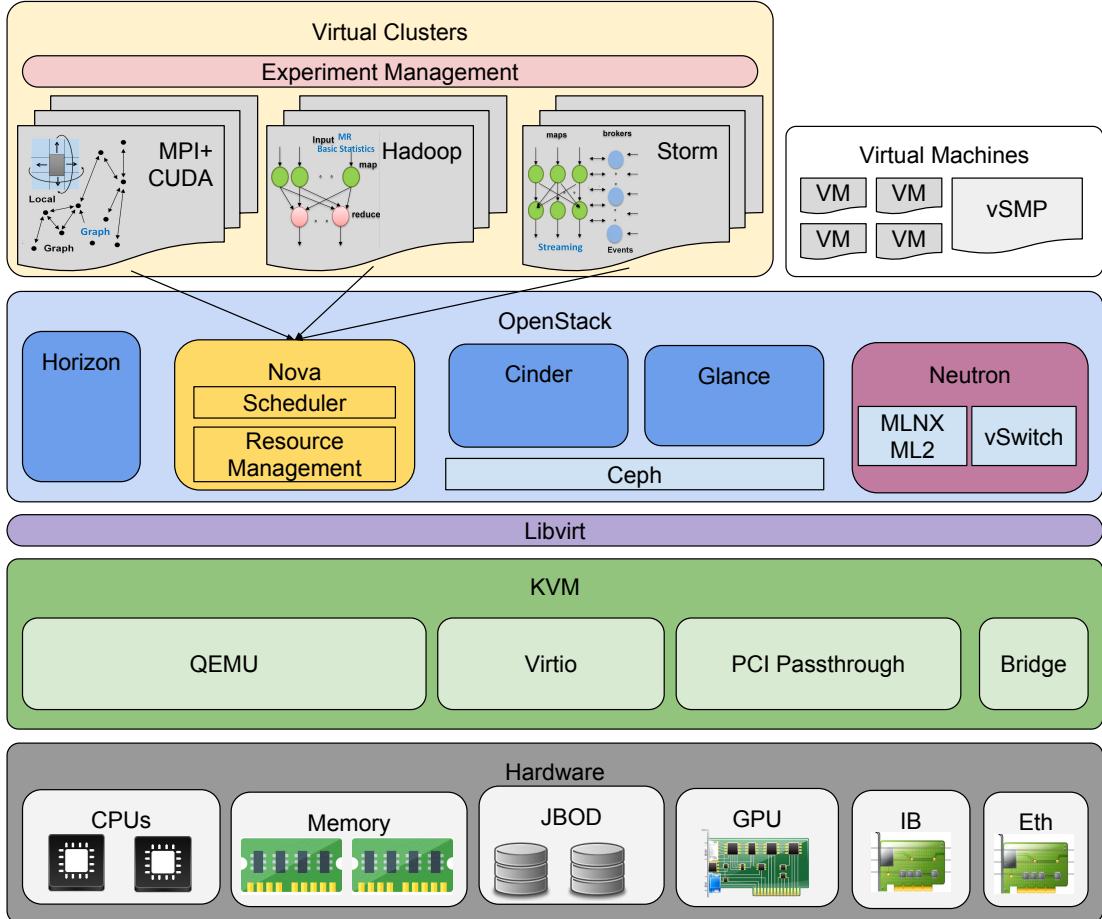


Figure 1.3 Architectural diagram for High Performance Virtual Clusters

₁₉₄ Building from the historical virtual clusters in Grid computing, we see a new archi-
₁₉₅ tectural model for high performance virtual clusters illustrated in 1.3 that is implied
₁₉₆ by this dissertation. Here, we leverage commodity hardware, as well as some ad-
₁₉₇ vanced HPC hardware. While this notion could incorporate a wide array of differing
₁₉₈ technologies, we focus here on GPU-based accelerators and InfiniBand interconnects
₁₉₉ in conjunction with x86 CPUs. Atop this, we leverage KVM and QEMU to provide an
₂₀₀ advanced hypervisor for creating and hosting VMs with direct hardware involvement
₂₀₁ from the lower level. Moving up, the Libvirt API is leveraged due to its hyper-

visor interoperability and popularity. Atop Libvirt is 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 Atop OpenStack, we can create a wide array of virtual clusters and machines to support the wide ranging 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 [38].

One of the higher-level aspects of providing high performance virtual clusters is the high level orchestration of the virtual clusters themselves, which we term experiment management. While this largely remains tangential to this immediate research, it is none the less a key aspect for a successful solution. Some effort has been put forth for virtual cluster experiment management [39], and many ongoing open sources solutions also offer compelling options, such as OpenStack Heat [40]. 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 [41].

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 arrangement of scientific applications. However, there still exist some challenges that will need

¹While 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>), much of the features have since been modified and matured by the OpenStack community in later releases and made available in upstream Nova.

223 to be addressed. This includes a sigma of virtualization being inherently slow and
224 unable to support tightly coupled computations, limitations with advancing at scale,
225 and even that containers may provide a better alternative. While this work hopes to
226 move beyond these challenges, they none the less must be considered.

227 The notion that virtualization and Cloud infrastructure are not able to support
228 parallel distributed memory applications has been characterized many times. One
229 of the most prominent examples of this is the Department of Energy’s Magellan
230 Project [42], where by the Magellan Final Report [43] states the following finding as
231 a Key Finding:

232 **“Finding 2. Scientific applications with minimal communication
233 and I/O are best suited for clouds.”**

234 We have used a range of application benchmarks and micro-benchmarks
235 to understand the performance of scientific applications. Performance of
236 tightly coupled applications running on virtualized clouds using commod-
237 ity networks can be significantly lower than on clusters optimized for these
238 workloads. This can be true at even mid-range computing scales. For ex-
239 ample, we observed slowdowns of about 50x for PARATEC on the Amazon
240 EC2 instances compared to Magellan bare-metal (non-virtualized) at 64
241 cores and about 7x slower at 1024 cores on Amazon Cluster Compute in-
242 stances, the specialized high performance computing offering. As a result,
243 current cloud systems are best suited for high-throughput, loosely coupled
244 applications with modest data requirements.“

245 These findings underscore how classical usage of virtualization in cloud infrastruc-
246 ture has serious performance issues when running tightly coupled distributed memory
247 applications. Many of these performance concerns are sound, given the limitation of a

248 number of virtualization overheads commonplace at the time, including shadow page
249 tables, emulated Ethernet drivers, experimental hypervisors, and a complete lack of
250 sophisticated hardware commonplace in supercomputers and clusters. As a result, the
251 advantages of virtualization, including on-demand resource allocation, live migration
252 and advanced hybrid migration, and user-defined environments, have not been able
253 to effectively show their value in the context of the HPC community.

254 Other and related efforts within the scientific community too found limitations
255 with HPC applications in public cloud environments. This includes the study by
256 Jackson et. al [44] which illustrates how the Amazon Elastic Compute Cloud (EC2)
257 creates a 6x performance impact compared to a local cluster, due to a large part
258 on the limiting Gigabit Ethernet that benchmarks relied heavily on within the EC2
259 system. Other studies also found similar results such as Ostermann [33], concluding
260 that Amazon EC2 “is insufficient for scientific computing at large, though it still
261 appeals to the scientists that need resources immediately and temporarily.” However,
262 these studies are now outdated and do not take into account the advancements in
263 virtualization and hardware availability detailed in this dissertation. Specifically, it is
264 estimated that with the KVM hypervisor in a performance-tuned environment, using
265 accelerators and most certainly a high-speed, low latency interconnect as detailed in
266 Chapter 6, the results would be drastically different.

267 One limitation in this research in high performance virtual clusters is the fact
268 that the degree to which applications can scale remains relatively unknown. While
269 initial results with SR-IOV InfiniBand are promising, scaling is naturally hard to
270 predict. While unfounded, it would be hypothetically possible that as the number
271 of VM’s increases, tail-latency could also increase, causing notable slowdowns during
272 distributed memory synchronization barriers. It is only when infrastructure comes
273 available to support high performance virtual clusters will scaling beyond to thousands

274 of cores and beyond be investigated.

275 Another potential challenge, and perhaps also a strength, is the rising use of
276 containers within industry, such as with efforts like Docker [45]. Recently, we've seen
277 efforts at NERSC/LBNL adapt a container solution called Shifter with the SLURM
278 resource manager on CRAY XC based systems [46]. Shifter's goal is to provide
279 user defined images for NERSC's bioinformatics users, and it adapts remarkably well
280 to the HPC environment. While further efforts are needed by Cray and NERSC
281 to fully provide a container-bases solution on a large scale Supercomputer for all
282 applications, its efforts are in many ways parallel to using virtualization. In Chapter 5,
283 we specifically compare virtualization efforts with LXC [47], a popular Linux container
284 solution and find performance to be comparable and largely near-native.

285 While the major concern with virtualization in the HPC community is perfor-
286 mance issues, virtualization itself may not be fundamentally limited by the overhead
287 that causes issues in running high performance computing applications. Recent im-
288 provements in performance, along with increased usability of accelerators and high
289 speed, low latency interconnects in virtualized environments as demonstrated in this
290 dissertation, have made virtual clusters a more viable solution for mid-tier sci-
291 entific applications. Furthermore, it is possible for virtualization technologies to bring
292 enhanced usability and enable specialized runtime components to future HPC re-
293 sources, adding significant value over today's supercomputing resources. This could
294 potentially include infrastructure advances for higher level cloud platform services for
295 supporting big data applications [23].

296 1.4 Outline

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

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

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

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

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

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

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

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

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

³⁴⁷ performance virtualization, as well as it's potential to impact and support a wide
³⁴⁸ array of disciplines.

³⁴⁹ Chapter 2

³⁵⁰ Related Research

³⁵¹ In order to accurately depict the research presented in this article, the topics within
³⁵² Virtualization, Cloud computing, and High Performance Computing are reviewed in
³⁵³ detail.

³⁵⁴ 2.1 Virtualization

³⁵⁵ Virtualization is a way to abstract the hardware and system resources from a operating
³⁵⁶ system. This is typically performed within a Cloud environment across a large
³⁵⁷ set of servers using a Hypervisor or Virtual Machine Monitor (VMM) which lies in
³⁵⁸ between the hardware and the Operating System (OS). From here, one or more virtualized
³⁵⁹ OSs can be started concurrently as seen in Figure 2.1, leading to one of the key
³⁶⁰ advantages of Cloud computing. This, along with the advent of multi-core processing
³⁶¹ capabilities, allows for a consolidation of resources within any data center. It is the
³⁶² Cloud's job to exploit this capability to its maximum potential while still maintaining
³⁶³ a given QoS. It should be noted that virtualization is not specific to Cloud computing.
³⁶⁴ IBM originally pioneered the concept in the 1960's with the M44/44X systems. It

³⁶⁵ has only recently been reintroduced for general use on x86 platforms.

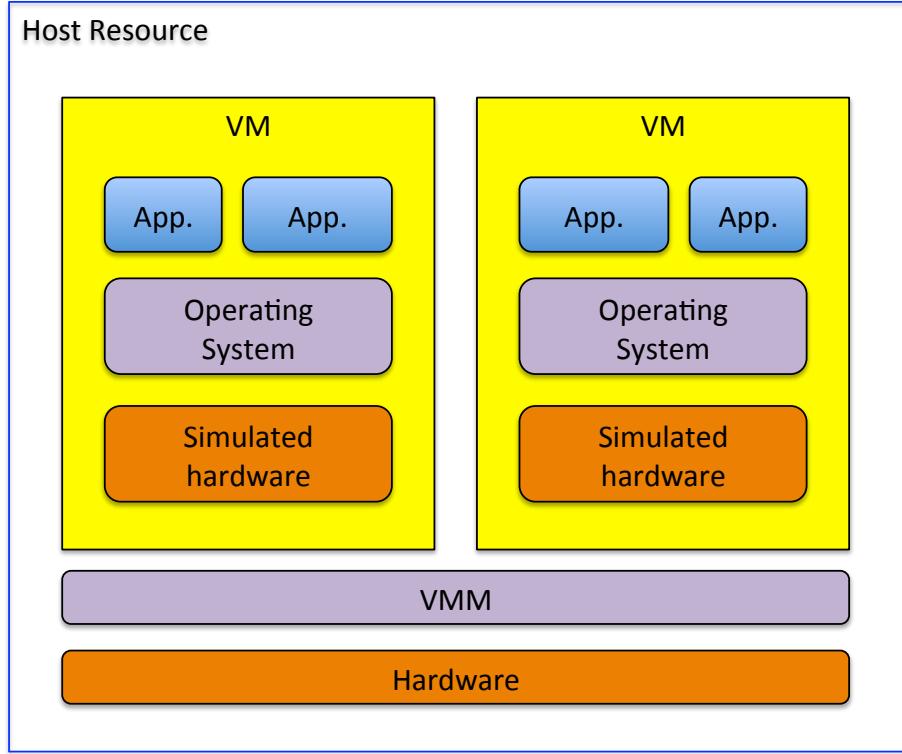


Figure 2.1 Virtual Machine Abstraction

³⁶⁶ However, virtualization is in the most general form, just another form of abstraction.
³⁶⁷ As such, there are in fact many levels to virtualization that exist [48].

- ³⁶⁸ • **ISA -** Virtualization can start from the instruction set architecture (ISA) level,
³⁶⁹ where by an entire processor instruction set is emulated and provided. This may
³⁷⁰ be useful for running software or services developed for one instruction set (say
³⁷¹ MIPS) but is needed to run on Intel x86 hardware. ISA level virtualization is
³⁷² usually emulated through an interpreter which translates source instructions to
³⁷³ target instructions, however this can often be extremely inefficient. Dynamic
³⁷⁴ binary translation can help aid in efficiency by translating blocks of source in-
³⁷⁵ structions to target instructions, however this still can be limiting.

- 376 ● **Hardware** - One of the most important technologies in cloud computing is
377 hardware level virtualization [49, 50]. Hardware virtualization, in its most pure
378 form, refers to the process of creating virtual abstraction to hardware plat-
379 forms, operating systems, or software resources. This enables the creation of
380 1 or more virtual machines (VMs) that are run concurrently on the same op-
381 erating environment, be it hardware or some higher software. Here, a virtual
382 hardware environment is generating for a VM, including providing virtual pro-
383 cessors, memory, and I/O devices, allowing for a multiplexing of VMs to exist,
384 as depicted in Figure 2.1. This layer also manages the physical hardware on
385 behalf of a host OS as well as for guests. While the most common type of hard-
386 ware virtualization is with the Xen Virtual Machine Monitor [49], this method
387 has further separated into type 1 and type 2 hypervisors, as detailed further in
388 Section 2.1.1.

- 399 ● **Operating System** - Moving up the latter of implementation with virtu-
390 alization, we find OS level virtualization, where multiplexing happens at the
391 level of the OS, rather than the hardware. Usually, this refers to isolating a
392 filesystem and associated process and runtime effects in a single "chroot" or
393 "container" environment at the user level, with kernel level operations being
394 handled by a single OS kernel. This containerization allows for multiple user
395 environments to exist concurrently without the complexity of hardware or ISA
396 level virtualization. Containers are also describe in the next section in more
397 detail.

- 400 ● **Library (API)** - Continuing, we move up to library or API level virtualization.
401 Here, some API is separated and a provided to a user or programming environ-
402 ment, and the communication between this API and the back-end library with

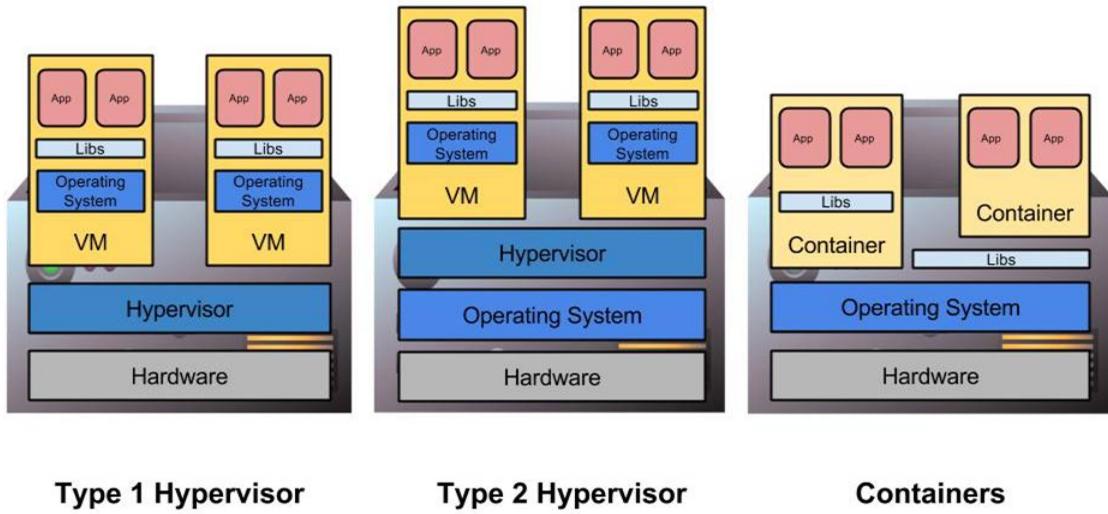
401 the computation is virtualized. This can give the effect that resources are local,
402 when really they are remote and abstract. This is how GPUs are "virtualized"
403 with tools such as rCUDA [51] and vCUDA [52]. This method's performance
404 often can be very dependent on the underlying communications mechanisms, as
405 well as complete virtualization of the whole API (which may be difficult). API
406 virtualization also exists for entire OS libraries to translate between differing
407 OS, without actual containerization.

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

413 2.1.1 Hypervisors and Containers

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

419 Hardware virtualization can actually be dissected in more detail to two different
420 types of hypervisors, and they can be directly compared to containers as seen in
421 Figure 2.2. With a Type 1 virtualization system, the hypervisor or VMM sits directly
422 on the bare-metal hardware, below the OS. These native hypervisors provide direct
423 control of the underlying hardware, and are controlled and operated usually through
424 the use of a privileged VM. One example of a type 1 hypervisor is the Xen Virtual

**Figure 2.2** Hypervisors and Containers

425 Machine Monitor [49], which uses its VMM as well as a privileged Linux OS, called
 426 Dom0, to create and manage other user VMs, or running DomU instances. The
 427 VMM in this place provides the necessary hardware abstraction of CPU, memory,
 428 and some I/O aspects, leaving the control aspects of the other DomUs to Dom0.
 429 With a type 1 hypervisor, all virtualization functions are kept separate from control
 430 and OS functionality, effectively making a cleaner design. This design could lead to
 431 end application performance implications, as illustrated with the Palacios VMM [27].

432 Type 2 hypervisors utilize a different, and sometimes more convoluted design.
 433 With a type 2 hypervisor, there is a "host" OS that, like with native OSs, sits di-
 434 rectly atop hardware. This OS is just like any normal native OS. However, the OS
 435 itself can abstract its own hardware, and provide and manage a VM, effectively as
 436 an OS process. In this case, the hypervisor providing the abstraction is effectively
 437 hosted within, atop, or as a module part of a given OS. There are many different
 438 type 2 hypervisors, the most common of which is the Linux Kernel Virtual Machine
 439 (KVM) [53]. KVM is often used in conjunction with QEMU, a ISA level hypervisor
 440 to provide some basic ISA level virtualization and emulation capabilities. KVM is

441 simply provided as a Linux kernel module within a given host, and guest VMs are
442 run as a single process on the host OS.

443 Both types of hypervisors are very distinct from OS level virtualization, also known
444 as containers. With containers, there is a single OS, however instead of direct hard-
445 ware abstraction,a single kernel is used to simultaneously run multiple user-space
446 instances in a jailed-root environment. These environments may look and feel like
447 a separate machine, but in fact are not. Often times the kernel itself provides re-
448 source management tools to help control resource utilization and allocations. Linux
449 containers (LXC) provide a great example of this, with their use of namespaces for
450 filesystem control and cgroups for resource management. With the recent advent of
451 Docker, which looks to control versioning and easy deployment of traceable contain-
452 ers, this aspect of OS level virtualization has grown in popularity, however security
453 and usability concerns still exist.

454 2.2 Cloud Computing

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

460 Grid computing [54,55], in its practical form, represents the concept of connect-
461 ing two or more spatially and administratively diverse clusters or supercomputers
462 together in a federating manner. The term “the Grid” was coined in the mid 1990’s
463 to represent a large distributed systems infrastructure for advanced scientific and en-
464 gineering computing problems. Grids aim to enable applications to harness the full

465 potential of resources through coordinated and controlled resource sharing by scalable
466 virtual organizations. While not all of these concepts carry over to the Cloud, the
467 control, federation, and dynamic sharing of resources is conceptually the same as in
468 the Grid. This is outlined by [3], as Grids and Clouds are compared at an abstract
469 level and many concepts are remarkably similar. From a scientific perspective, the
470 goals of Clouds and Grids are also similar. Both systems attempt to provide large
471 amounts of computing power by leveraging a multitude of sites running diverse ap-
472 plications concurrently in symphony. The only significant differences between Grids
473 and Clouds exist in the implementation details, and the reproductions of them, as
474 outlined later in this section.

475 The other major component, Web 2.0, is also a relatively new concept in the
476 history of Computer Science. The term Web 2.0 was originally coined in 1999 in a
477 futuristic prediction by Dracy DiNucci [56]: “The Web we know now, which loads
478 into a browser window in essentially static screenfulls, is only an embryo of the Web
479 to come. The first glimmerings of Web 2.0 are beginning to appear, and we are just
480 starting to see how that embryo might develop. The Web will be understood not
481 as screenfulls of text and graphics but as a transport mechanism, the ether through
482 which interactivity happens. It will [...] appear on your computer screen, [...] on your
483 TV set [...] your car dashboard [...] your cell phone [...] hand-held game machines
484 [...] maybe even your microwave oven.” Her vision began to form, as illustrated in
485 2004 by the O’Riley Web 2.0 conference, and since then the term has been a pivotal
486 buzz word among the internet. While many definitions have been provided, Web 2.0
487 really represents the transition from static HTML to harnessing the Internet and the
488 Web as a platform in of itself.

489 Web 2.0 provides multiple levels of application services to users across the Inter-
490 net. In essence, the web becomes an application suite for users. Data is outsourced

491 to wherever it is wanted, and the users have total control over what they interact
492 with, and spread accordingly. This requires extensive, dynamic and scalable host-
493 ing resources for these applications. This demand provides the user-base for much
494 of the commercial Cloud computing industry today. Web 2.0 software requires ab-
495 stracted resources to be allocated and relinquished on the fly, depending on the Web's
496 traffic and service usage at each site. Furthermore, Web 2.0 brought Web Services
497 standards [57] and the Service Oriented Architecture (SOA) [58] which outline the
498 interaction between users and cyber infrastructure. In summary, Web 2.0 defined
499 the interaction standards and user base, and Grid computing defined the underlying
500 infrastructure capabilities.

501 Cloud computing [59] is one of the most explosively expanding technologies in
502 the computing industry today. A Cloud computing implementation typically enables
503 users to migrate their data and computation to a remote location with some varying
504 impact on system performance [60]. This provides a number of benefits which could
505 not otherwise be achieved:

- 506 ● *Scalable* - Clouds are designed to deliver as much computing power as any
507 user needs. While in practice the underlying infrastructure is not infinite, the
508 cloud resources are projected to ease the developer's dependence on any specific
509 hardware.
- 510 ● *Quality of Service (QoS)* - Unlike standard data centers and advanced com-
511 puting resources, a well-designed Cloud can project a much higher QoS than
512 traditionally possible. This is due to the lack of dependence on specific hard-
513 ware, so any physical machine failures can be mitigated without the prerequisite
514 user awareness.
- 515 ● *Specialized Environment* - Within a Cloud, the user can utilize customized tools

516 and services to meet their needs. This can be to utilize the latest library, toolkit,
517 or to support legacy code within new infrastructure.

- 518 • *Cost Effective* - Users finds only the hardware required for each project. This
519 reduces the risk for institutions potentially want build a scalable system, thus
520 providing greater flexibility, since the user is only paying for needed infrastruc-
521 ture while maintaining the option to increase services as needed in the future.
- 522 • *Simplified Interface* - Whether using a specific application, a set of tools or
523 Web services, Clouds provide access to a potentially vast amount of computing
524 resources in an easy and user-centric way. We have investigated such an interface
525 within Grid systems through the use of the Cyberaid project [61, 62].

526 Many of the features noted above define what Cloud computing can be from a
527 user perspective. However, Cloud computing in its physical form has many different
528 meanings and forms. Since Clouds are defined by the services they provide and not
529 by applications, an integrated as-a-service paradigm has been defined to illustrate the
530 various levels within a typical Cloud, as in Figure 2.3.

- 531 • *Clients* - A client interacts with a Cloud through a predefined, thin layer of
532 abstraction. This layer is responsible for communicating the user requests and
533 displaying data returned in a way that is simple and intuitive for the user.
534 Examples include a Web Browser or a thin client application.
- 535 • *Software-as-a-Service (SaaS)* - A framework for providing applications or soft-
536 ware deployed on the Internet packaged as a unique service for users to consume.
537 By doing so, the burden of running a local application directly on the client's
538 machine is removed. Instead all the application logic and data is managed

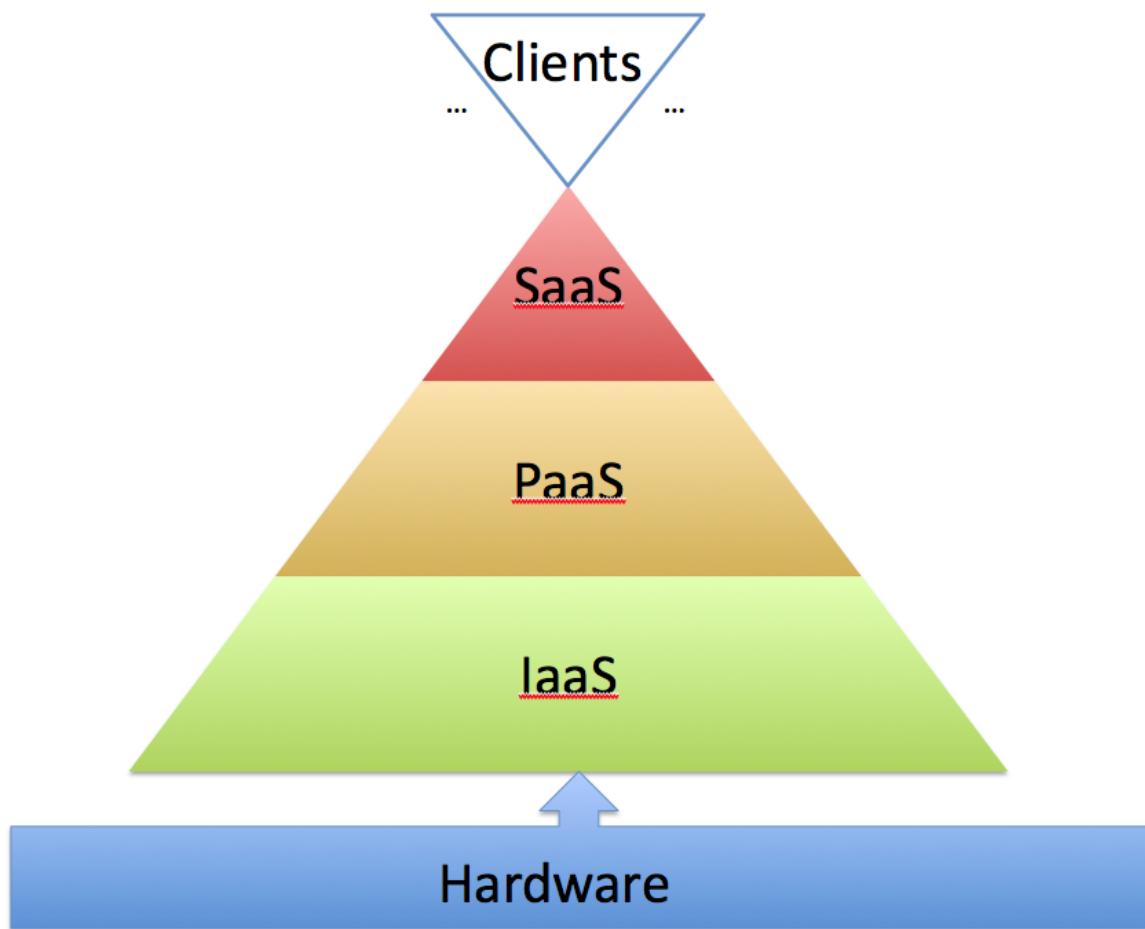


Figure 2.3 View of the Layers within a Cloud Infrastructure

539 centrally and to the user through a browser or thin client. Examples include
540 Google Docs, Facebook, or Pandora.

- 541 • *Platform-as-a-Service (PaaS)* - A framework for providing a unique computing
542 platform or software stack for applications and services to be developed on.
543 The goal of PaaS is to alleviate many of the burdens of developing complex,
544 scalable software by proving a programming paradigm and tools that make ser-
545 vice development and integration a tractable task for many. Examples include
546 Microsoft Azure and Google App Engine.
- 547 • *Infrastructure-as-a-Service (IaaS)* - A framework for providing entire computing

548 resources through a service. This typically represents virtualized Operating
549 Systems, thereby masking the underlying complexity details of the physical
550 infrastructure. This allows users to rent or buy computing resources on demand
551 for their own use without needing to operate or manage physical infrastructure.
552 Examples include Amazon EC2, and OpenStack, and is the major cloud focal
553 point for this dissertation.

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

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

560 2.2.1 Infrastructure-as-a-Service

561 Today there are a number of Clouds that offer solutions for Infrastructure-as-a-Service
562 (IaaS). There have been multiple comparison efforts between various IaaS service
563 [4, 63–65] which provide insight to the similarities and differences between the long
564 array of cloud infrastructure deployment solutions. However at a high level, IaaS can
565 be split into 3 tiers, based on their availability.

- 566 • **Public** - Public IaaS is where the services and virtualizaton of hardware re-
567 sources are provided over the internet. Usually this is in a centralized data
568 centers whereby many users concurrently access these resources from across
569 the globe, often at a pre-negotiated price point and service level agreement
570 (SLA) [66]. Public clouds are the best utilization of economies of scale, by

571 selling hosting services en-masse and thereby offering competitive costs. The
572 Amazon Elastic Compute Cloud (EC2) is a primary example of a public cloud.

- 573 ● **Private** - Private IaaS is where the cloud infrastructure is limited to within
574 a distinct group, set of users, business, or virtual organization. Usually such
575 private cloud infrastructure is also on a private, dedicated network that can
576 either be separate or connected to other services. Data within private clouds
577 are often more secure than those on a public cloud, and as such can be the choice
578 for many users who have sensitive data, or who large-scale users who find better
579 cost, performance, or QoS than what's provided within public clouds.

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

585 **Amazon EC2**

586 The Amazon Elastic Compute Cloud (EC2) [67], is probably the most popular of
587 cloud infrastructure to date, and is used extensively in the IT industry. EC2 is the
588 central component of Amazon Web Services platform. EC2 allows users to effectively
589 rent virtual machines (called instances), hosted within Amazon's data centers, at a
590 certain price point. Through their advanced UI or a RESTful API, users can start,
591 stop, pause, migrate, and destroy instances exactly as needed, and to match the
592 required computational tasks at hand.

593 Amazon EC2 predominantly relies on the Xen hypervisor to provide VMs on
594 demand to users, with an equivalent compute unit equal to a 1.7Ghz Intel Xeon

595 processor. However, recent advancements, instance types, and upgrades to EC2 have
596 increased this compute unit's power. Instance reservations have 3 types: On-demand,
597 Reserved, and Spot. With On-demand instances, users pay by the hour for however
598 long the desired instance is running. Users can instead rent reserved instances, where
599 they pay a one-time (discounted) cost based on a pre-determined allocation time.
600 There is also Spot pricing, where VMs are provisioned only when a given spot price is
601 met, which is determined simply based on supply and demand within the EC2 system
602 itself.

603 EC2 supports a wide array of user environments and setups. From an OS per-
604 spective, this includes running Linux, Unix, and even Windows VM instances. EC2
605 also provides instances with persistent storage through Elastic Block Storage (EBS)
606 and Simple Storage Service (S3) object storage mechanisms. These tools are necessary
607 for data persistence, as EC2 instances, as with most IaaS solutions, do not implic-
608 itly persist data beyond the lifetime of the instance. EBS-rooted instances use an
609 EBS volume as a root device and as such, persist there data beyond the lifetime of
610 a given instance using the EBS volume as a persistent root partition. EC2 also of-
611 fers elastic IPs, whereby public IP addresses are assigned to instances at boot (or in
612 situ), however these elastic IPs do not require the DNS updates to propagate or an
613 administrator to adjust the network.

614 These advanced features, coupled with the first-to-market viability and continual
615 updates have made EC2 the largest cloud infrastructure today. While vendor lock-in
616 is a concern (EC2 is not available for download or replication) and other alternatives
617 exist such as Google's Compute Engine [68], the prevalence and support with EC will
618 likely mean its status quo as the public cloud of choice will continue for the foreseeable
619 future.

620 **Nimbus**

621 Nimbus [69, 70] is a set of open source tools that provide a private IaaS cloud com-
622 putting solution. Nimbus is based on the concept of virtual workspaces previously
623 introduced for Globus [70]. A virtual workspace is an abstraction of an execution
624 environment that can be made dynamically available to authorized clients by using
625 well-defined protocols. In this way, it can create customized environments by de-
626 ploying virtual machines (VMs) among remote resources. To such an end, Nimbus
627 provides a web interface called Nimbus Web. Its aim is to provide administrative and
628 user functions in a friendly interface.

629 Within Nimbus, a storage cloud implementation called Cumulus [69] has been
630 tightly integrated with the other central services, although it can also be used stan-
631 dalone. Cumulus is compatible with the Amazon Web Services S3 REST API [71],
632 but extends its capabilities by including features such as quota management. The
633 Nimbus cloud client uses the Jets3t library [72] to interact with Cumulus. However,
634 since it is compatible with S3 REST API, other interfaces like boto [73] or s2cmd [74]
635 can also be used to interact with Nimbus.

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

642 Nimbus also provides an implementation of EC2’s interface that allows you to use
643 clients developed for the real EC2 system on Nimbus based clouds.

644 **Eucalyptus**

645 Eucalyptus is a product from Eucalyptus Systems [75–77], that developed out of
 646 a research project at the University of California, Santa Barbara. Eucalyptus was
 647 initially aimed at bringing the cloud computing paradigm of computing to academic
 648 super computers and clusters. Eucalyptus provides a Amazon Web Services (AWS)
 649 complaint EC2 based web service interface for interacting with the Cloud service.

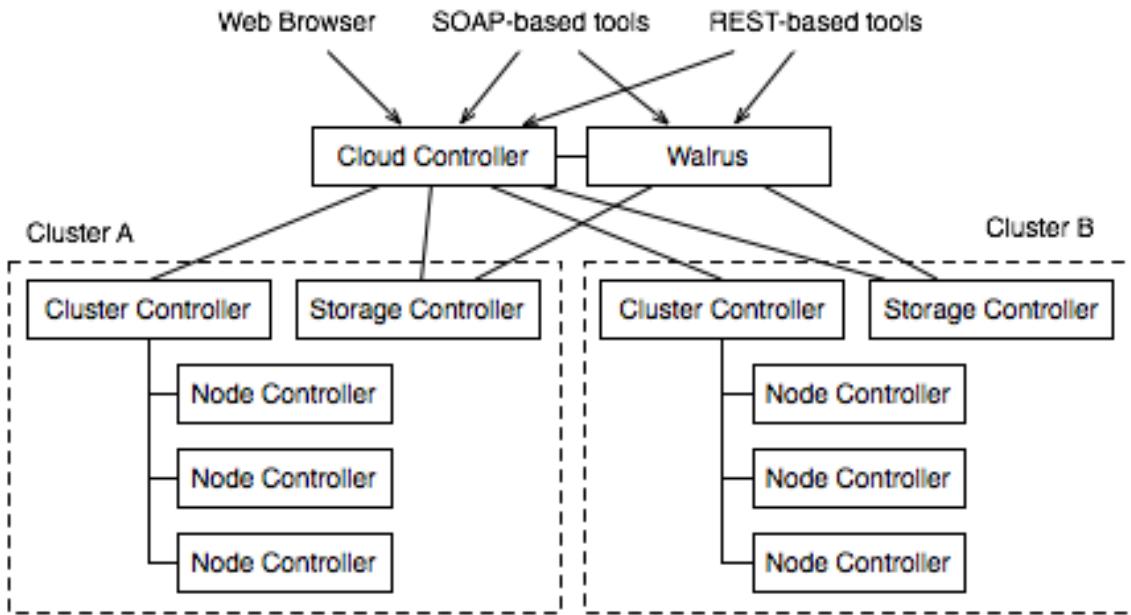


Figure 2.4 Eucalyptus Architecture

650 The architecture depicted in Figure 2.4 is based on a two level hierarchy of the
 651 Cloud controller and the Cluster controller [78]. The Cluster Controller usually man-
 652 ages the nodes within a single cluster and multiple such Cluster Controllers can be
 653 used to connect to a single Cloud Controller. The Cloud Controller is responsible for
 654 the resource management, scheduling and accounting aspects of the Cloud.

655 Being one of the first private cloud computing solutions, Eucalyptus has a focused
 656 user interface. Much of Eucalyptus's design is based on the functionality of Amazon's
 657 EC2 cloud solution, and the user interface is a prime example of that model. While

658 EC2 is a proprietary public cloud, it uses an open interface through the use of well
659 designed Web Services which are open to all. Eucalyptus, looking to provide complete
660 compatibility with EC2 to market the private cloud market, uses the same interface
661 for all communication to the Cloud Controller. With Eucalyptus's interface being
662 AWS complaint, it provides the same form of authentication that AWS supports,
663 namely the shared key and PKI models.

664 While Eucalyptus can be controlled using the EC2 AMI tools, it also provides its
665 own specific tool set; euca2ools. Euca2ools provides support for creating and man-
666 aging keypairs, querying the cloud system, managing VMs, starting and terminating
667 instances, network configuration, and block storage usage. The Eucalyptus system
668 also provides a secure web front end to allow new users to create and manage account
669 information, view available VMs, and download their security credentials.

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

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

681 **OpenStack**

682 OpenStack [79, 80], another private cloud infrastructure service, was introduced by
683 Rackspace and NASA in July 2010. The project is trying to build an open source
684 community spanning technologists, developers, researchers, and industry to share
685 resources and technologies with the goal to create a massively scalable and secure
686 cloud infrastructure. In tradition with other open source projects the entire software
687 is open sources and limited to just open source API's such as Amazon.

688 Historically, OpenStack focuses on the development of two aspects of cloud com-
689 puting to address compute and storage aspects with their OpenStack Compute and
690 OpenStack Storage solutions. According to the documentation “OpenStack Com-
691 pute is the internal fabric of the cloud creating and managing large groups of virtual
692 private servers” and “OpenStack Object Storage is software for creating redundant,
693 scalable object storage using clusters of commodity servers to store Terabytes or even
694 petabytes of data.” However, OpenStack as a platform has evolved much more than
695 its original efforts, and has created a wide array of new sub-projects.

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

706 The OpenStack Swift storage solution is build around a number of interacting com-

707 ponents and concepts including a Proxy Server, a Ring, Object Server, a Container
708 Server, an Account Server, Replication, Updaters, and Auditors. This distributed
709 architecture attempts to have no centralized components, to enable scalability and
710 resiliency for data. Swift represents the long-term, object-based storage, similar to
711 Amazon S3, and attempts to maintain rough API compatibility with S3. As Swift
712 looks to use simple data replication as a main form of resiliency and fast read/write
713 is rarely a priority, Swift is often built using commodity disk drives instead of most
714 costly flash solutions.

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

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

728 Currently, OpenStack exists as one of the largest ongoing private IaaS efforts,
729 with over 500 companies contributing to the effort, and thousands of deployments.
730 While releases have pushed forth approximately every 6 months, the latest current

731 release at the time of writing is *Mitaka*, which now includes full support for GPUs
732 and SR-IOV interconnects. It is expected that OpenStack's prevalence in the cloud
733 computing community will only increase in the next few years.

734 **OpenNebula**

735 OpenNebula [81, 82] is an open-source toolkit which allows to transform existing in-
736 frastructure into an Infrastructure as a Service (IaaS) cloud with cloud-like interfaces.
737 Figure 2.5 shows the OpenNebula architecture and their main components.

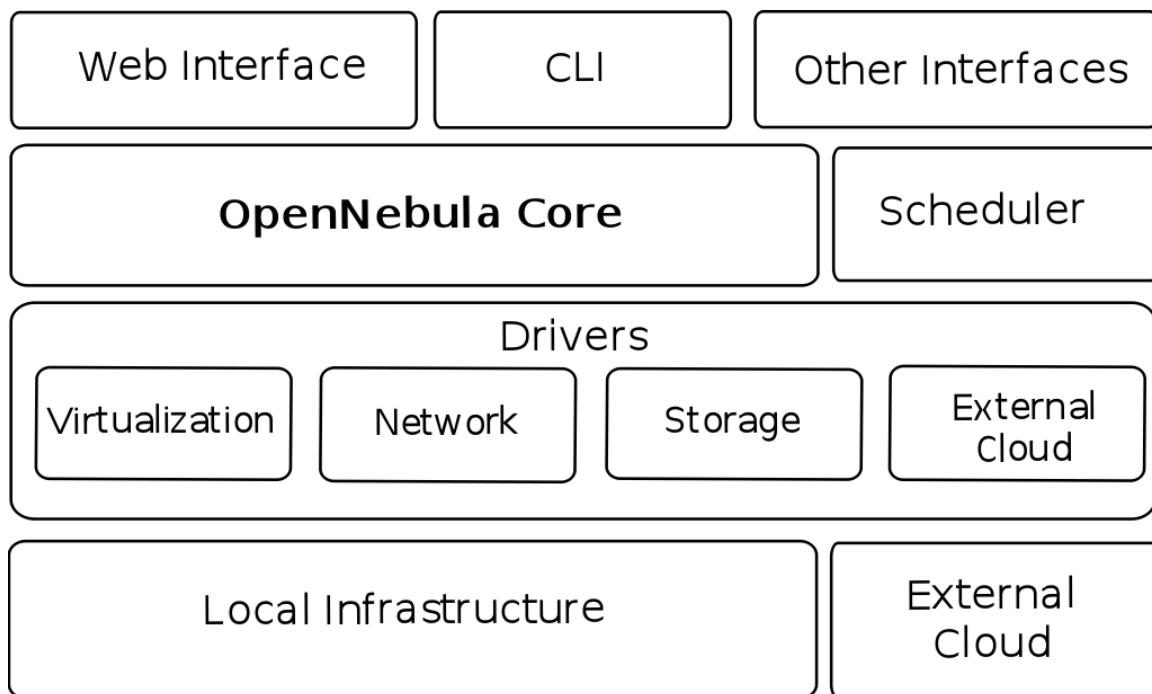


Figure 2.5 OpenNebula Architecture

738 The architecture of OpenNebula has been designed to be flexible and modular to
739 allow its integration with different storage and network infrastructure configurations,
740 and hypervisor technologies. Here, the core is a centralized component that manage
741 the virtual machine's (VM) full life cycle, including setting up networks dynamically
742 for groups of VMs and managing their storage requirements, such as VM disk image

743 deployment or on-the-fly software environment creation. Another important compo-
744 nent is the capacity manager, which governs the functionality provided by the core
745 for scheduling. The default capacity scheduler is a requirement/rank matchmaker.
746 However, it is also possible to develop more complex scheduling policies, through a
747 lease model and advance reservations like Haizea [83]. The last main components are
748 the access drivers. They provide an abstraction of the underlying infrastructure to
749 expose the basic functionality of the monitoring, storage and virtualization services
750 available in the cluster. Therefore, OpenNebula is not tied to any specific environ-
751 ment and can provide a uniform management layer regardless of the virtualization
752 platform.

753 Additionally, OpenNebula offers management interfaces to integrate the core's
754 functionality within other data center management tools, such as accounting or mon-
755 itoring frameworks. To this end, OpenNebula implements the libvirt API [84], an open
756 interface for VM management, as well as a command line interface (CLI). A subset
757 of this functionality is exposed to external users through a cloud interface. Due to
758 its architecture, OpenNebula is able to adapt to organizations with changing resource
759 needs, including the addition or failure of physical resources [65]. Some essential fea-
760 tures to support changing environments are the live migration and the snapshotting
761 of VMs [81]. Furthermore, when the local resources are insufficient, OpenNebula can
762 support a hybrid cloud model by using cloud drivers to interface with external clouds.
763 This lets organizations supplement the local infrastructure with computing capacity
764 from a public cloud to meet peak demands, or implement high availability strategies.
765 OpenNebula includes an EC2 driver, which can submit requests to Amazon EC2 and
766 Eucalyptus [75], as well as an ElasticHosts driver [85].

767 Regarding the storage, an OpenNebula Image Repository allows users to easily
768 specify disk images from a catalog without worrying about low-level disk configuration

769 attributes or block device mapping. Also, image access control is applied to the
770 images registered in the repository, hence simplifying multi-user environments and
771 image sharing. Nevertheless, users can also set up their own images.

772 **Others**

773 Other cloud specific projects exist such as In-VIGO [86], Cluster-on-Demand [87],
774 and VMWare's own proprietary vCloud Air [88]. Each effort provides their own
775 interpretation of private cloud services within a data center, often with the ability to
776 interplay with public cloud offerings such as Amazon's EC2. Docker [45] also looks to
777 provide IaaS capabilities with specialized and easily configurable containers, based on
778 LXC and libcontainer solutions described in the previous section. While it is still to be
779 determined how Docker and containers will change the private IaaS landscape, they
780 do provide similar functionality for Linux users without some of the complexities of
781 traditional virtualized IaaS.

782 **2.2.2 Virtual Clusters**

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

786 Let us consider a typical usage for a Cloud data center that is used in part to
787 provide computational power for the Large Hadron Collider at CERN [89], a global
788 collaboration from more than 2000 scientists of 182 institutes in 38 nations. Such a
789 system would have a small number of experiments to run. Each experiment would
790 require a very large number of jobs to complete the computation needed for the anal-
791 ysis. Examples of such experiments are the ATLAS [90] and CMS [91] projects, which
792 (combined) require Petaflops of computing power on a daily basis. Each job of an

793 experiment is unique, but the application runs are often the same. Therefore, virtual
 794 machines are deployed to execute incoming jobs. There is a file server which provides
 795 virtual machine templates. All typical jobs are preconfigured in virtual machine tem-
 796 plates. When a job arrives at the head node of the cluster, a correspondent virtual
 797 machine is dynamically started on a certain compute node within the cluster to ex-
 798 ecute the job. While the LHC project and CERN's cloud effort is a formidable one,
 799 it only covers pleasingly parallel HTC workloads, and often times HPC and big data
 800 workloads can equally complex in differing ways.

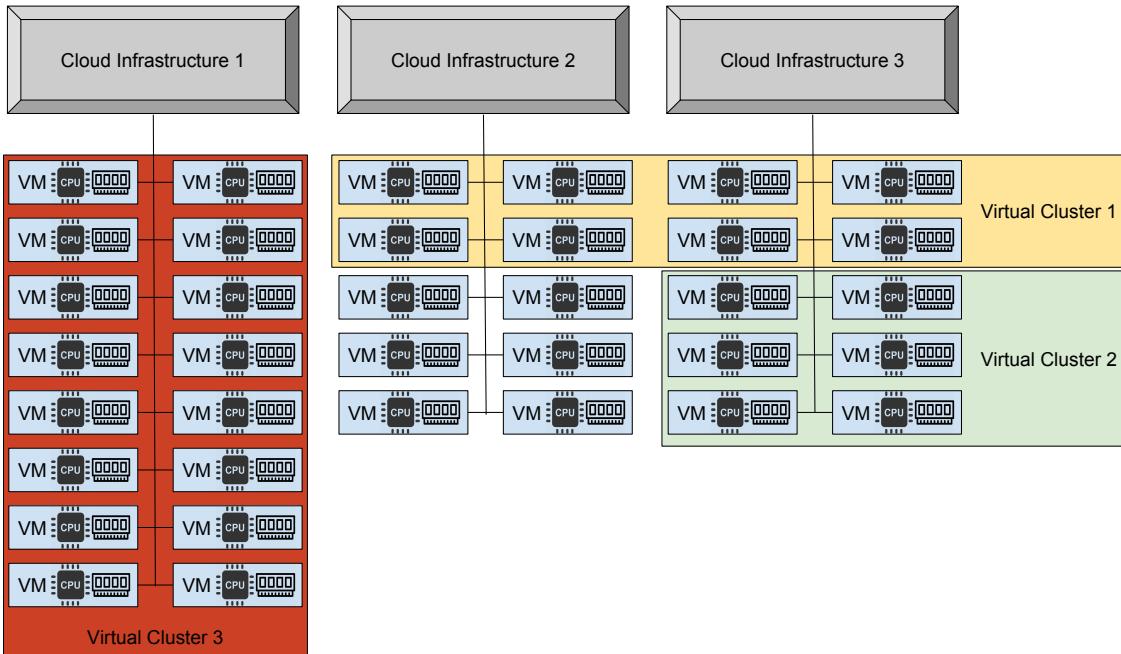


Figure 2.6 Virtual Clusters on Cloud Infrastructure

801 Cluster computing has become one of the core tools in distributed systems for
 802 use in parallel computation. Cluster computing revolves around the desire to get more
 803 computing power and better reliability by utilizing many computers together across a
 804 network to achieve larger computational tasks. Clusters have manifested themselves
 805 in many different ways, ranging from Beowulf clusters [6] which run using commodity

806 PCs to some of the TOP500 [8] supercomputing systems today. Virtual clusters rep-
807 resents the growing need of users to effectively organize computational resources in an
808 environment specific to their tasks at hand, instead of sharing a common architecture
809 across many users. With the advent of modern virtualization, virtual clusters are
810 deployed across a set of VMs in order to gain relative isolation and flexibility between
811 disjoint virtual clusters. Virtual clusters, or a set of multiple cluster computing de-
812 ployments on a single, larger physical cluster infrastructure, often have the following
813 properties and attributes [48]:

- 814 ● Resources allocation based on a VM unit
- 815 ● Clusters built of many VMs together, or by provisioning physical nodes
- 816 ● Leverage local infrastructure management tools to provide a middleware solu-
817 tion for virtual clusters
- 818 – Implementations could be a cloud IaaS such as OpenStack
- 819 – Some instances use a queueing system such as PBS
- 820 ● User experience based on virtual cluster management, not single VM manage-
821 ment
- 822 ● Consolidates functionality on a smaller resource platform using multiple VMs
- 823 ● Can provide fault tolerance through VM migration and management
- 824 ● Can utilize dynamic scaling through the addition or deletion of VMs from the
825 virtual cluster
- 826 ● 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 are 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 persistence, 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 quickly deploy custom environments becomes critical. As such, efforts have been put forth to quickly 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 [92].

Initially, virtual clusters were proposed for the use of Grid communities [28]. Specifically, Foster et al look to provide commodity clusters to various Virtual Organizations [93], whereby grid services can instantiate and deploy VMs. This design and implementation was further refined through the use of metadata and contextualization using appliances [94]. Some of these ideas have even come to take shape in larger scale supercomputing deployments, such as with SDSC Comet’s virtual cluster availability [95].

Virtual Clusters, require orchestration services to be able to organize, deploy,

853 manage, and re-play the desired user environment, and there have been a number of
854 efforts to bring this orchestration to utility. One efforts within FutureGrid is with
855 the experiment management design [39], which attempts to define how resources are
856 connected to and monitored, as well as how VMs are stored in a repository and
857 provisioned across multiple heterogeneous resources. This effort moved forward with
858 Cloudmesh [41], which provides a simple client interface to access multiple cloud
859 resources with a command-line shell interface.

860 Another virtual cluster orchestration effort has developed within the OpenStack
861 private IaaS solution itself, named OpenStack Heat [40]. Heat provides a method
862 by which you can deploy "stacks", which could essentially be virtual clusters, us-
863 ing OpenStack infrastructure. Specifically, Heat provides a human readable and
864 machine-accessible template for specifying environments and requirements, as well
865 as a RESTful API. In submitting a Heat orchestration template to the API, heat
866 will interpret and build the designed custom environment within a given OpenStack
867 cloud deployment. Kubernetes [96], a related effort, is a infrastructure orchestration
868 framework for managing containerized applications within Docker.

869 **2.2.3 The FutureGrid Project**

870 FutureGrid was a NSF-funded national-scale Grid and Cloud test-bed facility that
871 included a number of computational resources across many distributed locations.
872 This FutureGrid test-bed allowed users can evaluate differing systems for applica-
873 bility with their given research task or application. These areas include computer
874 science research topics ranging from authentication, authorization, scheduling, virtu-
875 alization, middleware design, interface design and cybersecurity, to the optimization
876 of grid-enabled and cloud-enabled computational schemes for Astronomy, Chemistry,
877 Biology, Engineering, High Energy Physics, or Atmospheric Science. This project

878 started at an opportune time, when cloud infrastructure was still in its experimental
 879 stages and its applicability to mid-tier scientific efforts were unknown.

880 The FutureGrid features a unique WAN network structure that lent itself to a
 881 multitude of experiments specifically for evaluating middleware technologies and ex-
 882 periment management services. This network can be dedicated to conduct experi-
 883 ments in isolation, using a network impairment device for introducing a variety of
 884 predetermined network conditions. Figure 2.7 depicts the geographically distributed
 885 resources that are outlined in Table 2.1 in more detail. All network links within Fu-
 886 tureGrid are dedicated 10GbE links with the exception of a shared 10GbE link to
 887 TACC over the TeraGrid [97,98] network, enabling high-speed data management and
 888 transfer between each partner site within FutureGrid.

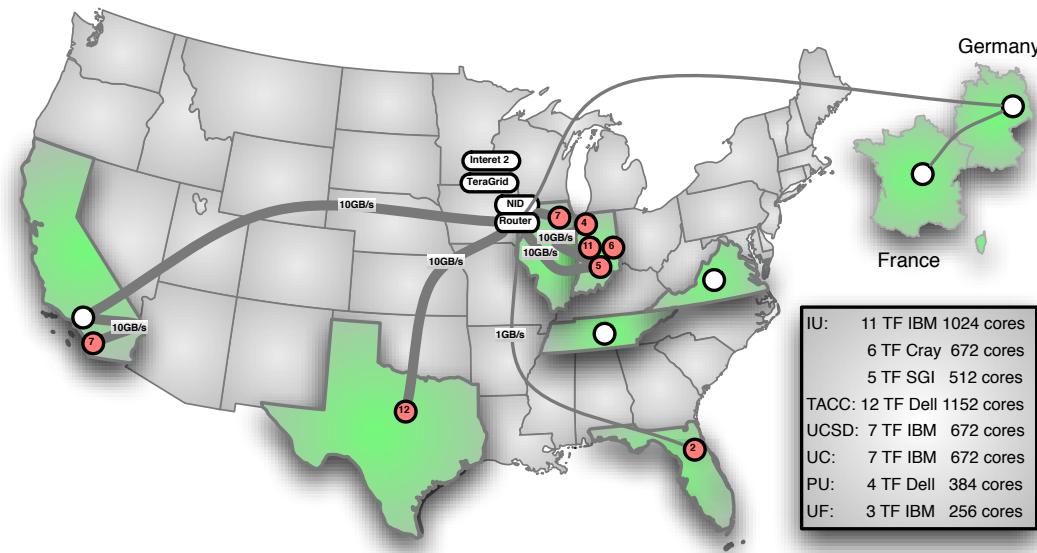


Figure 2.7 FutureGrid Participants, Network, and Resources

889 Although the total number of systems within FutureGrid is comparatively conser-
 890 vative, they provide some heterogeneity to the architecture and are connected by the
 891 high-bandwidth network links. One important feature to note is that most systems

Table 2.1 FutureGrid hardware

System type	Name	CPUs	Cores	TFLOPS	RAM(GB)	Disk(TB)	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

892 can be dynamically provisioned, e.g. these systems can be reconfigured when needed
 893 by special software that is part of FutureGrid with proper access control by users
 894 and administrators. Therefore its believed that this hardware infrastructure can fully
 895 accommodate the needs of an experiment management system.

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

902 **2.3 High Performance Computing**

903 **2.3.1 Brief History of Supercomputing**

904 Supercomputing itself can date back to some of the forefront of computing itself,
905 especially if we consider the ENIAC [99], the first Turing Complete general purpose
906 digital computer, also as the first supercomputer. ENIAC was first deployed to cal-
907 culate artillery firing tables, but was later used during the Second World War for
908 helping the Manhattan project's thermonuclear calculations and later dedicated to
909 the University of Pennsylvania after the war.

910 The first properly termed supercomputer was the Control Data Corporation's 6600
911 mainframe [100], first delivered to CERN in 1965. The CDC 6600 was notably faster
912 than the IBM counterparts, and the first deployments were able to perform on the
913 order of 1 MFLOP. Interestingly, the CPU design that came from Seymour Cray's
914 CDC 6600 took advantage of a simplified yet fast CPU design with silicon-based
915 transistors, which founded the basis of the RISC processor architecture.

916 Cray's efforts eventually lead him to start his own company, and in 1975 released
917 the Cray 1 system [101]. The Cray 1 took the powerful aspects of vector processing
918 and memory pipelining from the STAR architecture (developed later by CDC) and in-
919 troduced scalar performance through splitting vectors and instruction chaining. This
920 resulted in an overall performance of around 250 MFLOPS at peak, but realistically
921 closer to 1300 MFLOPS for general applications. The Cray 1 system also helped push
922 forward the integrated circuit design, which was finally performant to be used, and
923 also required an entirely new freon based coolant system.

924 The Cray 1 system gave way to the Cray X-MP and Y-MP in the mid 1980s.
925 These machines were shared-memory vector processors, with two processors in the
926 X-MP and up to 8 processors for the later Y-MP systems. These first shared-memory

927 systems were aided by increased memory speeds. The X-MP machine was capable
928 of 200 MFLOPS sustained and 400 MFLOPS peak performance, whereas the Y-MP
929 variants were capable of over 2 GFLOPS.

930 Also concurrently during the 1980s the advent of distributed memory architectures
931 for supercomputing were starting to emerge. Specifically work on Caltech's Cosmic
932 Cube, also known as a Hypercube, by Seitz and Fox [102,103], started the movement of
933 concurrent or parallel computing. The Cosmic Cube leveraged new VLSI techniques
934 and assembled Intel 8086/87 processors together with a novel hypercube interconnect
935 which required no switching, creating one of the first truly parallel computers. SIMD
936 programming and computation was done through a novel message passing architec-
937 ture, instead of shared variables. This design was first commercialized with Intel's
938 iPSC, and later contributing directly to designs in the Intel Paragon, ASCI Red, and
939 Cray T3D/E systems.

940 While concurrent and parallel processor supercomputers continued into the 90s
941 with aforementioned hypercube designs and the IBM Thinking Machines [104], a new
942 commodity-based strategy emerged with Becker and Sterling's Beowulf clusters [6].
943 Effectively, a Beowulf cluster is simply a cluster of commodity x86 machines linked
944 together with a simplified LAN network. Beowulf clusters often (but not always)
945 run Linux OS and leverage Ethernet solutions, and are programmed using a message
946 passing construct such as MPI [16]. This allows for the building of massively parallel
947 systems with relatively low cost and investment. Many specialized clusters today
948 still utilize commodity x86 hardware and Linux OSs similar to the original Beowulf
949 systems.

950 Concurrency and parallel computation on supercomputing resources has only
951 flourished since. This has been even more pronounced as CPU clock frequencies
952 stabilized and multi-core architectures took hold, driving the need for concurrency

953 not only at an increased rate for supercomputing, but also even for commodity sys-
954 tems. While commodity single-CPU, multi-core systems often look towards exploiting
955 shared memory parallelism, distributed memory architectures have become a way of
956 life for high performance computing.

957 2.3.2 Distributed Memory Computation

958 Abstractly, distributed memory architectures consist of multiple instances of a pro-
959 cessor, memory, and an interconnect which allows each instance to perform inde-
960 pendent computations and communicate over the interconnect. These interconnects
961 could be built using point-to-point, or through more complex and advanced switching
962 hardware, building a larger topology. While a simple example could involve several
963 commodity PCs connected through Ethernet switch, this model scales to the latest
964 supercomputing resources of today with millions of cores [105].

965 Programming such distributed memory systems is a nontrivial task that the
966 greater HPC community has been wrangling for years. In the early days, this took
967 the form of either the Message Passing Interface (MPI) [16] or PVM [106], however
968 MPI has been far more successful and dominates the market for distributed memory
969 parallel computation. MPI is a standardized message passing system, which defines
970 the semantics and syntax for writing parallel programs in C, C++, or Fortran. MPI
971 has even been implemented in other languages such as Java [107]. As MPI is a stan-
972 dardization, there are many implementations that exist, including OpenMPI [108],
973 MPICH [109], and MVAPICH [110], to name a few. Many MPI-enabled applications
974 have been shown to be, with proper and careful design, the most efficient way to run
975 a parallel application across a large subset of tightly coupled distributed resources,
976 and can often represent the status-quo for HPC applications today.

977 More recently, the MPI programming model has been modified or joined with

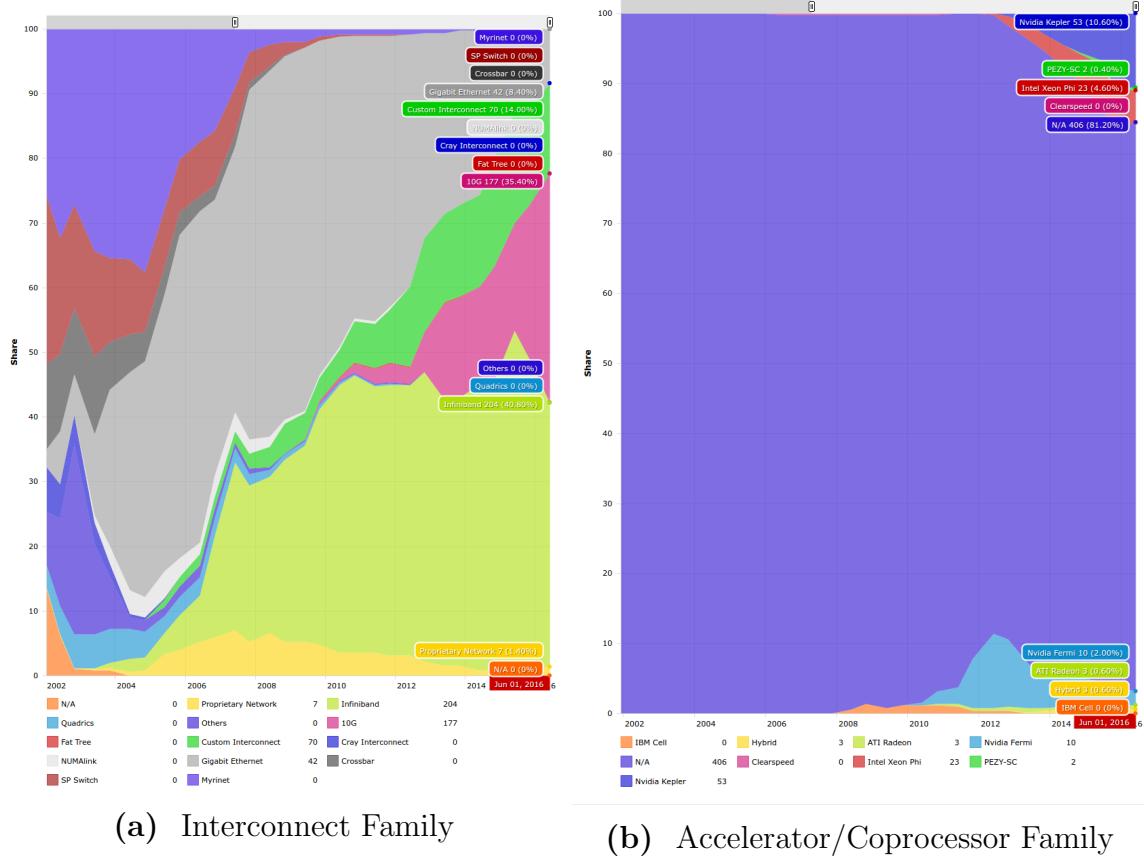


Figure 2.8 Top 500 Development over time from 2002 to 2016 [8]

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 [111], or a MPI+CUDA model which interleaves message passing with GPU utilization [112]. As some of the latest supercomputing resources have been deployed specifically with Nvidia GPUs for GPGPU programming such as the ORNL's Titan system [113], the need for distributed memory computation with GPU computing 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 can see

989 some trends emerge within hardware. Specifically looking at Interconnect and Copro-
990 cessor architectures in Figure 2.8, we see a notable jump in the number of deployed
991 system that are using both InfiniBand and GPUs in the past decade. Specifically, In-
992 finiBand usage has increased to roughly 40% of the total number of deployed systems
993 and remained somewhat stable as an interconnect family. Concurrently, the use of
994 accelerators has increased from only 1 in 500 systems a decade ago, to almost a 20%
995 of the top 500 systems that are using coprocessors. Within that factor, the majority
996 of such accelerator-equipped systems have been using GPUs, with a concurrent in-
997 crease in the Intel Xeon Phi coprocessor as well. While these do not represent all of
998 supercomputing nor exclusively the most high end of systems, they do represent ad-
999 vanced hardware that has been increasingly common in the last decade, yet relatively
1000 underutilized in comparison within cloud infrastructure. As such, much of the effort
1001 in this dissertation focuses on, but is not limited to, these two technology families.

1002 **2.3.3 Exascale**

1003 As the forefront of supercomputing moves beyond the latest petaflop machines in the
1004 past few years [113], the HPC community is setting their sights on the next signifi-
1005 cant milestone: Exascale. Exascale computing refers broadly to performing roughly
1006 one exaFLOPS, or 10^{18} floating point operations per second. However, exascale itself
1007 is far more than just a theoretical FLOPS goal, but instead a set of new comput-
1008 ing advancements and challenges that requires reaching computational power at such
1009 magnitude. While FLOPs are often used as the ubiquitous yard stick for supercom-
1010 puting with the LINPACK benchmark [114], other efforts have taken hold to classify
1011 systems under a different set of parameters [115, 116], with the loose understanding
1012 that these may incorporate a richer application set destined for exascale systems. This
1013 could include for instance integer calculations at a similar scale to satisfy defence and

1014 intelligence perspectives, or graph processing at a

1015 With exascale, there are a number of barriers that exist with current technolo-
1016 gies that must be overcome to reach exascale. The exascale Computing Study [35]
1017 specifically lists 4 major focal areas:

1018 1. Energy and Power Challenge

1019 • Describes the physical difficulties in providing the amount of power needed
1020 to drive a sufficiently large exascale system. The US DOE estimates the
1021 maximum power envelope for a deployed first exascale system to be within
1022 20-40MW. Extrapolating current technology power utilization shows an
1023 order of magnitude more energy utilization than the specificity power en-
1024 envelop. As such, new architectures and conversation techniques will need
1025 to be investigated.

1026 2. Memory and Storage Challenge

1027 • This challenge illustrates the problem that has grown in relation to the
1028 memory wall, defined by the exponential difference between processor and
1029 memory performance, as well as the storage capacity limits to support
1030 calculations at the level of performance necessary. This challenge incorpo-
1031 rates not only main memory limitations, but also tertiary storage issues as
1032 well.

1033 3. Concurrency and Storage Challenge

1034 • This challenge is born from the recent limit in CPU clock rates as a way
1035 to gain performance. Instead, performance must be gained through paral-
1036 lelism. The depth of this challenge is especially profound when we consider
1037 parallelism on the order of millions, if not billions of threads.

1038 4. Resiliency Challenge

- 1039 • The resiliency challenge defines the necessity of computation to recover
1040 and continue in the event of a fault or fluctuation. As parallelism and
1041 the number of individualized components substantially increases in a path
1042 towards exascale, the mean time to failure of any given component also
1043 increases.

1044 While current exascale efforts are as wide as they are varying, not only with
1045 concepts, architectures, and runtime systems, but also with deployment plans and
1046 expectations between future deployments. Of particular interest in current exas-
1047 scale research is in Operating System and runtime (OS/R) developments to support
1048 new extreme-scale applications in an efficient manner. Two examples of novel OS
1049 approaches are the Hobbes project [117] and the ARGO Exascale Operating Sys-
1050 tem [118]. These OS efforts, along with novel programming models for exascale such
1051 as ParalleX [119] look to fundamentally change the relationship between HPC hard-
1052 ware architectures and the libraries and applications to be leveraged on such future
1053 exascale deployments.

1054 It is possible that virtualization itself may have an impact in OS and runtime ser-
1055 vices in exascale [117]. While some of the work herein may be tangentially of utility
1056 to such efforts, the immediate goal of this dissertation is not to investigate the appli-
1057 cability of virtualization for exascale systems, but rather to enable the diversification
1058 of HPC towards cloud infrastructure.

1059 **Chapter 3**

1060 **Analysis of Virtualization**

1061 **Technologies for High Performance**

1062 **Computing Environments**

1063 **3.1 Abstract**

1064 As Cloud computing emerges as a dominant paradigm in distributed systems, it is
1065 important to fully understand the underlying technologies that make Clouds possible.
1066 One technology, and perhaps the most important, is virtualization. Recently virtual-
1067 ization, through the use of hypervisors, has become widely used and well understood
1068 by many. However, there are a large spread of different hypervisors, each with their
1069 own advantages and disadvantages. This chapter provides an in-depth analysis of
1070 some of today's commonly accepted virtualization technologies from feature com-
1071 parison to performance analysis, focusing on the applicability to High Performance
1072 Computing environments using FutureGrid resources. The results indicate virtualiza-
1073 tion sometimes introduces slight performance impacts depending on the hypervisor

1074 type, however the benefits of such technologies are profound and not all virtualization
1075 technologies are equal.

1076 **3.2 Introduction**

1077 Cloud computing [59] is one of the most explosively expanding technologies in the
1078 computing industry today. A Cloud computing implementation typically enables
1079 users to migrate their data and computation to a remote location with some varying
1080 impact on system performance [60]. This provides a number of benefits which could
1081 not otherwise be achieved.

1082 Such benefits include:

- 1083 ● *Scalability* - Clouds are designed to deliver as much computing power as any
1084 user needs. While in practice the underlying infrastructure is not infinite, the
1085 cloud resources are projected to ease the developer's dependence on any specific
1086 hardware.
- 1087 ● *Quality of Service (QoS)* - Unlike standard data centers and advanced com-
1088 puting resources, a well-designed Cloud can project a much higher QoS than
1089 traditionally possible. This is due to the lack of dependence on specific hard-
1090 ware, so any physical machine failures can be mitigated without the prerequisite
1091 user awareness.
- 1092 ● *Customization* - Within a Cloud, the user can utilize customized tools and
1093 services to meet their needs. This can be to utilize the latest library, toolkit, or
1094 to support legacy code within new infrastructure.
- 1095 ● *Cost Effectiveness* - Users finds only the hardware required for each project.
1096 This reduces the risk for institutions potentially want build a scalable system,

1097 thus providing greater flexibility, since the user is only paying for needed in-
1098 frastructure while maintaining the option to increase services as needed in the
1099 future.

- 1100 • *Simplified Access Interfaces* - Whether using a specific application, a set of
1101 tools or Web services, Clouds provide access to a potentially vast amount of
1102 computing resources in an easy and user-centric way.

1103 While Cloud computing has been driven from the start predominantly by the in-
1104 dustry through Amazon [67], Google [120] and Microsoft [121], a shift is also occurring
1105 within the academic setting as well. Due to the many benefits, Cloud computing is
1106 becoming immersed in the area of High Performance Computing (HPC), specifically
1107 with the deployment of scientific clouds [122] and virtualized clusters [28].

1108 There are a number of underlying technologies, services, and infrastructure-level
1109 configurations that make Cloud computing possible. One of the most important
1110 technologies is virtualization. Virtualization, in its simplest form, is a mechanism to
1111 abstract the hardware and system resources from a given Operating System. This is
1112 typically performed within a Cloud environment across a large set of servers using a
1113 Hypervisor or Virtual Machine Monitor (VMM), which lies in between the hardware
1114 and the OS. From the hypervisor, one or more virtualized OSs can be started concur-
1115 rently, leading to one of the key advantages of Cloud computing. This, along with the
1116 advent of multi-core processors, allows for a consolidation of resources within any data
1117 center. From the hypervisor level, Cloud computing middleware is deployed atop the
1118 virtualization technologies to exploit this capability to its maximum potential while
1119 still maintaining a given QoS and utility to users.

1120 The rest of this chapter is as follows: First, we look at what virtualization is,
1121 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 [123]. 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) [124] is probably the most popular Cloud and is used extensively in the IT industry to this day. Nimbus [125] and Eucalyptus [75] 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 [126], In-VIGO [127], and Cluster-on-Demand [87], all of which leverage one or more hypervisors to provide computing infrastructure on demand. In recent history, OpenStack [128] has also come to light from a joint collaboration between NASA and Rackspace which also provide compute and storage

1146 resources in the form of a Cloud.

1147 While there are currently a number of virtualization technologies available today,
1148 the virtualization technique of choice for most open platforms over the past 5 years has
1149 typically been the Xen hypervisor [49]. However more recently VMWare ESX [129]
1150 ¹, Oracle VirtualBox [130] and the Kernel-based Virtual Machine (KVM) [53] are
1151 becoming more commonplace. As these look to be the most popular and feature-
1152 rich of all virtualization technologies, we look to evaluate all four to the fullest extent
1153 possible. There are however, numerous other virtualization technologies also available,
1154 including Microsoft's Hyper-V [131], Parallels Virtuozzo [132], QEMU [133], OpenVZ
1155 [134], Oracle VM [135], and many others. However, these virtualization technologies
1156 have yet to seen widespread deployment within the HPC community, at least in their
1157 current form, so they have been placed outside the scope of this work.

1158 In recent history there have actually been a number of comparisons related to
1159 virtualization technologies and Clouds. The first performance analysis of various hy-
1160 pervisors started with, unsurprisingly, the hypervisor vendors themselves. VMWare
1161 has happy to put out its own take on performance in [136], as well as the original
1162 Xen article [49] which compares Xen, XenoLinux, and VMWare across a number of
1163 SPEC and normalized benchmarks, resulting in a conflict between both works. From
1164 here, a number of more unbiased reports originated, concentrating on server consol-
1165 idation and web application performance [129, 137, 138] with fruitful yet sometimes
1166 incompatible results. A feature base survey on virtualization technologies [139] also
1167 illustrates the wide variety of hypervisors that currently exist. Furthermore, there
1168 has been some investigation into the performance within HPC, specifically with In-
1169 finiBand performance of Xen [140] and rather recently with a detailed look at the
1170 feasibility of the Amazon Elastic Compute cloud for HPC applications [44], however

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

1171 both works concentrate only on a single deployment rather than a true comparison
1172 of technologies.

1173 As these underlying hypervisor and virtualization implementations have evolved
1174 rapidly in recent years along with virtualization support directly on standard x86
1175 hardware, it is necessary to carefully and accurately evaluate the performance impli-
1176 cations of each system. Hence, we conducted an investigation of several virtualization
1177 technologies, namely Xen, KVM, VirtualBox, and in part VMWare. Each hypervisor
1178 is compared alongside one another with base-metal as a control and (with the exception
1179 of VMWare) run through a number of High Performance benchmarking tools.

1180 3.4 Feature Comparison

1181 With the wide array of potential choices of virtualization technologies available, its
1182 often difficult for potential users to identify which platform is best suited for their
1183 needs. In order to simplify this task, we provide a detailed comparison chart between
1184 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

1185 The first point of investigation is the virtualization method of each VM. Each

1186 hypervisor supports full virtualization, which is now common practice within most
1187 x86 virtualization deployments today. Xen, originating as a para-virtualized VMM,
1188 still supports both types, however full virtualization is often preferred as it does
1189 not require the manipulation of the guest kernel in any way. From the Host and
1190 Guest CPU lists, we see that x86 and, more specifically, x86-64/amd64 guests are all
1191 universally supported. Xen and KVM both support Itanium-64 architectures for full
1192 virtualization (due to both hypervisors dependency on QEMU), and KVM also claims
1193 support for some recent PowerPC architectures. However, we concern ourselves only
1194 with x86-64 features and performance, as other architectures are out of the scope of
1195 this manuscript. Of the x86-64 platforms, KVM is the only hypervisor to require
1196 either Intel VT-X or AMD-V instruction sets in order to operate. VirtualBox and
1197 VMWare have internal mechanisms to provide full virtualization even without the
1198 virtualization instruction sets, and Xen can default back to para-virtualized guests.

1199 Next, we consider the host environments for each system. As Linux is the pri-
1200 mary OS type of choice within HPC deployments, its key that all hypervisors sup-
1201 port Linux as a guest OS, and also as a host OS. As VMWare ESX is meant to be
1202 a virtualization-only platform, it is built upon a specially configured Linux/UNIX
1203 proprietary OS specific to its needs. All other hypervisors support Linux as a host
1204 OS, with VirtualBox also supporting Windows, as it was traditionally targeted for
1205 desktop-based virtualization. However, as each hypervisor uses VT-X or AMD-V in-
1206 structions, each can support any modern OS targeted for x86 platforms, including all
1207 variants of Linux, Windows, and UNIX.

1208 While most hypervisors have desirable host and guest OS support, hardware sup-
1209 port within a guest environment varies drastically. Within the HPC environment,
1210 virtual CPU (vCPU) and maximum VM memory are critical aspects to choosing the
1211 right virtualization technology. In this case, Xen is the first choice as it supports up

1212 to 128 vCPUs and can address 4TB of main memory in 64-bit modes, more than
1213 any other. VirtualBox, on the other hand, supports only 32 vCPUs and 16GB of
1214 addressable RAM per guest OS, which may lead to problems when looking to deploy
1215 it on large multicore systems. KVM also faces an issue with the number of vCPU
1216 supported limited to 16, recent reports indicate it is only a soft limit [141], so deploy-
1217 ing KVM in an SMP environment may not be a significant hurdle. Furthermore, all
1218 hypervisors provide some 3D acceleration support (at least for OpenGL) and support
1219 live migration across homogeneous nodes, each with varying levels of success.

1220 Another vital juxtaposition of these virtualization technologies is the license agree-
1221 ments for its applicability within HPC deployments. Xen, KVM, and VirtualBox are
1222 provided for free under the GNU Public License (GPL) version 2, so they are open
1223 to use and modification by anyone within the community, a key feature for many
1224 potential users. While VirtualBox is under GPL, it has recently also offered with
1225 additional features under a more proprietary license dictated by Oracle since its ac-
1226 quirement from Sun last year. VMWare, on the other hand, is completely proprietary
1227 with an extremely limited licensing scheme that even prevents the authors from will-
1228 fully publishing any performance benchmark data without specific and prior approval.
1229 As such, we have neglected VMWare form the remainder of this chapter. Whether
1230 going with a proprietary or open source hypervisor, support can be acquired (usually
1231 for an additional cost) with ease from each option.

1232 **3.4.1 Usability**

1233 While side by side feature comparison may provide crucial information about a poten-
1234 tial user's choice of hypervisor, that may also be interested in its ease of installation
1235 and use. We will take a look at each hypervisor from two user perspectives, a systems
1236 administrator and normal VM user.

1237 One of the first things on any system administrator's mind on choosing a hypervi-
1238 sor is the installation. For all of these hypervisors, installation is relatively painless.
1239 For the FutureGrid support group, KVM and VirualBox are the easiest of the all
1240 tested hypervisors to install, as there are a number of supported packages available
1241 and installation only requires the addition of one or more kernel modules and the sup-
1242 port software. Xen, while still supported in binary form by many Linux distributions,
1243 is actually much more complicated. This is because Xen requires a full modification
1244 to the kernel itself, not just a module. Loading a new kernel into the boot process
1245 which may complicate patching and updating later in the system's maintenance cycle.
1246 VMWare ESX, on the other hand, is entirely separate from most other installations.
1247 As previously noted, ESX is actually a hypervisor and custom UNIX host OS com-
1248 bined, so installation of ESX is likewise to installing any other OS from scratch. This
1249 may be either desirable or adverse, depending on the system administrator's usage of
1250 the systems and VMWare's ability to provide a secure and patched environment.

1251 While system administrators may be concerned with installation and maintenance,
1252 VM users and Cloud developers are more concerned with daily usage. The first thing
1253 to note about all of such virtualiation technologies is they are supported (to some
1254 extent) by the libvirt API [142]. Libvirt is commonly used by many of today's IaaS
1255 Cloud offerings, including Nimbus, Eucalyptus, OpenNebula and OpenStack. As
1256 such, the choice of hypervisor for Cloud developer's is less of an issue, so long as
1257 the hypervisor supports the features they desire. For individual command line usage
1258 of each tool, it varies quite a bit more. Xen does provide their own set of tools for
1259 controlling and monitoring guests, and seem to work relatively well but do incur a
1260 slight learning curve. KVM also provides its own CLI interface, and while it is often
1261 considered less cumbersome it provides less advanced features directly to users, such as
1262 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) [143] 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.



Figure 3.2 FutureGrid Participants and Resources

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 become the users' choice toolkit. FG provides the ability to compare these frameworks with each other while considering real scientific applications [39]. 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

1300 transition from one environment to the other. These interests and research objectives
1301 make the FutureGrid project the perfect match for this work. Furthermore, we expect
1302 that the results gleaned from this chapter will have a direct impact on the FutureGrid
1303 deployment itself.

1304 **3.5.2 Experimental Environment**

1305 Currently, one of FutureGrid's latest resources is the *India* system, a 256 CPU IBM
1306 iDataPlex machine consisting of 1024 cores, 2048 GB of ram, and 335 TB of storage
1307 within the Indiana University Data Center. In specific, each compute node of India
1308 has two Intel Xeon 5570 quad core CPUs running at 2.93Ghz, 24GBs of Ram, and a
1309 QDR InfiniBand connection. A total of four nodes were allocated directly from India
1310 for these experiments. All were loaded with a fresh installation of Red Hat Enterprise
1311 Linux server 5.5 x86_64 with the 2.6.18-194.8.1.el5 kernel patched. Three of the four
1312 nodes were installed with different hypervisors; Xen version 3.1, KVM (build 83), and
1313 VirtualBox 3.2.10, and the forth node was left as-is to act as a control for bare-metal
1314 native performance.

1315 Each guest virtual machine was also built using Red Hat EL server 5.5 running
1316 an unmodified kernel using full virtualization techniques. All tests were conducted
1317 giving the guest VM 8 cores and 16GB of ram to properly span a compute node.
1318 Each benchmark was run a total of 20 times, with the results averaged to produce
1319 consistent results, unless indicated otherwise.

1320 **3.5.3 Benchmarking Setup**

1321 As this chapter aims to objectively evaluate each virtualization technology from a
1322 side-by-side comparison as well as from a performance standpoint, the selection of

1323 benchmarking applications is critical.

1324 The performance comparison of each virtual machine is based on two well known
1325 industry standard performance benchmark suites; HPCC and SPEC. These two
1326 benchmark environments are recognized for their standardized reproducible results in
1327 the HPC communit, and the National Science Foundation (NSF), Department of En-
1328 ergy (DOE), and DARPA are all sponsors of the HPCC benchmarks. The following
1329 benchmarks provide a means to stress and compare processor, memory, inter-process
1330 communication, network, and overall performance and throughput of a system. These
1331 benchmarks were selected due to their importance to the HPC community sinse they
1332 are often directly correlated with overall application performance [144].

1333 **HPCC Benchmarks**

1334 The HPCC Benchmarks [145, 146] are an industry standard for performing bench-
1335 marks for HPC systems. The benchmarks are aimed at testing the system on multiple
1336 levels to test their performance. It consists of 7 different tests:

- 1337 ● *HPL* - The Linpack TPP benchmark measures the floating point rate of exe-
1338 cution for solving a linear system of equations. This benchmark is perhaps the
1339 most important benchmark within HPC today, as it is the basis of evaluation
1340 for the Top 500 list [8].
- 1341 ● *DGEMM* - Measures the floating point rate of execution of double precision real
1342 matrix-matrix multiplication.
- 1343 ● *STREAM* - A simple synthetic benchmark program that measures sustainable
1344 memory bandwidth (in GB/s) and the corresponding computation rate for sim-
1345 ple vector kernel.

- 1346 ● *PTRANS* - Parallel matrix transpose exercises the communications where pairs
1347 of processors communicate with each other simultaneously. It is a useful test of
1348 the total communications capacity of the network.

- 1349 ● *RandomAccess* - Measures the rate of integer random updates of memory (GUPS).

- 1350 ● *FFT* - Measures the floating point rate of execution of double precision complex
1351 one-dimensional Discrete Fourier Transform (DFT).

- 1352 ● *Communication bandwidth and latency* - A set of tests to measure latency and
1353 bandwidth of a number of simultaneous communication patterns; based on b_eff
1354 (effective bandwidth benchmark).

1355 This benchmark suite uses each test to stress test the performance on multiple
1356 aspects of the system. It also provides reproducible results which can be verified by
1357 other vendors. This benchmark is used to create the Top 500 list [8] which is the list
1358 of the current top supercomputers in the world. The results that are obtained from
1359 these benchmarks provide an unbiased performance analysis of the hypervisors. Our
1360 results provide insight on inter-node PingPong bandwidth, PingPong latency, and
1361 FFT calculation performance.

1362 **SPEC Benchmarks**

1363 The Standard Performance Evaluation Corporation (SPEC) [147, 148] is the other
1364 major standard for evaluation of benchmarking systems. SPEC has several different
1365 testing components that can be utilized to benchmark a system. For our benchmark-
1366 ing comparison we will use the SPEC OMP2001 because it appears to represent a
1367 vast array of new and emerging parallel applications while simultaneously providing
1368 a comparison to other SPEC benchmarks. SPEC OMP continues the SPEC tradi-

1369 tion of giving HPC users the most objective and representative benchmark suite for
1370 measuring the performance of SMP (shared memory multi-processor) systems.

- 1371 ● The benchmarks are adapted from SPEC CPU2000 and contributions to its
1372 search program.
- 1373 ● The focus is to deliver systems performance to real scientific and engineering
1374 applications.
- 1375 ● The size and runtime reflect the needs of engineers and researchers to model
1376 large complex tasks.
- 1377 ● Two levels of workload characterize the performance of medium and large sized
1378 systems.
- 1379 ● Tools based on the SPEC CPU2000 toolset make these the easiest ever HPC
1380 tests to run.
- 1381 ● These benchmarks place heavy demands on systems and memory.

1382 **3.6 Performance Comparison**

1383 The goal of this chapter is to effectively compare and contrast the various virtual-
1384 ization technologies, specifically for supporting HPC-based Clouds. The first set of
1385 results represent the performance of HPCC benchmarks. Each benchmark was run
1386 a total of 20 times, and the mean values taken with error bars represented using the
1387 standard deviation over the 20 runs. The benchmarking suite was built using the Intel
1388 11.1 compiler, uses the Intel MPI and MKL runtime libraries, all set with defaults
1389 and no optimizations whatsoever.

1390 We open first with High Performance Linpack (HPL), the de-facto standard for
1391 comparing resources. In Figure 3.3, we can see the comparison of Xen, KVM, and
1392 Virtual Box compared to native bare-metal performance. First, we see that native
1393 is capable of around 73.5 Gflops which, with no optimizations, achieves 75% of the
1394 theoretical peak performance. Xen, KVM and VirtualBox perform at 49.1, 51.8 and
1395 51.3 Gflops, respectively when averaged over 20 runs. However Xen, unlike KVM
1396 and VirtualBox, has a high degree of variance between runs. This is an interesting
1397 phenomenon for two reasons. First, this may impact performance metrics for other
1398 HPC applications and cause errors and delays between even pleasingly-parallel appli-
1399 cations and add to reducer function delays. Second, this wide variance breaks a key
1400 component of Cloud computing providing a specific and predefined quality of service.
1401 If performance can sway as widely as what occurred for Linpack, then this may have
1402 a negative impact on users.

1403 Next, we turn to another key benchmark within the HPC community, Fast Fourier
1404 Transforms (FFT). Unlike the synthetic Linpack benchmark, FFT is a specific, pur-
1405 poseful benchmark which provides results which are often regarded as more relative
1406 to a user's real-world application than HPL. From Figure 3.4, we can see rather dis-
1407 tinct results from what was previously provided by HPL. Looking at Star and Single
1408 FFT, its clear performance across all hypervisors is roughly equal to bare-metal per-
1409 formance, a good indication that HPC applications may be well suited for use on
1410 VMs. The results for MPI FFT also show similar results, with the exception of Xen,
1411 which has a decreased performance and high variance as seen in the HPL benchmark.
1412 Our current hypothesis is that there is an adverse affect of using Intel's MPI runtime
1413 on Xen, however the investigation is still ongoing.

1414 Another useful benchmark illustrative of real-world performance between bare-
1415 metal performance and various hypervisors are the ping-pong benchmarks. These

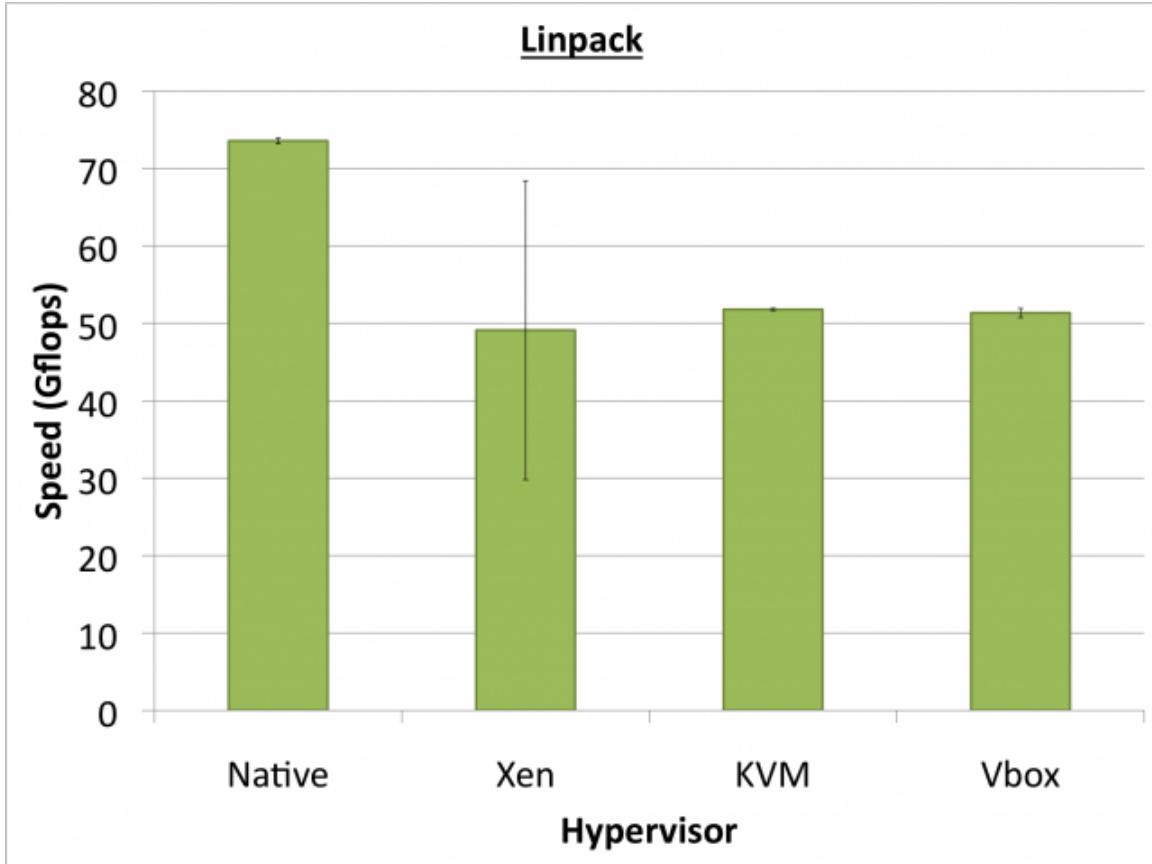


Figure 3.3 Linpack performance

1416 benchmarks measure the bandwidth and latency of passing packets between multiple
 1417 CPUs. With this experiment, all ping-pong latencies are kept within a given node,
 1418 rather than over the network. This is done to provide further insight into the CPU and
 1419 memory overhead within each hypervisor. From Figure 3.5 the intranode bandwidth
 1420 performance is uncovered, with some interesting distinctions between each hypervi-
 1421 sor. First, Xen performs, on average, close to native speeds, which is promising for
 1422 the hypervisor. KVM, on the other hand, shows consistent overhead proportional
 1423 to native performance across minimum, average, and maximum bandwidth. Virtu-
 1424 alBox, on the other hand, performs well, in fact too well to the point that raises
 1425 alarm. While the minimum and average bandwidths are within native performance,
 1426 the maximum bandwidth reported by VirtualBox is significantly greater than native

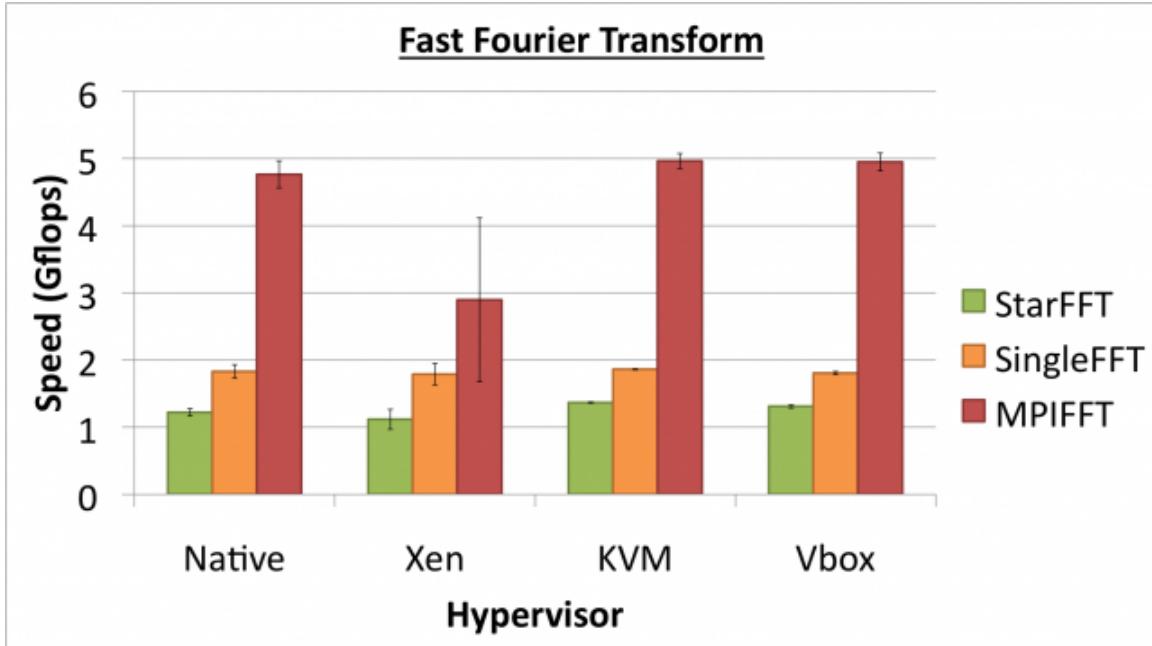


Figure 3.4 Fast Fourier Transform performance

1427 measurements, with a large variance. After careful examination, it appears this is
 1428 due to how VirtualBox assigns its virtual CPUs. Instead of locking a virtual CPU to
 1429 a real CPU, a switch may occur which could benefit on the off-chance the two CPU's
 1430 in communication between a ping-pong test could in fact be the same physical CPU.
 1431 The result would mean the ping-pong packet would remain in cache and result in a
 1432 higher perceived bandwidth than normal. While this effect may be beneficial for this
 1433 benchmark, it may only be an illusion towards the real performance gleaned from the
 1434 VirtualBox hypervisor.

1435 The Bandwidth may in fact be important within the ping-pong benchmark, but
 1436 the latency between each ping-pong is equally useful in understanding the perfor-
 1437 mance impact of each virtualization technology. From Figure 3.6, we see KVM and
 1438 VirtualBox have near-native performance; another promising result towards the util-
 1439 ity of hypervisors within HPC systems. Xen, on the other hand, has extremely high
 1440 latencies, especially at for maximum latencies, which in turn create a high variance

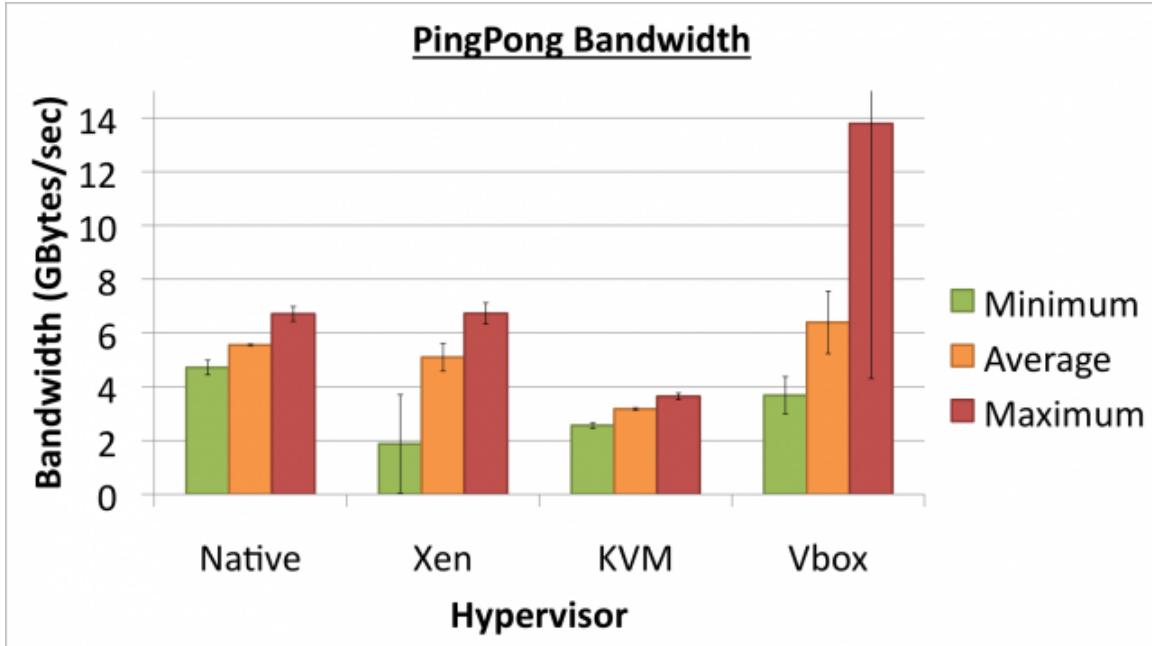


Figure 3.5 Ping Pong bandwidth performance

¹⁴⁴¹ within the average latency within the VM's performance.

¹⁴⁴² 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. ¹⁴⁴⁵ 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. ¹⁴⁴⁶ KVM performance comes close with a score of 34384, which is so similar to the native ¹⁴⁴⁷ performance that most users will never notice the difference. Xen and VirtualBox ¹⁴⁴⁸ both perform notably slower with scores of 31824 and 31695, respectively, however ¹⁴⁴⁹ this is only an 8% performance drop compared to native speeds. Further results can ¹⁴⁵⁰ be found on the SPEC website [149].

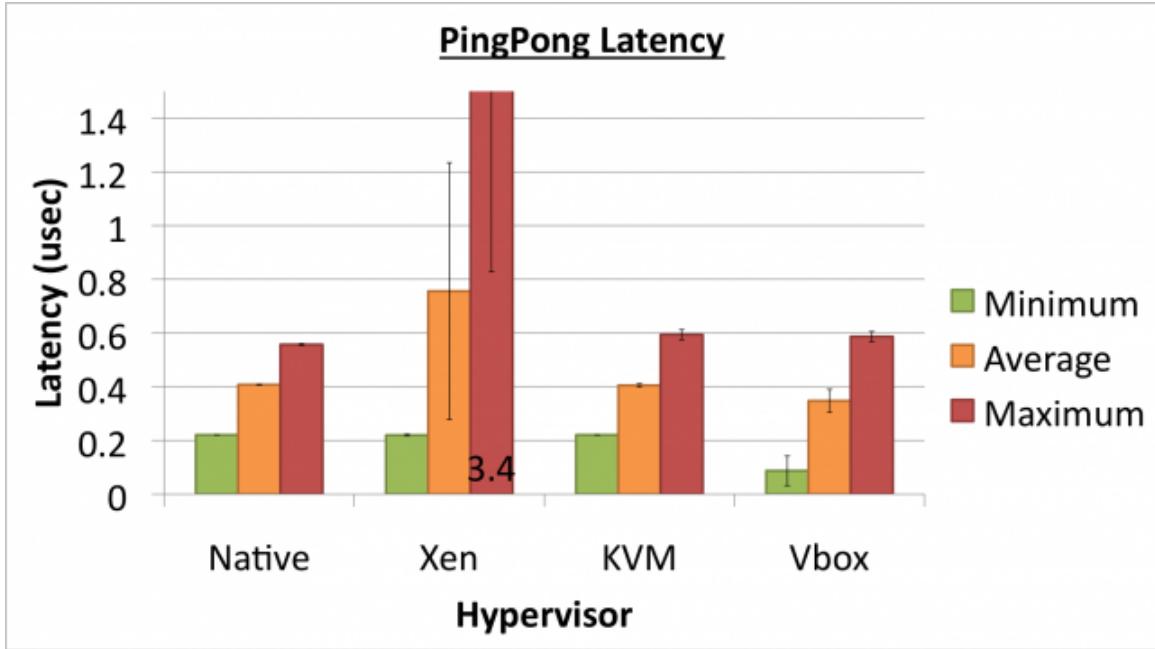


Figure 3.6 Ping Pong latency performance (lower is better)

1453 3.7 Discussion

1454 The primary goal of this chapter is to evaluate the viability of virtualization within
 1455 HPC. After our analysis, the answer seems to be a resounding "yes." However, we
 1456 also hope to select the best virtualization technology for such an HPC environment.

1457 In order to do this, we combine the feature comparison along with the performance
 1458 results, and evaluate the potential impact within the FutureGrid testbed.

1459 From a feature standpoint, most of today's virtualization technologies fit the bill
 1460 for at least small scale deployment, including VMWare. In short, each support Linux
 1461 x86_64 platforms, use VT-X technology for full virtualization, and support live mi-
 1462 gration. Due to VMWare's limited and costly licensing, it is immediately out of
 1463 contention for most HPC deployments. From a CPU and memory standpoint, Xen
 1464 seems to provide the best expandability, supporting up to 128 cpus and 4TB of ad-
 1465 dressable RAM. So long as KVM's vCPU limit can be extended, it too shows promise

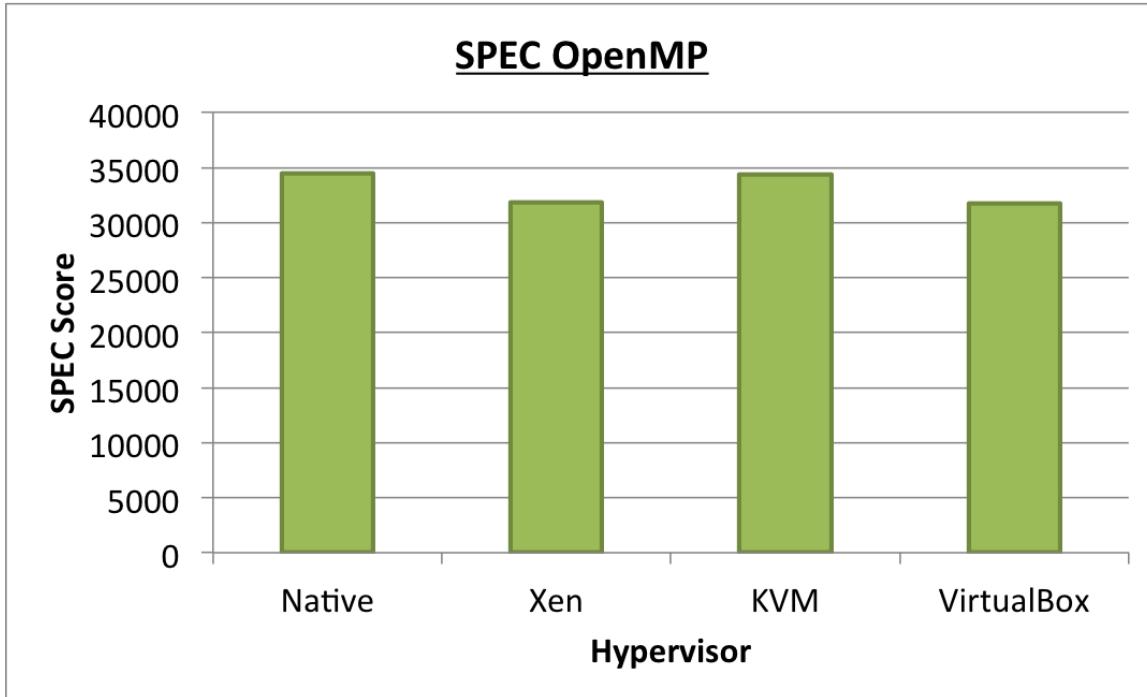


Figure 3.7 Spec OpenMP performance

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.

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

	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)

1479 through the SPEC OMP benchmarking suite also shows promising results for the use
 1480 of hypervisors in general, with KVM taking a clear lead by almost matching native
 1481 speeds.

1482 While Xen is typically regarded as the most widely used hypervisor, especially
 1483 within academic clouds and grids, it's performance has shown lack considerably when
 1484 compared to either KVM or VirtualBox. In particular, Xen's wide and unexplained
 1485 fluctuations in performance throughout the series of benchmarks suggests that Xen
 1486 may not be the best choice for building a lasting quality of service infrastructure upon.

1487 From Figure 3.8, KVM rates the best across all performance benchmarks, making it
 1488 the optimal choice for *general* deployment in an HPC environment. Furthermore,
 1489 this work's illustration of the variance in performance among each benchmark and
 1490 the applicability of each benchmark towards new applications may make possible the
 1491 ability to preemptively classify applications for accurate prediction towards the ideal
 1492 virtualized Cloud environment. We hope to further investigate this concept through
 1493 the use of the FutureGrid experiment management framework at a later date.

1494 In summary, it is the authors' projection that KVM is the best overall choice for

1495 use within HPC Cloud environments. KVM's feature-rich experience and near-native
1496 performance makes it a natural fit for deployment in an environment where usability
1497 and performance are paramount. Within the FutureGrid project specifically, we hope
1498 to deploy the KVM hypervisor across our Cloud platforms in the near future, as it
1499 offers clear benefits over the current Xen deployment. Furthermore, we expect these
1500 findings to be of great importance to other public and private Cloud deployments, as
1501 system utilization, Quality of Service, operating cost, and computational efficiency
1502 could all be improved through the careful evaluation of underlying virtualization
1503 technologies.

1504 **Chapter 4**

1505 **Evaluating GPU Passthrough in
1506 Xen for High Performance Cloud
1507 Computing**

1508 **4.1 Abstract**

1509 With the advent of virtualization and Infrastructure-as-a-Service (IaaS), the broader
1510 scientific computing community is considering the use of clouds for their technical
1511 computing needs. This is due to the relative scalability, ease of use, advanced user
1512 environment customization abilities clouds provide, as well as many novel comput-
1513 ing paradigms available for data-intensive applications. However, there is concern
1514 about a performance gap that exists between the performance of IaaS when com-
1515 pared to typical high performance computing (HPC) resources, which could limit the
1516 applicability of IaaS for many potential scientific users.

1517 Most recently, general-purpose graphics processing units (GPGPUs or GPUs) have
1518 become commonplace within high performance computing. We look to bridge the

1519 gap between supercomputing and clouds by providing GPU-enabled virtual machines
1520 (VMs) and investigating their feasibility for advanced scientific computation. Specif-
1521 ically, the Xen hypervisor is utilized to leverage specialized hardware-assisted I/O
1522 virtualization and PCI passthrough in order to provide advanced HPC-centric Nvidia
1523 GPUs directly in guest VMs. This methodology is evaluated by measuring the per-
1524 formance of two Nvidia Tesla GPUs within Xen VMs and comparing to bare-metal
1525 hardware. Results show PCI passthrough of GPUs within virtual machines is a vi-
1526 able use case for many scientific computing workflows, and could help support high
1527 performance cloud infrastructure in the near future.

1528 **4.2 Introduction**

1529 Cloud computing [4] has established itself as a prominent paradigm within the realm
1530 of Distributed Systems [150] in a very short period of time. Clouds are an internet-
1531 based solution that provide computational and data models for utilizing resources,
1532 which can be accessed directly by users on demand in a uniquely scalable way. Cloud
1533 computing functions by providing a layer of abstraction on top of base hardware
1534 to enable a new set of features that are otherwise intangible or intractable. These
1535 benefits and features include Scalability, a guaranteed Quality of Service (QoS), cost
1536 effectiveness, and direct user customization via a simplified user interface [60].

1537 While the origin of cloud computing is based in industry through solutions such
1538 as Amazon's EC2 [151], Google's MapReduce [152], and Microsoft's Azure [153], the
1539 paradigm has since become integrated in all areas of science and technology. Most
1540 notably, there is an increasing effort within the High Performance Computing (HPC)
1541 community to leverage the utility of clouds for advanced scientific computing to solve
1542 a number of challenges still standing in the field. This can be clearly seen in large-

scale efforts such as the FutureGrid project [154], the Magellan project [43], and through various other Infrastructure-as-a-Service projects including OpenStack [155], Nimbus [70], and Eucalyptus [156].

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 [157] from Nvidia as well as the OpenCL effort [158]. When combining GPUs in an otherwise typical HPC environment or supercomputer, major gains in performance and computational ability have been reported in numerous fields [159, 160], ranging from Astrophysics to Bioinformatics. Furthermore, these gains in computational power have also reportedly come at an increased performance-per-watt [161], a metric that is increasingly important to the HPC community as we move closer to exascale computing [162] 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 [163]. 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

1569 given methodology using two different Nvidia Tesla GPUs and compare to the best-
1570 case native application in Section 5. Then, we discuss the implications of these results
1571 in Section 6 and consider the applicability of each method within a production cloud
1572 system. Finally, we conclude with our findings and suggest directions for future work.

1573 4.3 Virtual GPU Directions

1574 Recently, GPU programming has been a primary focus for numerous scientific com-
1575 puting applications. Significant progress has been accomplished in many different
1576 workloads, both in science and engineering, based on parallel abilities of GPUs for
1577 floating point operations and very high on-GPU memory bandwidth. This hardware,
1578 coupled with CUDA and OpenCL programming frameworks, has led to an explosion
1579 of new GPU-specific applications. In some cases, GPUs outperform even the fastest
1580 multicore counterparts by an order of magnitude [164]. In addition, further research
1581 could leverage the per-node performance of GPU accelerators with the high speed,
1582 low latency interconnects commonly utilized in supercomputers and clusters to create
1583 a hybrid GPU + MPI class of applications. The number of distributed GPU appli-
1584 cations is increasing substantially in supercomputing, usually scaling many GPUs
1585 simultaneously [165].

1586 Since the establishment of cloud computing in industry, research groups have
1587 been evaluating its applicability to science [3]. Historically, HPC and Grids have
1588 been on similar but distinct paths within distributed systems, and have concentrated
1589 on performance, scalability, and solving complex, tightly coupled problems within
1590 science. This has led to the development of supercomputers with many thousands
1591 of cores, high speed, low latency interconnects, and sometimes also coprocessors and
1592 FPGAs [166, 167]. Only recently have these systems been evaluated from a cloud

1593 perspective [43]. An overarching goal exists to provide HPC Infrastructure as its own
1594 service (HPCaaS) [168], aiming to classify and limit the overhead of virtualization, and
1595 reducing the bottlenecks classically found in CPU, memory, and I/O operations within
1596 hypervisors [44, 169]. Furthermore, the transition from HPC to cloud computing
1597 becomes more complicated when we consider adding GPUs to the equation.

1598 GPU availability within a cloud is a new concept that has sparked a large amount
1599 of interest within the community. The first successfully deployment of GPUs within
1600 a cloud environment was the Amazon EC2 GPU offering. A collaboration between
1601 Nvidia and Citrix also exists to provide cloud-based gaming solutions to users using
1602 the new Kepler GPU architecture [170]. However, this is currently not targeted
1603 towards HPC applications.

1604 The task of providing a GPU accelerator for use in a virtualized cloud environment
1605 is one that presents a myriad of challenges. This is due to the complicated nature of
1606 virtualizing drivers, libraries, and the heterogeneous offerings of GPUs from multiple
1607 vendors. Currently, two possible techniques exist to fill the gap in providing GPUs
1608 in a cloud infrastructure: back-end I/O virtualization, which this chapter focuses on,
1609 and Front-end remote API invocation.

1610 4.3.1 Front-end Remote API invocation

1611 One method for using GPUs within a virtualized cloud environment is through front-
1612 end library abstractions, the most common of which is remote API invocation. Also
1613 known as API remoting or API interception, it represents a technique where API
1614 calls are intercepted and forwarded to a remote host where the actual computation
1615 occurs. The results are then returned to the front-end process that spawned the
1616 invocation, potentially within a virtual machine. The goal of this method is to provide
1617 an emulated device library where the actual computation is offloaded to another

1618 resource on a local network.

1619 Front-end remote APIs for GPUs have been implemented by a number of differ-
1620 ent technologies for different uses. To solve the problem of graphics processing in
1621 VMs, VMWare [171] has developed a device-emulation approach that emulates the
1622 Direct3D and OpenGL calls to leverage the host OS graphics processing capabili-
1623 ties to provide a 3D environment within a VM. API interception through the use of
1624 wrapper binaries has also been implemented by technologies such as Chromium [172],
1625 and Blink. However these graphics processing front-end solutions are not suitable for
1626 general purpose scientific computing, as they do not expose interfaces that CUDA or
1627 OpenCL can use.

1628 Currently, efforts are being made to provide a front-end remote API invocation
1629 solutions for the CUDA programming architecture. vCUDA [52] was the first of such
1630 technologies to enable transparent access of GPUs within VMs by API call inter-
1631 ception and redirection of the CUDA API. vCUDA substitutes the CUDA runtime
1632 library and supports a transmission mode using XMLRPC, as well as a sharing mode
1633 that is built on VMRPC, a dedicated remote procedure call architecture for VMM
1634 platforms. This share model can leads to better performance, especially as the volume
1635 of data increases, although there may be limitations in VMM interoperability.

1636 Like vCUDA, gVirtuS uses API interception to enable transparent CUDA, OpenCL,
1637 and OpenGL support for Xen, KVM, and VMWare virtual machines [173]. gVirtuS
1638 uses a front-end/back-end model to provide a VMM-independent abstraction layer
1639 to GPUs. Data transport from gVirtuS' front-end to the back-end is accomplished
1640 through a combination of shared memory, sockets, or other hypervisor-specific APIs.
1641 gVirtuS' primary disadvantage is in its decreased performance in host-to-device and
1642 device-to-host data movement due to overhead of data copies to and from its shared
1643 memory buffers. Recent work has also enabled the dynamic sharing of GPUs by

1644 leveraging the gVirtus back-end system with relatively good results [174], however
1645 process-level GPU resource sharing is outside the scope of this manuscript.

1646 rCUDA [51], a recent popular remote CUDA framework, also provides remote API
1647 invocation to enable VMs to access remote GPU hardware by using a sockets based
1648 implementation for high-speed near-native performance of CUDA based applications.
1649 rCUDA recently added support for using InfiniBand's high speed, low latency network
1650 to increase performance for CUDA applications with large data volume requirements.
1651 rCUDA version 4.1 also supports the CUDA runtime API version 5.0, which supports
1652 peer device memory access and unified addressing. One drawback of this method
1653 is that rCUDA cannot implement the undocumented and hidden functions within
1654 the runtime framework, and therefore does not support all CUDA C extensions.
1655 While rCUDA provides some support tools, native execution of CUDA programs
1656 is not possible and programs need to be recompiled or rewritten to use rCUDA.
1657 Furthermore, like gVirtuS and many other solutions, performance between host-to-
1658 device data movement is only as fast as the underlying interconnect, and in the best
1659 case with native RDMA InfiniBand, is roughly half as fast as native PCI Express
1660 usage when using the standard QDR InfiniBand.

1661 4.3.2 Back-end PCI passthrough

1662 Another approach to using a GPU in a virtualized environment is to provide a VM
1663 with direct access to the GPU itself, instead of relying on a remote API. This chapter
1664 focuses on such an approach. Devices on a host's PCI-express bus are virtualized
1665 using directed I/O virtualization technologies recently implemented by chip manu-
1666 facturers, and then direct access is relinquished upon request to a guest VM. This
1667 can be accomplished using the VT-d and IOMMU instruction sets from Intel and
1668 AMD, respectively. This mechanism, typically called PCI passthrough, uses a mem-

1669 ory management unit (MMU) to handle direct memory access (DMA) coordination
1670 and interrupt remapping directly to the guest VM, thus bypassing the host entirely.
1671 With host involvement being nearly non-existent, near-native performance of the PCI
1672 device within the guest VM can be achieved, which is an important characteristic for
1673 using a GPU within a cloud infrastructure.

1674 PCI passthrough itself has recently become a standard technique for many other
1675 I/O systems such as storage or network controllers. However, GPUs (even from
1676 the same vendor) have additional legacy VGA compatibility issues and non-standard
1677 low-level interface DMA interactions that make direct PCI passthrough nontrivial.
1678 VMWare has started use of a vDGA system for hardware GPU utilization, however it
1679 remains in tech preview and only documentation for Windows VMs is present [171]. In
1680 our experimentation, we have found that the Xen hypervisor provides a good platform
1681 for performing PCI passthrough of GPU devices to VMs due to its open nature,
1682 extensive support, and high degree of reconfigurability. Work with Xen in [175] gives
1683 hints at good performance for PCI passthrough in Xen, however further evaluation
1684 with independent benchmarks is needed when looking at scientific computing with
1685 GPUs.

1686 Today's GPUs can provide a variety of frameworks for application programmers to
1687 use. Two common solutions are CUDA and OpenCL. CUDA, or the Compute Unified
1688 Device Architecture, is a framework for creating and running parallel applications on
1689 Nvidia GPUs. OpenCL provides a more generic and open framework for parallel
1690 computation on CPUs and GPUs, and is available for a number of Nvidia, AMD, and
1691 Intel GPUs and CPUs. While OpenCL provides a more robust and portable solution,
1692 many HPC applications utilize the CUDA framework. As such, we focus only on
1693 Nvidia based CUDA-capable GPUs as they offer the best support for a wide array of
1694 programs, although this work is not strictly limited to Nvidia GPUs.

1695 **4.4 Implementation**

1696 In this chapter we use a specific host environment to enable PCI passthrough. First,
1697 we start with the Xen 4.2.2 hypervisor on Centos 6.4 and a custom 3.4.50-8 Linux
1698 kernel with Dom0 Xen support. Within the Xen hypervisor, GPU devices are seized
1699 upon boot and assigned to the xen-pciback kernel module. This process blocks the
1700 host devices from loading the Nvidia or nouveau drivers, keeping the GPUs uninitialized
1701 and therefore able to be assigned to DomU VMs.

1702 Xen, like other hypervisors, provides a standard method of passing through PCI
1703 devices to guest VMs upon creation. When assigning a GPU to a new VM, Xen loads
1704 a specific VGA BIOS to properly initialize the device enabling DMA and interrupts
1705 to be assigned to the guest VM. Xen also relinquishes control of the GPU via the
1706 xen-pciback module. From there, the Linux Nvidia drivers are loaded and the device
1707 is able to be used as expected within the guest. Upon VM termination, the xen-
1708 pciback module re-seizes the GPU and the devices can be re-assigned to new VMs in
1709 the future.

1710 This mechanism of PCI passthrough for GPUs can be implemented using multiple
1711 devices per host, as illustrated in Figure 6.1. Here, we see how the device's connection
1712 to the VM totally bypasses the Dom0 host as well as the Xen VMM, and is managed
1713 by VT-d or IOMMU to access the PCI-Express bus which the GPUs utilize. This is in
1714 contrast to other common virtual device uses, where hardware is emulated by the host
1715 and shared across all guests. This is the common usage for Ethernet controllers and
1716 input devices to enable users to interact with VMs as they would with native hosts,
1717 unlike the bridged model shown in the figure. The potential downside of this method
1718 is there can be a 1:1 or 1:N mapping of VMs to GPUs only. A M:1 mapping where
1719 multiple VMs use a GPU is not possible. However, almost all scientific applications

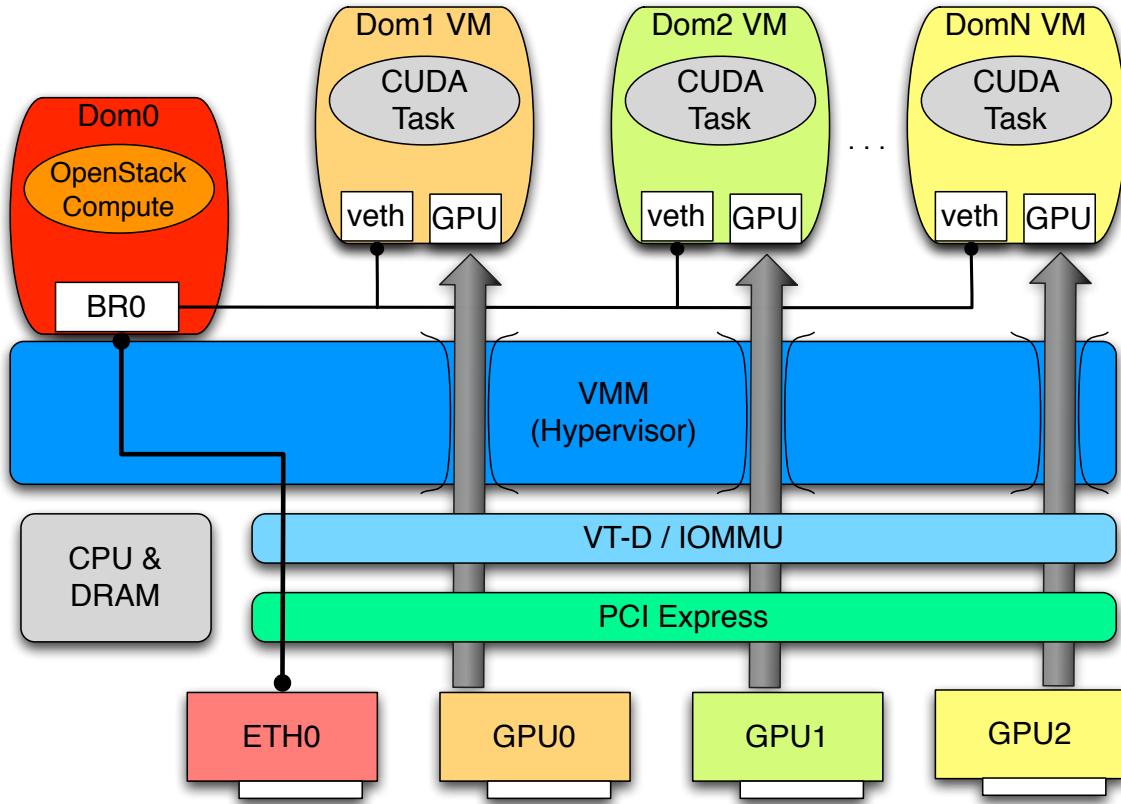


Figure 4.1 GPU PCI passthrough within the Xen Hypervisor

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,

1730 and even enables researchers to potentially use other tools within a VM that might
1731 not be supported within a supercomputer or cluster. Unlike with remote API im-
1732 plementations, users don't need to recompile or modify their code, as the GPUs are
1733 essentially local to the data. Further comparing to remote API implementations,
1734 using PCI passthrough within a VM allows users to leverage any GPU framework
1735 available, OpenCL or CUDA, and any higher level programming frameworks such as
1736 within Matlab or Python.

1737 Through the use of advanced scheduling techniques within cloud infrastructure, we
1738 can also take advantage of PCI passthrough implementation for efficiency purposes.
1739 Users could request VMs with GPUs which get scheduled for creation on machines
1740 that provide such resources, but can also request normal VMs as well. The scheduler
1741 can correctly map VM requirement requests to heterogeneous hardware. This enables
1742 large scale resources to support a wide array of scientific computing applications
1743 without the added cost of putting GPUs in all compute nodes.

1744 4.5 Experimental Setup

1745 In this chapter back-end GPU PCI passthrough to virtual machines using the Xen
1746 hypervisor is detailed, however proper evaluation of the performance of such method
1747 needs to be properly considered. As such, we ran an array of benchmarks that
1748 evaluate the performance of this method compared to the same hardware running
1749 native bare-metal GPU code without any virtualization. We focus our tests on single-
1750 node performance to best understand low level overhead.

1751 To evaluate the effectiveness of GPU-enabled VMs within Xen, two different ma-
1752 chines were used to represent two generations of Nvidia GPUs. The first system at
1753 Indiana University consists of dual-socket Intel Xeon X5660 6-core CPUs at 2.8Ghz

1754 with 192GB DDR3 RAM, 8TB RAID5 array, and two Nvidia Tesla "Fermi" C2075
1755 GPUs. The system at USC/ISI uses a dual-socket Intel Xeon E5-2670 8-core CPUs
1756 at 2.6Ghz with 48GB DDR3 RAM, 3 600GB SAS disk drives, but with the latest
1757 Nvidia Tesla "Kepler" K20m GPU supplied by Nvidia. These machines represent
1758 the present Fermi series GPUs along with the recently release Kepler series GPUs,
1759 providing a well-rounded experimental environment. Native systems were installed
1760 with standard Centos 6.4 with a stock 2.6.32-279 Linux kernel. Xen host systems
1761 were still loaded with Centos 6.4 but with a custom 3.4.53-8 Linux kernel and Xen
1762 4.2.22. Guest VMs were also Centos 6.4 with N-1 processors and 24GB of memory
1763 and 1 GPU passed through in HVM full virtualization mode. Using both IU and
1764 USC/ISI machine configurations in native and VM modes represent the 4 test cases
1765 for our work.

1766 In order to evaluate the performance, the SHOC Benchmark suite [176] was used
1767 to extensively evaluate performance across each test platform. The SHOC bench-
1768 marks were chosen because they provide a higher level of evaluation regarding GPU
1769 performance than the sample applications provided in the Nvidia SDK, and can also
1770 evaluate OpenCL performance in similar detail. The benchmarks were compiled us-
1771 ing the NVCC compiler in CUDA5 driver and library, along with OpenMPI 1.4.2 and
1772 GCC 4.4. Each benchmark was run a total of 20 times, with the results given as an
1773 average of all runs.

1774 4.6 Results

1775 Results of all benchmarks are compressed into three subsections: floating point oper-
1776 ations, device bandwidth and pci bus performance. Each represents a different level
1777 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.

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%).

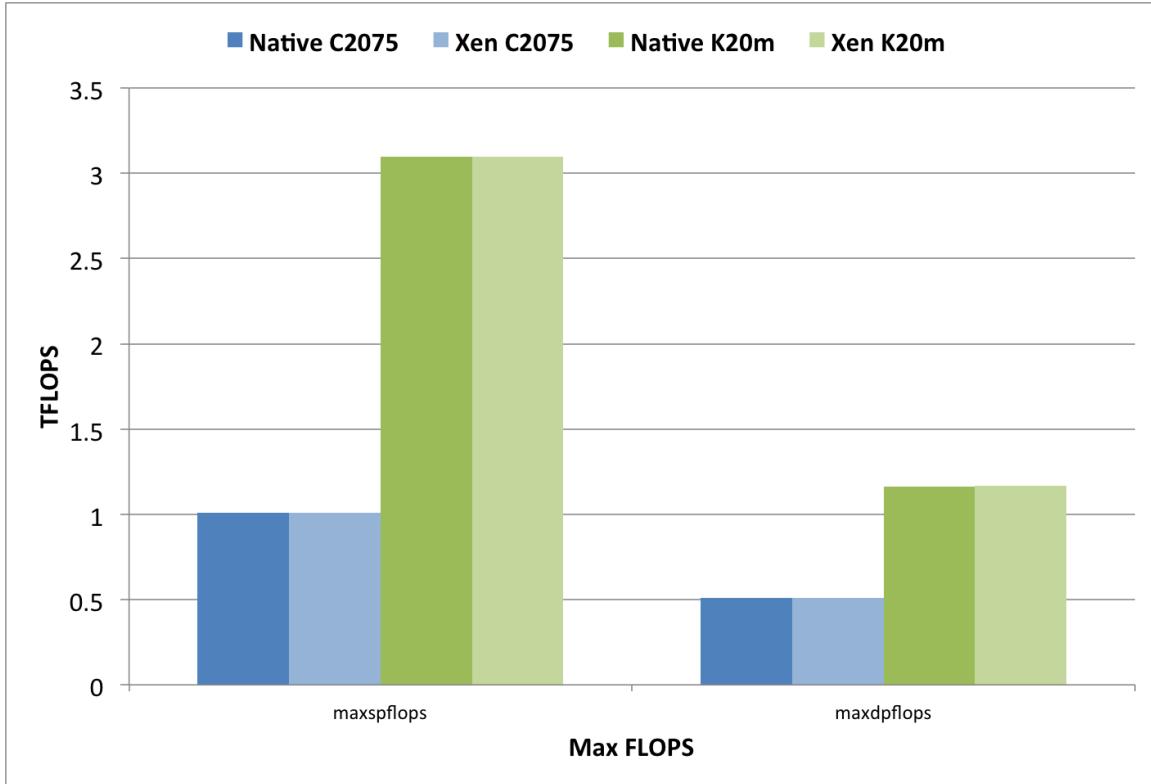


Figure 4.2 GPU Floating Point Operations per Second

1804 Other FLOP-based benchmarks are used to emulate higher level applications.
 1805 Stencil represents a 9-point stencil operation applied to a 2D data set, and the S3D
 1806 benchmark is a computationally-intensive kernel from the S3D turbulent combustion
 1807 simulation program [177]. In Figure 4.5, we see that both the Fermi C2075 and Kepler
 1808 K20m GPUs performing well compared to the native base case, showing the overhead
 1809 of virtualization is low. The C2075-enabled VMs experience slightly more overhead
 1810 when compared to native performance again for pcie runs, but overhead is at most
 1811 7% for the S3D benchmark.

1812 4.6.2 Device Speed

1813 While floating point operations allow for the proposed solution to relate to many tradi-
 1814 tional HPC applications, they are just one facet of GPU performance within scientific

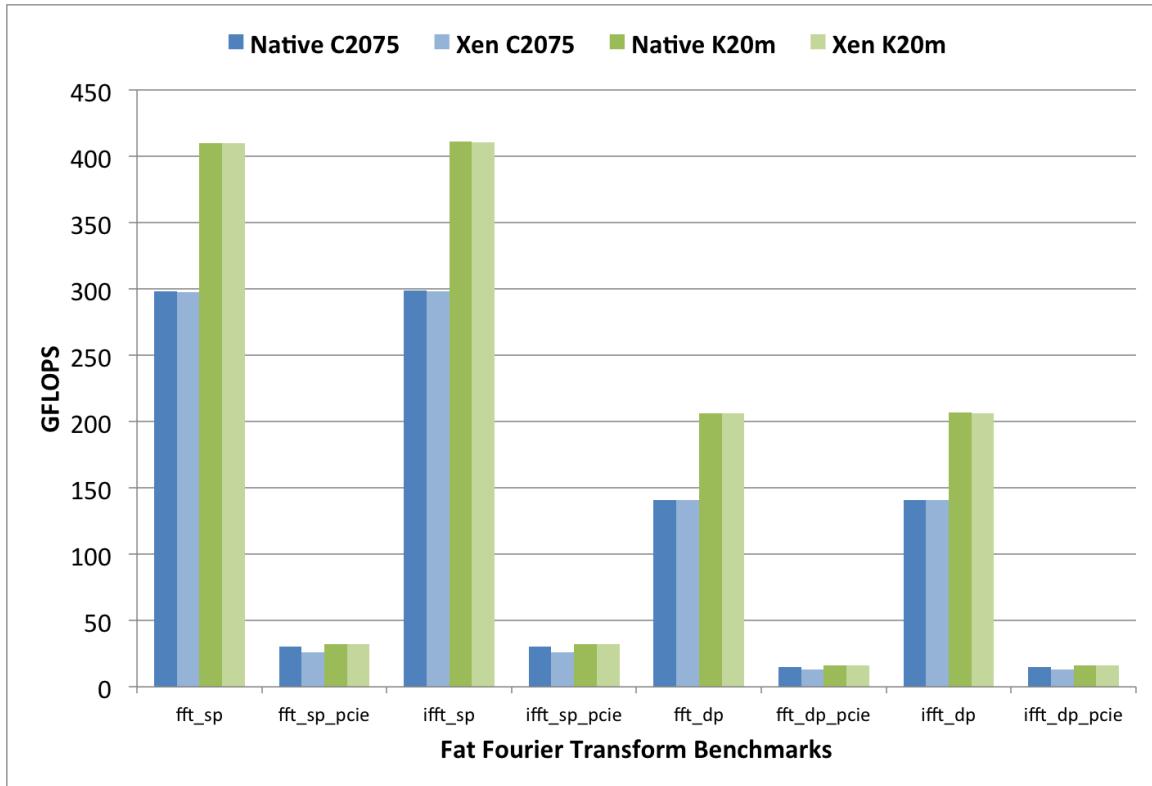


Figure 4.3 GPU Fast Fourier Transform

1815 computing. Device speed, measured in both raw bandwidth and additional bench-
 1816 marks, provides a different perspective towards evaluating GPU PCI passthrough in
 1817 Xen. Figure 4.6 illustrates device level memory access of various GPU device mem-
 1818 ory structures. With both Nvidia GPUs, virtualization has little to no impact on the
 1819 performance of inter-device memory bandwidth. As expected the Kepler K20m out-
 1820 performed the C2075 VMs and there was a higher variance between runs with both
 1821 native and VM cases. Molecular Dynamics and Reduction benchmarks in Figure 4.7
 1822 perform again at near-native performance without the pcie bus taken into account.
 1823 However the overhead observed increases to 10-15% when the PCI-Express bus is
 1824 considered when looking at the Fermi C2075 VMs.

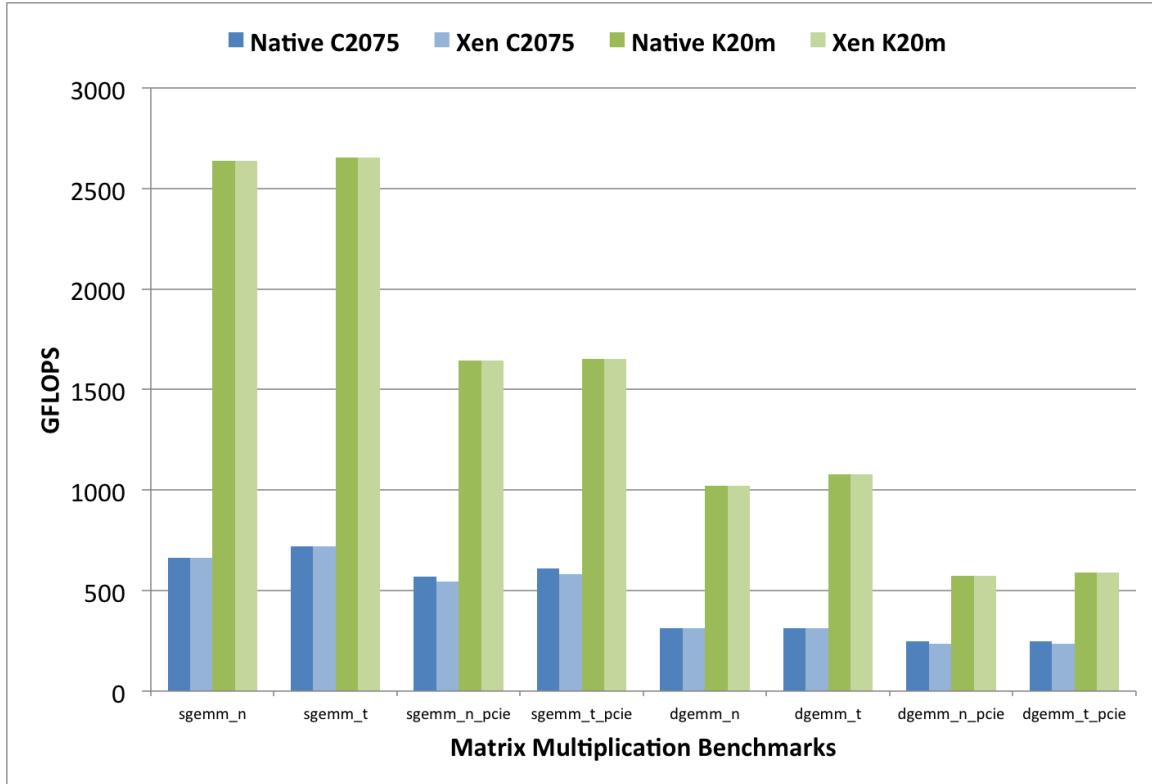


Figure 4.4 GPU Matrix Multiplication

1825 4.6.3 PCI Express Bus

1826 Upon evaluating PCI passthrough of GPUs, it is apparent that the PCI express bus
 1827 is subject to the greatest potential for overhead, as was observed in the Fermi C2075
 1828 benchmarks. The VT-d and IOMMU chip instruction sets interface directly with the
 1829 PCI bus to provide operational and security related mechanisms for each PCI device,
 1830 thereby ensuring proper function in a multi-guest environment but potentially intro-
 1831 ducing some overhead. As such, it is imperative to investigate any and all overhead
 1832 at the PCI Express bus.

1833 Figure 4.8 looks at maximum PCI bus speeds for each experimental implemen-
 1834 tation. First, we see a consistent overhead in the Fermi C2075 VMs, with a 14.6%
 1835 performance impact for download (to-device) and a 26.7% impact in readback (from-

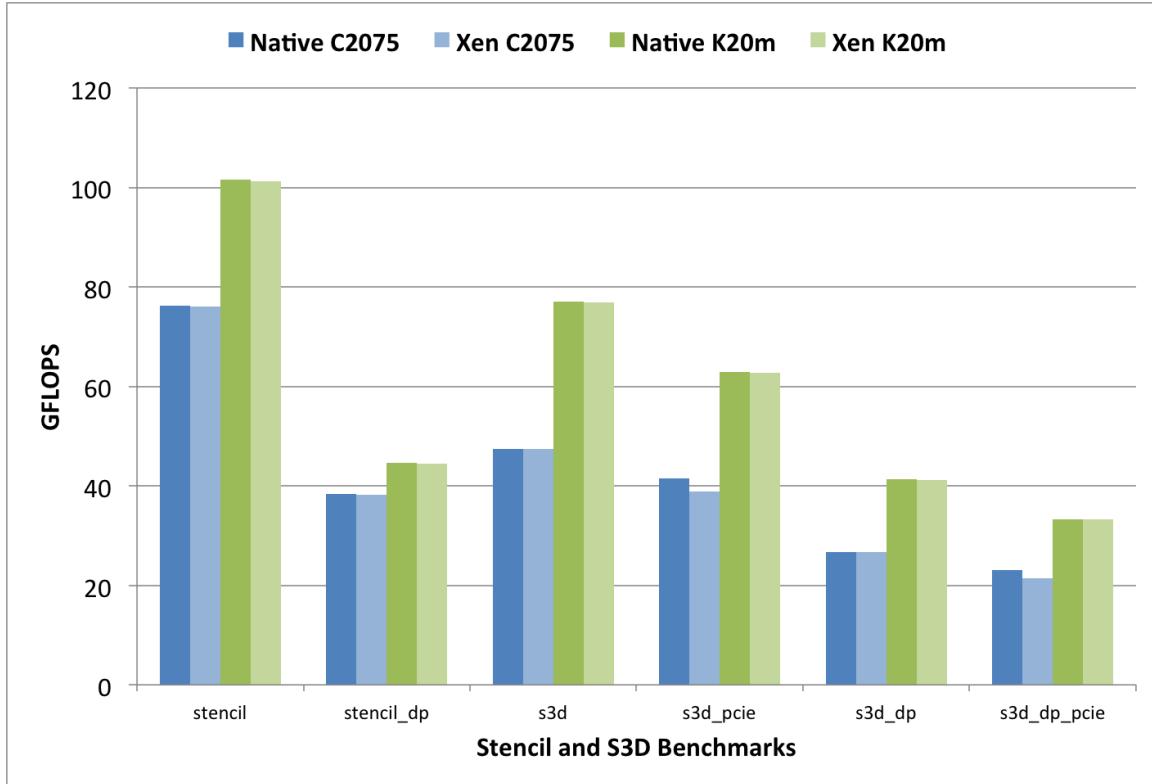


Figure 4.5 GPU Stencil and S3D

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 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 a 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 ar-

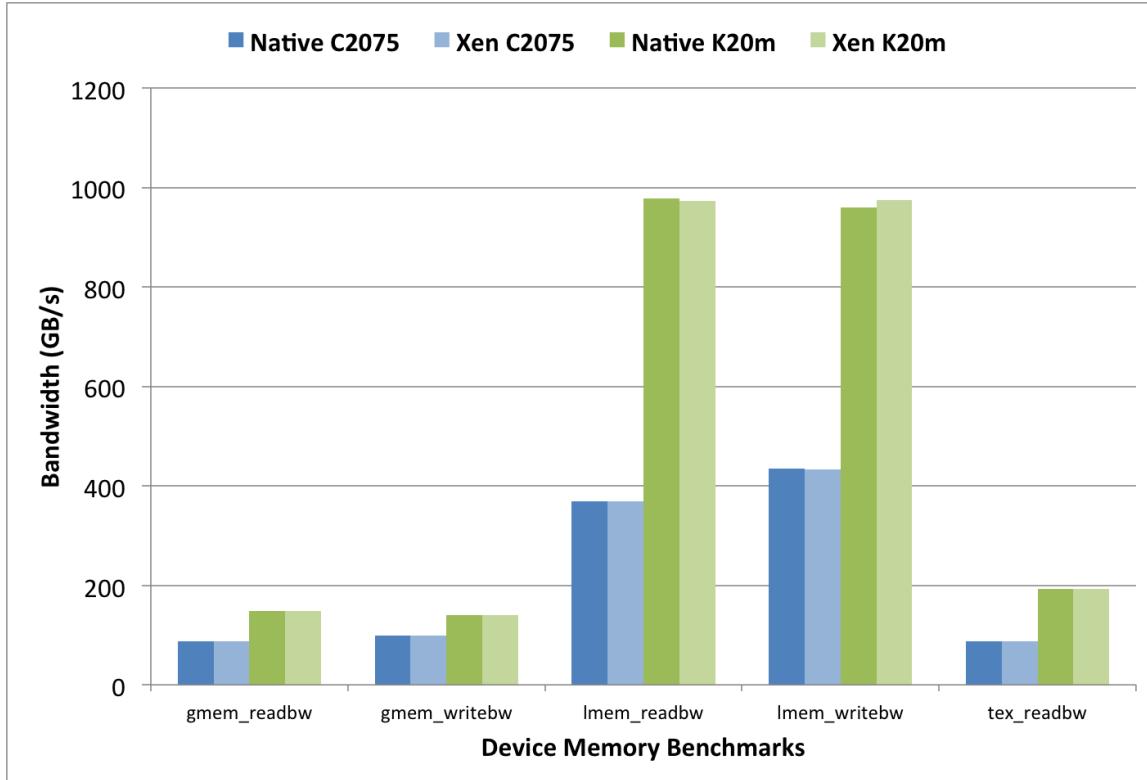


Figure 4.6 GPU Device Memory Bandwidth

1848 chitecture and the Kepler K20m experimental setup. In summary, GPU PCI-Express
 1849 performance in a VM is sensitive to the host CPU's NUMA architecture and care is
 1850 needed to mitigate the impact, either by leveraging new architectures or by proper us-
 1851 age of Xen's VM core assignment features. Furthermore, the overhead in this system
 1852 diminishes significantly when using the new Kepler GPUs by Nvidia.

1853 4.7 Discussion

1854 This chapter evaluates the use of general purpose GPUs within cloud computing in-
 1855 frastructure, primarily targeted towards advanced scientific computing. The method
 1856 of PCI passthrough of GPUs directly to a guest virtual machine running on a tuned
 1857 Xen hypervisor shows initial promise for an ubiquitous solution in cloud infrastruc-

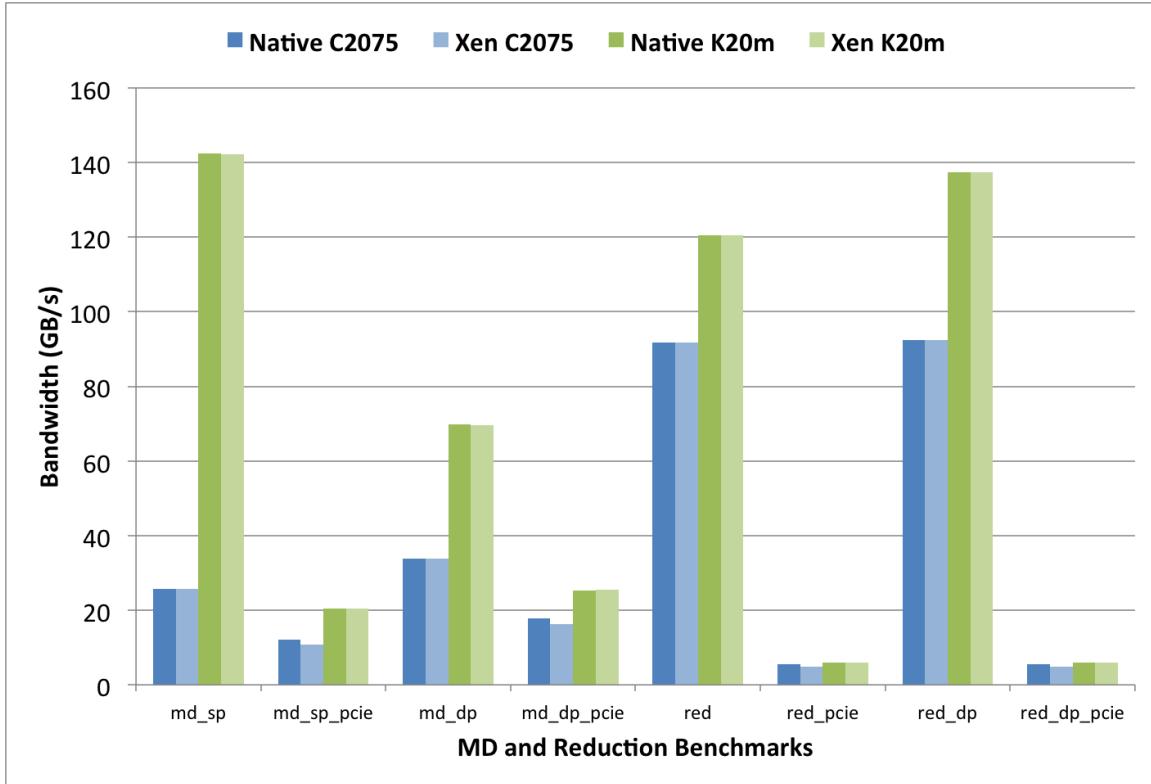


Figure 4.7 GPU Molecular Dynamics and Reduction

ture. 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-

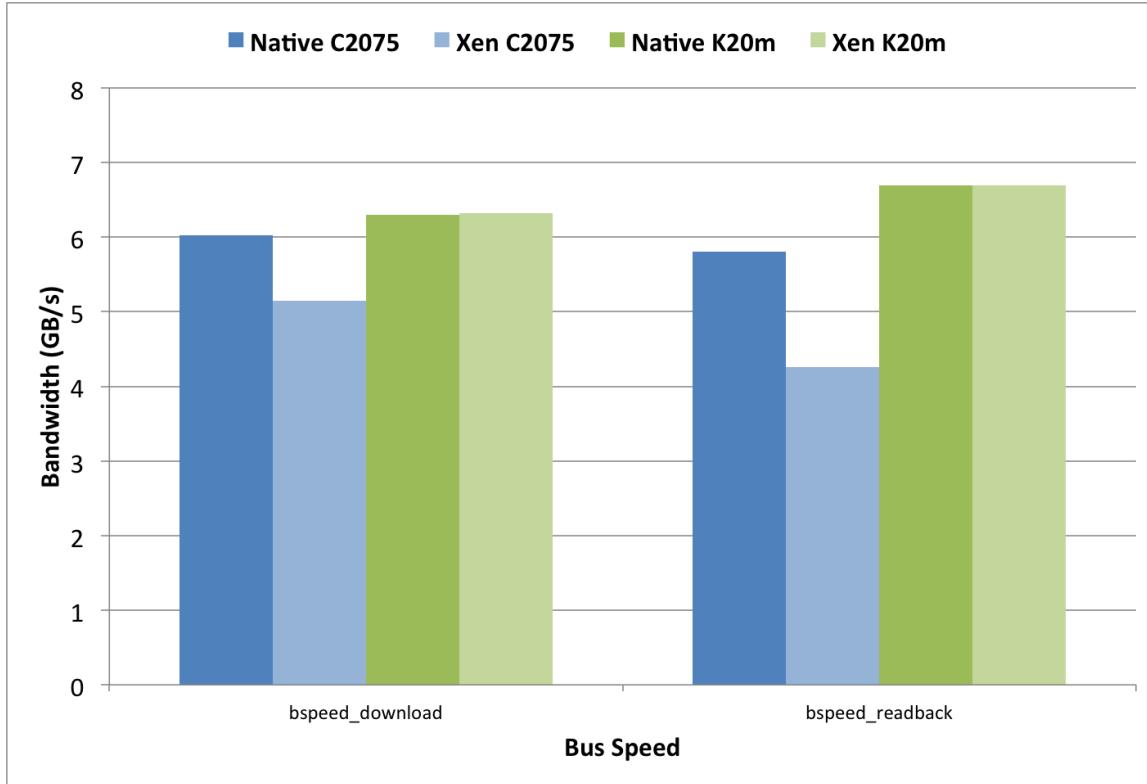


Figure 4.8 GPU PCI Express Bus Speed

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 [178], 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 infras-

1882 tructure. Already this has been confirmed in [179], where similar a methodology has
1883 been leveraged specifically for Bioinformatics applications in the cloud.

1884 **4.8 Chapter Summary and Future Work**

1885 The ability to use GPUs within virtual machines represents a leap forward for sup-
1886 porting advanced scientific computing within cloud infrastructure. The method of
1887 direct PCI passthrough of Nvidia GPUs using the Xen hypervisor offers a clean, re-
1888 producible solution that can be implemented within many Infrastructure-as-a-Service
1889 (IaaS) deployments. Performance measurements indicate that the overhead of pro-
1890 viding a GPU within Xen is minimal compared to the best-case native use, however
1891 NUMA inconsistencies can impact performance. The New Kepler-based GPUs oper-
1892 ate with a much lower overhead, making those GPUs an ideal choice when designing
1893 a new GPU IaaS system.

1894 Next steps for this work could involve providing GPU-based PCI passthrough
1895 within the OpenStack nova IaaS framework. This will enable research laboratories
1896 and institutions to create new private or national-scale cloud infrastructure that have
1897 the ability to support new scientific computing challenges. Other hypervisors could
1898 also leverage GPU PCI passthrough techniques and warrant in-depth evaluation in
1899 the future. Furthermore, we hope to integrate this work with advanced interconnects
1900 and other heterogeneous hardware and provide a parallel high performance cloud
1901 infrastructure to enable mid-tier scientific computing.

₁₉₀₂ **Chapter 5**

₁₉₀₃ **GPU-Passthrough Performance: A
1904 Comparison of KVM, Xen,
1905 VMWare ESXi, and LXC for
1906 CUDA and OpenCL Applications**

₁₉₀₇ **5.1 Introduction**

₁₉₀₈ As scientific workloads continue to demand increasing performance at greater power
₁₉₀₉ efficiency, high performance architectures have been driven towards heterogeneity and
₁₉₁₀ specialization. Intel's Xeon Phi, and GPUs from both NVIDIA and AMD represent
₁₉₁₁ some of the most common accelerators, with each capable of delivering improved
₁₉₁₂ performance and power efficiency over commodity multi-core CPUs.

₁₉₁₃ Infrastructure-as-a-Service (IaaS) clouds have the potential to democratize access
₁₉₁₄ to the latest, fastest, and most powerful computational accelerators. This is true
₁₉₁₅ of both public and private clouds. Yet today's clouds are typically homogeneous

1916 without access to even the most commonly used accelerators. Historically, enabling
1917 virtual machine access to GPUs and other PCIe devices has proven complex and
1918 error-prone, with only a small subset of GPUs being certified for use within a few
1919 commercial hypervisors. This is especially true for NVIDIA GPUs, likely the most
1920 popular for scientific computing, but whose drivers have always been closed source.

1921 Given the complexity surrounding the choice of GPUs, host systems, and hypervi-
1922 sors, it is perhaps no surprise that Amazon is the only major cloud provider offering
1923 customers access to GPU-enabled instances. All of this is starting to change, however,
1924 as open source and other freely available hypervisors now provide sufficiently robust
1925 PCI passthrough functionality to enable GPU and other accelerator access whether
1926 in the public or private cloud.

1927 Today, it is possible to access GPUs at high performance within all of the major
1928 hypervisors, merging many of the advantages of cloud computing (e.g. custom images,
1929 software defined networking, etc.) with the accessibility of on-demand accelerator
1930 hardware. Yet, no study to date has systematically compared the performance of PCI
1931 passthrough across all major cloud hypervisors. Instead, alternative solutions have
1932 been proposed that attempt to virtualize the GPU [180] , but sacrifice performance.

1933 In this chapter, we characterize the performance of both NVIDIA Fermi and
1934 Kepler GPUs operating in PCI passthrough mode in VMWare VSphere, Linux KVM,
1935 Xen, and Linux Containers (LXC). Through a series of microbenchmarks as well as
1936 scientific and Big Data applications, we make two contributions:

- 1937 1. We demonstrate that PCI passthrough at high performance is possible for GPUs
1938 across 4 major hypervisors.
- 1939 2. We describe the lessons learned through our performance analysis, as well as
1940 the relative advantages and disadvantages of each hypervisor for GPU support.

1941 5.2 Related Work & Background

1942 GPU virtualization and GPU-passthrough are used within a variety of contexts, from
1943 high performance computing to virtual desktop infrastructure. Accessing one or more
1944 GPUs within a virtual machine is typically accomplished by one of two strategies: 1)
1945 via API remoting with device emulation; or 2) using PCI passthrough.

1946 5.2.1 GPU API Remoting

1947 rCUDA, vCUDA, GViM, and gVirtuS are well-known API remoting solutionsFun-
1948 damentally, these approaches operate similarly by splitting the driver into a front-
1949 end/back-end model, where calls into the interposed CUDA library (front-end) are
1950 sent via shared memory or a network interface to the back-end service that executes
1951 the CUDA call on behalf of the virtual machine. Notably, this technique is not limited
1952 to CUDA, but can be used to decouple OpenCL, OpenGL, and other APIs from their
1953 local GPU or accelerator.

1954 The performance of API-remoting depends largely on the application and the
1955 remoting solution's implementation. Bandwidth and latency-sensitive benchmarks and
1956 applications will tend to expose performance bottlenecks more than compute-intensive
1957 applications. Moreover, solutions that rely on high speed networks, such as Infini-
1958 band, will compete with application-level networking for bandwidth.

1959 5.2.2 PCI Passthrough

1960 Input/Output Memory Management Units, or IOMMUs, play a fundamental roll in
1961 the PCI-passthrough virtualization mechanism. Like traditional MMUs that provide
1962 a virtual memory address space to CPUs [181], an IOMMU serves the fundamental
1963 purpose of connecting a direct memory access (DMA) capable I/O bus to main mem-

1964 ory. The IOMMU unit, typically within the chipset, maps device virtual addresses to
1965 physical memory addresses. This process also has the added improvement of guaran-
1966 teeing device isolation by blocking rogue DMA and interrupt requests [182], with a
1967 slight overhead, especially in early implementations [183].

1968 Currently two major IOMMU implementations exist, VT-d and AMD-Vi by In-
1969 tel and AMD, respectively. Both specifications provide DMA remapping to enable
1970 PCI-passthrough as well as other features such as interrupt remapping, hypervisor
1971 snooping, and security control mechanisms to ensure proper and efficient hardware
1972 utilization. PCI passthrough has been studied within the context of networking [184],
1973 storage [185], and other PCI-attached devices; however, GPUs have historically lagged
1974 behind other devices in their support for virtual machine passthrough.

1975 **5.2.3 GPU Passthrough, a Special Case of PCI Passthrough**

1976 While generic PCI passthrough can be used with IOMMU technologies to pass through
1977 many PCI-Express devices, GPUs represent a special case of PCI devices, and a spe-
1978 cial case of PCI passthrough. In traditional usage, GPUs usually serve as VGA
1979 devices primarily to render screen output, and while the GPUs used in this study do
1980 not render screen out, the function still exists in legacy. In GPU-passthrough, another
1981 VGA device (such as onboard graphics built into the motherboard, or a baseboard
1982 management controller) is necessary to serve as the primary display for the host, as
1983 well as providing emulated VGA devices for each guest VM. Most GPUs also have a
1984 video BIOS that requires full initialization and reset functions, which is often difficult
1985 due to the proprietary nature of the cards and their drivers.

1986 Nevertheless, for applications that require native or near-native GPU performance
1987 across the full spectrum of applications with immediate access to the latest GPU
1988 drivers and compilers, GPU passthrough solutions are preferable to API remoting.

1989 Today, Citrix Xenserver, open source Xen [186], and VMWare ESXi [187], and most
1990 recently KVM all support GPU passthrough. To our knowledge, no one has system-
1991 atically characterized the performance of GPU passthrough across a range of hyper-
1992 visors, across such a breadth of benchmarks, and across multiple GPU generations as
1993 we do.

1994 **5.3 Experimental Methodology**

1995 **5.3.1 Host and Hypervisor Configuration**

1996 We used two hardware systems, named Bespin and Delta, to evaluate four hypervisors.
1997 The Bespin system at USC/ISI represents Intel’s Sandy Bridge microarchitecture with
1998 a Kepler class K20 GPU. The Delta system, provided by the FutureGrid project [188],
1999 represents the Westmere microarchitecture with a Fermi class C2075 GPU. Table 5.1
2000 provides the major hardware characteristics of both systems. Note that in addition,
2001 both systems include 10 gigabit Ethernet, gigabit Ethernet, and either FDR or QDR
2002 Infiniband. Our experiments do not emphasize networking, and we use the gigabit
2003 ethernet network for management only.

2004 A major design goal of these experiments was to reduce or eliminate NUMA effects
2005 (non-uniform memory access) on the PCI passthrough results in order to facilitate
2006 fair comparisons across hypervisors and to reduce experimental noise. To this end,
2007 we configured our virtual machines and containers to execute only on the NUMA
2008 node containing the GPU under test. We acknowledge that the NUMA effects on
2009 virtualization may be interesting in their own right, but they are not the subject of
2010 this set of experiments.

2011 We use Bespin and Delta to evaluate three hypervisors and one container-based
2012 approach to GPU passthrough. The hypervisors and container system, VMWare

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

2013 ESXi, Xen, KVM, and LXC, are summarized in Table 5.2. Note that each virtual-
 2014 ization solution imposes its own unique requirements on the base operating system.
 2015 Hence, Xen’s Linux kernel refers to the Domain-0 kernel, whereas KVM’s Linux kernel
 2016 represents the actual running kernel hosting the KVM hypervisor. Linux Containers
 2017 share a single kernel between the host and guests, and VMWare ESXi does not rely
 2018 on a Linux kernel at all.

2019 Similarly, hypervisor requirements prevented us from standardizing on a single
 2020 host operating system. For Xen and KVM, we relied on the Arch Linux 2013.10.01
 2021 distribution because it provides easy access to the mainline Linux kernel. For our
 2022 LXC tests, we use CentOS 6.4 because its shared kernel was identical to the base
 2023 CentOS 6.4 kernel used in our testing. VMWare has a proprietary software stack.
 2024 All of this makes comparison challenging, but as we describe in Section 5.3.2, we are
 2025 running a common virtual machine across all experiments.

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
VMWare ESXi 5.5.0	N/A	N/A
LXC	2.6.32-358.23.2	CentOS 6.4

2026 5.3.2 Guest Configuration

2027 We treat each hypervisor as its own system, and compare virtual machine guests
 2028 to a base CentOS 6.4 system. The base system and the guests are all composed of
 2029 CentOS 6.4 installation with a 2.6.32-358.23.2 stock kernel and CUDA version 5.5.
 2030 Each guest is allocated 20 GB of RAM and a full CPU socket (either 6 or 8 CPU
 2031 cores). Bespin experiments received 8 cores and Delta experiments received 6 cores.
 2032 VMs were restricted to a single NUMA node. On the Bespin system, the K20m GPU
 2033 was attached to NUMA node 0. On the Delta system, the C2075 GPU was attached
 2034 to NUMA node 1. Hence VMs ran on NUMA node 0 for the Bespin experiments,
 2035 and node 1 for the Delta experiments.

2036 5.3.3 Microbenchmarks

2037 Our experiments are composed of a mix of microbenchmarks and application-level
 2038 benchmarks, as well as a combination of CUDA and OpenCL benchmarks. The
 2039 SHOC benchmark suite provides a series of microbenchmarks in both OpenCL and
 2040 CUDA [189]. For this analysis, we focus on the OpenCL benchmarks in order to
 2041 exercise multiple programming models. Benchmarks range from low-level peak Flops

2042 and bandwidth measurements, to kernels and mini-applications.

2043 **5.3.4 Application Benchmarks**

2044 For our application benchmarks, we have chosen the LAMMPS molecular dynam-
2045 ics simulator [190], the GPU-LIBSVM [191], and the LULESH shock hydrodynamics
2046 simulator [192]. These represent a range of computational characteristics, from com-
2047 putational physics to big data analytics, and are representative of GPU-accelerated
2048 applications in common use.

2049 **LAMMPS** The Large-scale Atomic/Molecular Parallel Simulator (LAMMPS), is a
2050 parallel molecular dynamics simulator [190, 193] used for production MD simulation
2051 on both CPUs and GPUs [194]. LAMMPS has two packages for GPU support, the
2052 USER-CUDA and GPU packages. With the USER-CUDA package, each GPU is used
2053 by a single CPU, whereas the GPU package allows multiple CPUs to take advantage
2054 of a single GPU. There are performance trade-offs with both approaches, but we chose
2055 to use the GPU package in order to stress the virtual machine by exercising multiple
2056 CPUs. Consistent with the existing GPU benchmarking approaches, our results are
2057 based on the Rhodopsin protein.

2058 **GPU-LIBSVM** LIBSVM is a popular implementation [195] of the machine learn-
2059 ing classification algorithm support vector machine (SVM). GPU-accelerated LIB-
2060 SVM [191] enhances LIBSVM by providing GPU-implementations of the kernel ma-
2061 trix computation portion of the SVM algorithm for radial basis kernels. For bench-
2062 marking purposes we use the NIPS 2003 feature extraction gisette data set. This data
2063 set has a high dimensional feature space and large number of training instances, and
2064 these qualities are known to be computational intensive to generate SVM models.

2065 The GPU-accelerated SVM implementation shows dramatic improvement over the
2066 CPU-only implementation.

2067 **LULESH** Hydrodynamics is widely used to model continuum properties and inter-
2068 actions in materials when there is an applied force [196]. Hydrodynamics applications
2069 consume approximately one third of the runtime of data center resource throughout
2070 the U.S. DoD (Department of Defense). The Livermore Unstructured Lagrange Ex-
2071 plicit Shock Hydro (LULESH) was developed by Lawrence Livermore National Lab
2072 as one of five challenge problems in the DARPA UHPC program. LULESH is widely
2073 used as a proxy application in the U.S. DOE (Department of Energy) co-design effort
2074 for exascale applications [192].

2075

5.4 Performance Results

2076 We characterize GPGPU performance within virtual machines across two hardware
2077 systems, 4 hypervisors, and 3 application sets. We begin with the SHOC benchmark
2078 suite before describing the GPU-LIBSVM, LAMMPS, and LULESH results. All
2079 benchmarks are run 20 times and averaged. Results are scaled with respect to a base
2080 CentOS 6.4 system for both systems. That is, we compare virtualized Bespin per-
2081 formance to non-virtualized Bespin performance, and virtualized Delta performance
2082 to non-virtualized Delta performance. Values less than 1 indicate that the base sys-
2083 tem outperformed the virtual machine, while values greater than 1 indicate that the
2084 virtual machine outperformed the base system. In cases where we present geometric
2085 means across multiple benchmarks, the means are taken over these scaled values, and
2086 the semantics are the same: less than 1 indicates overhead in the hypervisor, greater
2087 than 1 indicates a performance increase over the base system.

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		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	0.999	1.57	0.997	3.34	1.00	1.77	1.00	1.90	
L1	0.998	1.23	0.998	1.39	1.00	1.47	1.00	0.975	
L2	0.998	0.48	0.995	0.846	0.999	1.90	0.999	1.20	
Bespin PCIe-only									
		KVM		Xen		LXC		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	0.997	0.317	0.999	0.143	0.995	0.981	0.995	1.01	
L1	0.998	0.683	0.997	1.39	1.00	0.928	1.01	0.975	
L2	0.998	0.478	0.996	0.846	1.00	0.247	1.01	0.133	

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		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	1.01	0.031	0.969	12.7	1.00	0.073	1.00	4.95	
L1	1.00	1.45	0.959	24.0	1.00	0.663	0.933	36.6	
L2	1.00	0.101	0.982	4.60	1.00	0.016	0.962	7.01	
Delta PCIe-only									
		KVM		Xen		LXC		VMWare	
		Mean	Max	Mean	Max	Mean	Max	Mean	Max
L0	1.04	0.029	0.889	12.7	1.00	0.01	0.995	4.37	
L1	1.00	1.45	0.914	20.5	0.999	0.380	0.864	36.6	
L2	1.00	0.075	0.918	4.60	1.00	N/A	0.869	7.01	

2088 5.4.1 SHOC Benchmark Performance

2089 SHOC splits its benchmarks into 3 Levels, named Levels 0 through 2. Level 0 repre-
2090 sents device-level characteristics, peak Flop/s, bandwidth, etc. Level 1 characterizes
2091 computational kernels: FFT and matrix multiplication, among others. Finally, Level
2092 2 includes “mini-applications,” in this case an implementation of the S3D, a compu-
2093 tational chemistry application.

2094 Because the SHOC OpenCL benchmarks report more than 70 individual mi-
2095 crobenchmarks, space does not allow us to show each benchmark individually. In-
2096 stead, we start with a broad overview of SHOC’s performance across all benchmarks,
2097 hypervisors, and systems. We then discuss in more detail those benchmarks that
2098 either outperformed or underperformed the Bespin (K20) system by 0.50% or more.
2099 We call these benchmarks outliers. As we will show, those outlier benchmarks iden-
2100 tified on the Bespin system, also tend to exhibit comparable characteristics on the
2101 Delta system as well, but the overhead is typically higher.

2102 In Tables 5.3 and 5.4, we provide geometric means for each SHOC level across each
2103 hypervisor and system. We also include the maximum overhead for each hypervisor
2104 at each level to facilitate comparison across hypervisors and systems. Finally, we
2105 provide a comparable breakdown of only the PCIe SHOC benchmarks, based on the
2106 intuition that PCIe-specific benchmarks will likely result in higher overhead.

2107 At a high level, we immediately notice that in the cases of KVM and LXC, both
2108 perform very near native across both the Bespin and Delta platforms. On average,
2109 these systems are almost indistinguishable from their non-virtualized base systems.
2110 So much so, that experimental noise occasionally boosts performance slightly above
2111 their base systems.

2112 This is in sharp contrast to the Xen and VMWare hypervisors, which perform
2113 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 and VMWare 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%. VMWare exhibits similar characteristics, resulting in a maximum overhead of 1.9% in the case of the Bespin system, and a surprising 36.6% in the case of the Delta system. We provide a more in-depth discussion of these overheads below.

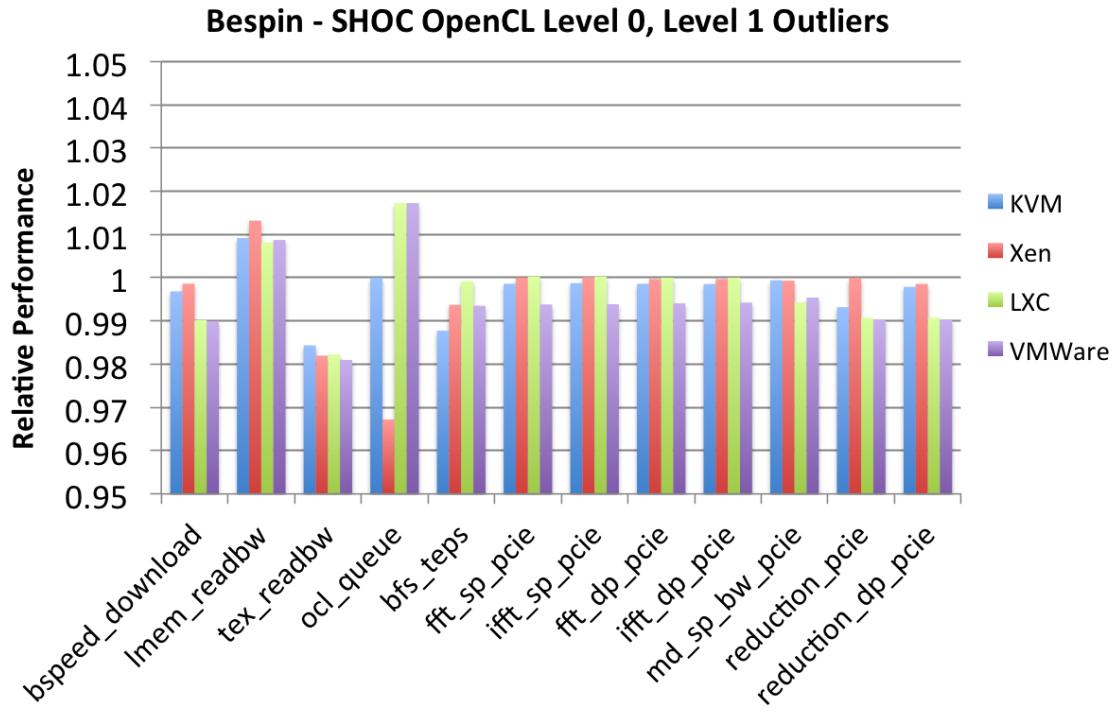


Figure 5.1 SHOC Levels 0 and 1 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

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

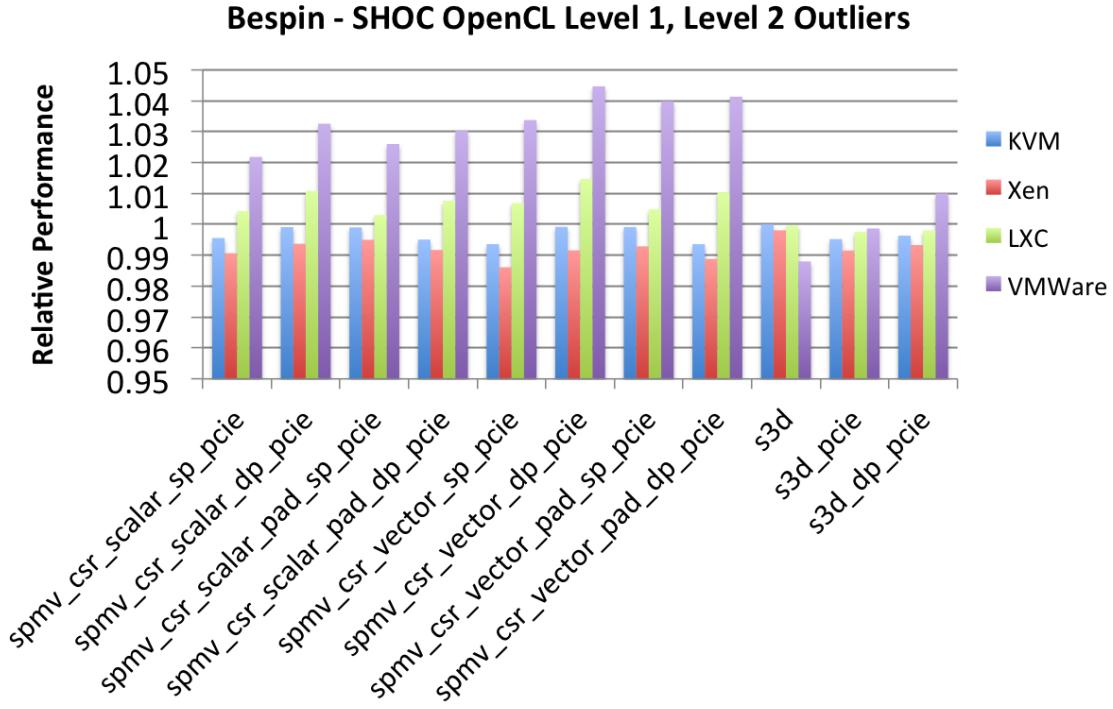


Figure 5.2 SHOC Levels 1 and 2 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

overhead. The remainder perform within one standard deviation of the base, despite an overhead of greater than 0.5%.

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 1% overhead for bspeed_download in VMWare, and less than 0.5% overhead 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

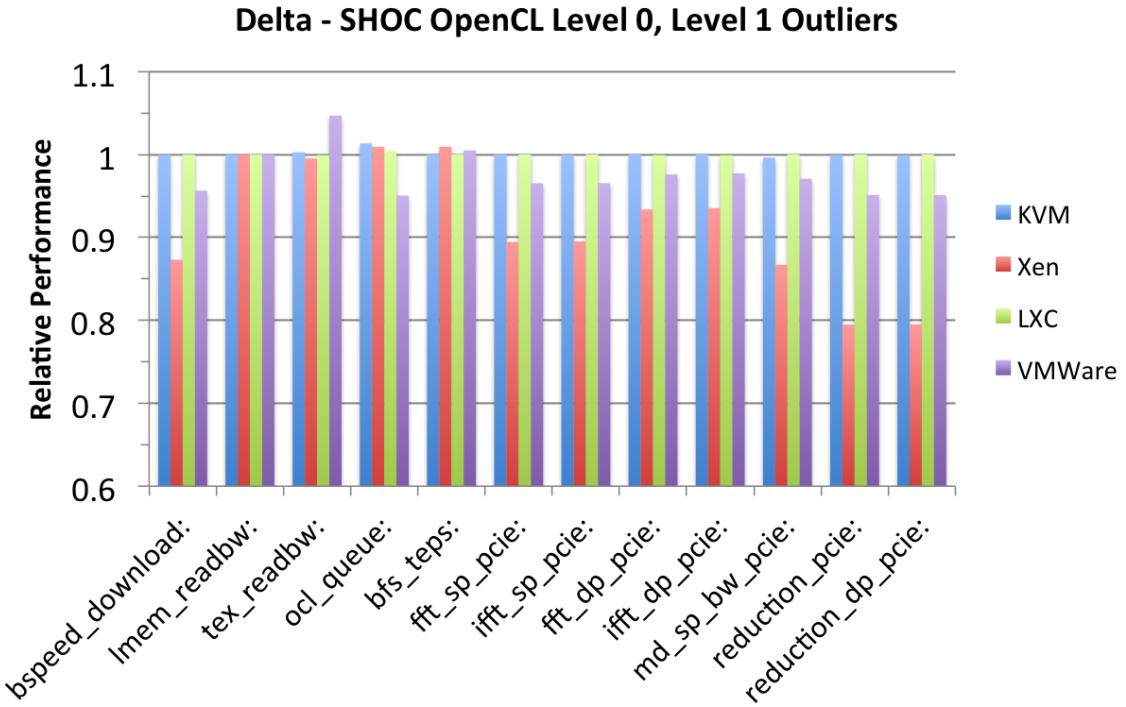


Figure 5.3 SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.

2136 PCIe portion of SHOC's benchmarks. This is unsurprising, since the Level 0 benchmarks
 2137 suggest PCIe bandwidth as the major source overhead. Still, results remain
 2138 consistent with the bspeed_download overhead observed in the Level 0 benchmarks,
 2139 further suggesting that host/device data movement is the major source of overhead.

2140 In Figure 5.2 we present the remaining SHOC Level 1 results as well as the SHOC
 2141 Level 2 results (S3D). In these results we begin to see an interesting trend, namely
 2142 that VMWare consistently outperforms the base system in the Spmv and S3D mi-
 2143 crobenchmarks on the Bespin system. We believe this to be a performance regression
 2144 in CentOS 6.4, rather than a unique improvement due to the VMWare ESXi hyper-
 2145 visor. When running the benchmarks on a non-virtualized Bespin system with Arch
 2146 Linux, the VMWare ESXi performance gains were erased. An interesting finding of
 2147 this was that Spmv was unique in this way - no other benchmarks were affected by

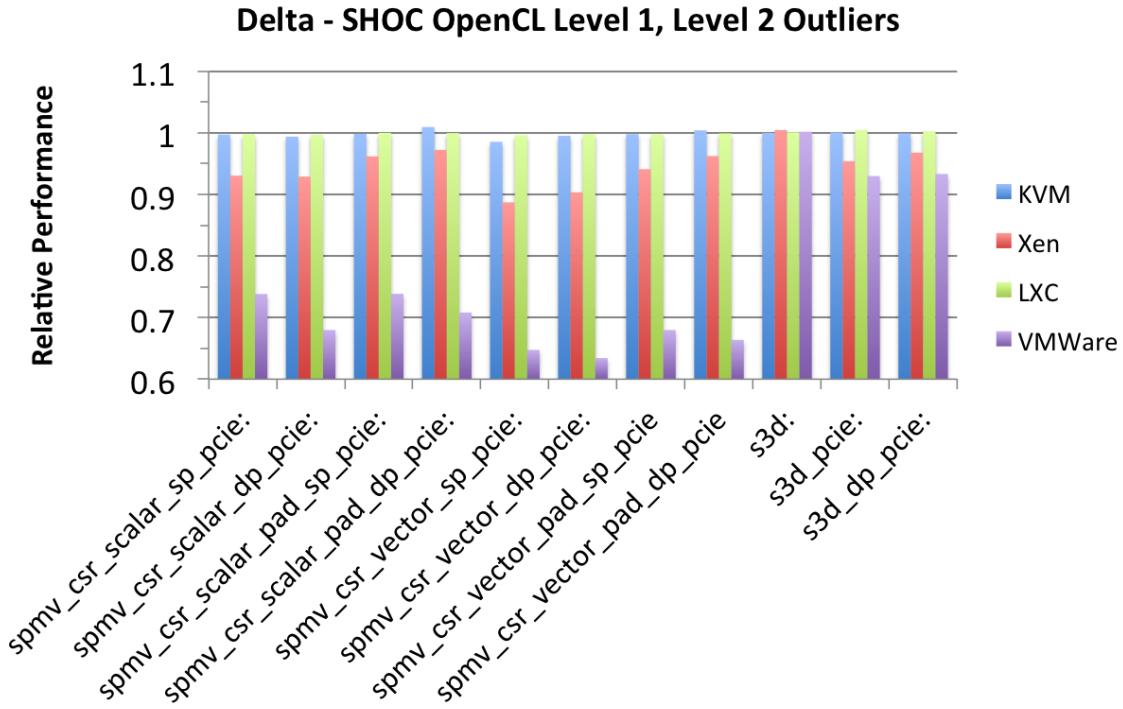


Figure 5.4 SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.

2148 this performance issue.

2149 Turning to the Delta system, in Figures 5.3 and 5.4, we show the same bench-
 2150 marks for the Delta system as was shown in Figures 5.1 and 5.2. Again, we find that
 2151 the same benchmarks are responsible for most of the overhead on the Delta system.
 2152 This is unsurprising, since PCIe was shown to be the source of the bulk of the over-
 2153 head. A major difference in the case of the Delta system, however, is the amount
 2154 of overhead. While the Bespin system saw overheads of approximately 1%, Delta's
 2155 overhead routinely jumps above 35%, especially in the case of the Spmv benchmark
 2156 for VMWare.

2157 On further examination, we determined that Xen was unable to activate IOMMU
 2158 large page tables on the Delta system. KVM successfully allocated 4k, 2M, and 1G
 2159 page table sizes, while Xen was limited to size 4k page tables. The Bespin system

2160 was able to take advantage of 4k, 2M, and 1G page sizes on both KVM and Xen. It
2161 appears that this issue is correctable and does not represent a fundamental limitation
2162 to the Xen hypervisor on the Nehalem/Westmere microarchitecture. While as a
2163 closed source product, we have limited insight into VMWare ESXi, we speculate that
2164 VMWare may be experiencing a similar issue on the Delta system and not on our
2165 Bespin system.

2166 In light of this, we broadly find that virtualization overhead across hypervisors and
2167 architectures is minimal. Questions remain as to the source of the exceptionally high
2168 overhead in the case of Xen and VMWare on the Delta system, but because KVM
2169 shows no evidence of this overhead, we believe the Westmere/Fermi architecture to
2170 be suitable for VGA passthrough in a cloud environment. In the case of the Bespin
2171 system, it is clear that VGA passthrough can be achieved across hypervisors with
2172 virtually no overhead.

2173 A surprising finding is that LXC showed little performance advantage over KVM,
2174 Xen, and VWMare. While we expected near-native performance from LXC we did
2175 not expect the hardware-assisted hypervisors to achieve such high performance. Still,
2176 LXC carries some advantages. In general, its maximum overheads are comparable to
2177 or less than KVM, Xen, and VMWare, especially in the case of the Delta system.

2178 **5.4.2 GPU-LIBSVM Performance**

2179 In Figures 5.5 and 5.6, we display our GPU-LIBSVM performance results for the
2180 Bespin (K20m) and Delta (C2075) systems. Using the NIPS 2003 gisette data set,
2181 we show the performance across 4 problems sizes, ranging from 1800 to 6000 training
2182 instances. The NIPS 2003 gisette dataset is a standard dataset that was used as a
2183 part of the NIPS 2003 feature selection challenge.

2184 KVM again performs well across both the Delta and Bespin systems. In the case

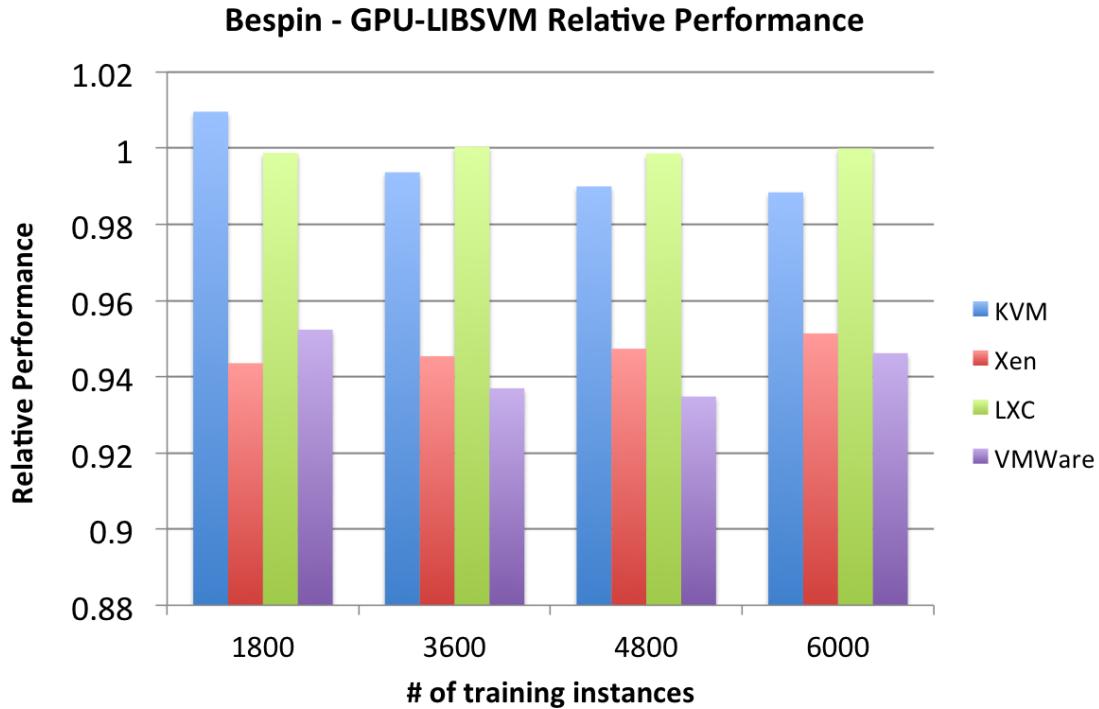


Figure 5.5 GPU-LIBSVM relative performance on Bespin system. Higher is better.

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 [197]. 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 substantially improve the performance of virtualized guests under some circumstances. LXC and VMWare perform 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 enable page sizes larger than 4k.

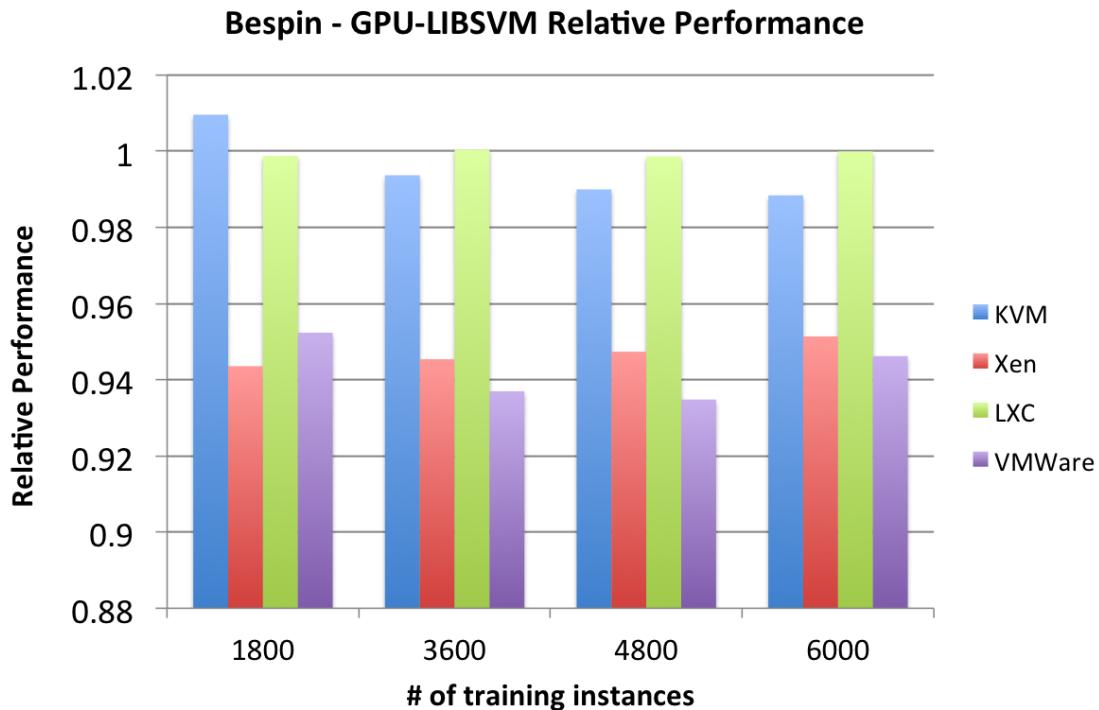


Figure 5.6 GPU-LIBSVM relative performance on Delta showing improved performance within virtual environments. Higher is better.

2197 5.4.3 LAMMPS Performance

2198 In Figures 5.7 and 5.8, we show the LAMMPS Rhodopsin protein simulation results.
 2199 LAMMPS is unique among our benchmarks, in that it exercises both the GPU and
 2200 multiple CPU cores. In keeping with the LAMMPS benchmarking methodology,
 2201 we execute the benchmark using 1 GPU and 1–8 CPU cores on the Bespin system,
 2202 selecting the highest performing configuration. In the case of the Delta system, we
 2203 execute the benchmark on 1–6 cores and 1 GPU, selecting the highest performing
 2204 configuration.

2205 Overall, LAMMPS performs well across both hypervisors and systems. Surpris-
 2206 ingly, LAMMPS showed better efficiency on the Delta system than the Bespin system,
 2207 achieving greater than 98% efficiency across the board, while Xen on the Bespin sys-

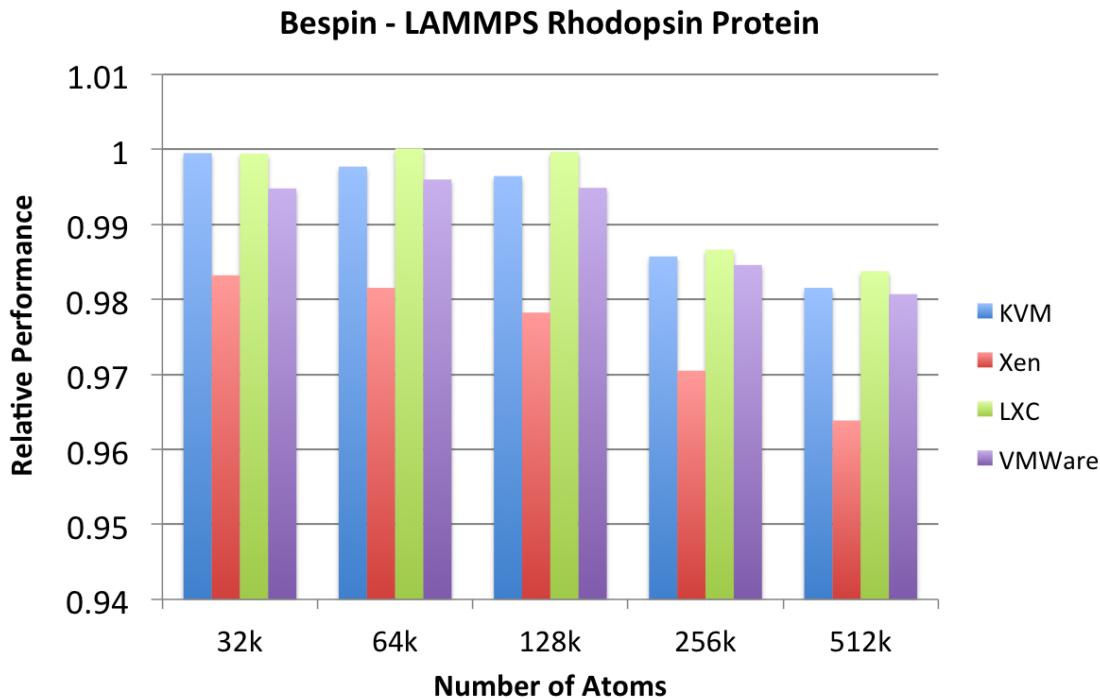


Figure 5.7 LAMMPS Rhodopsin benchmark relative performance for Bespin system. Higher is better.

tem 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.

5.4.4 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

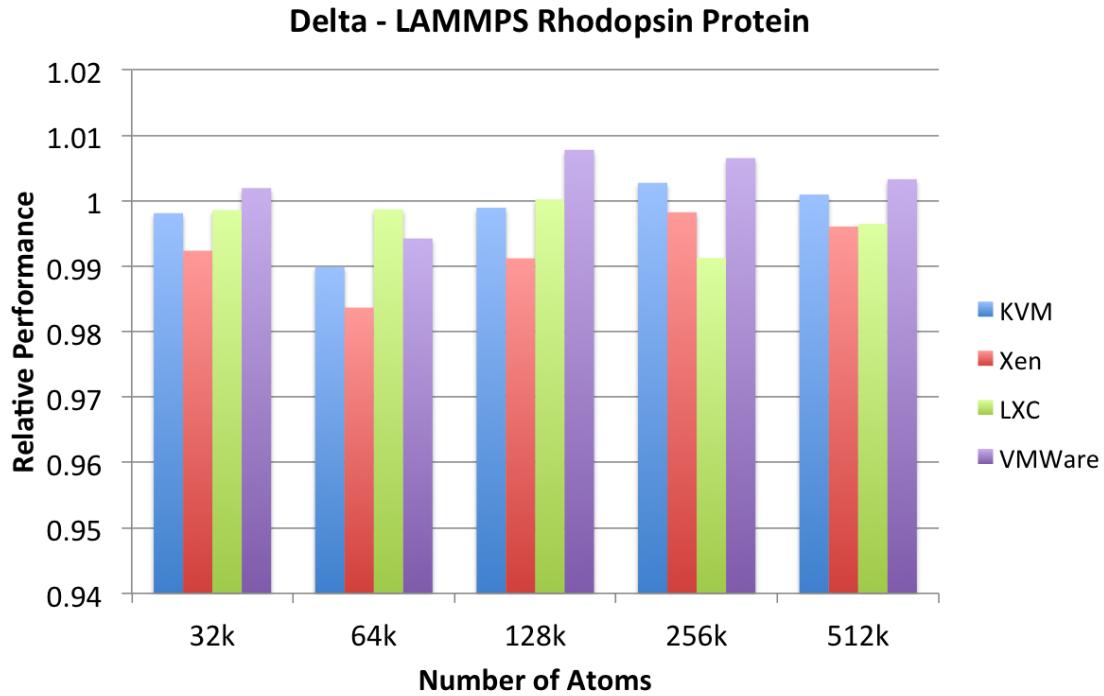


Figure 5.8 LAMMPS Rhodopsin benchmark relative performance for Delta system. Higher is better.

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, LXC, and VMWare.

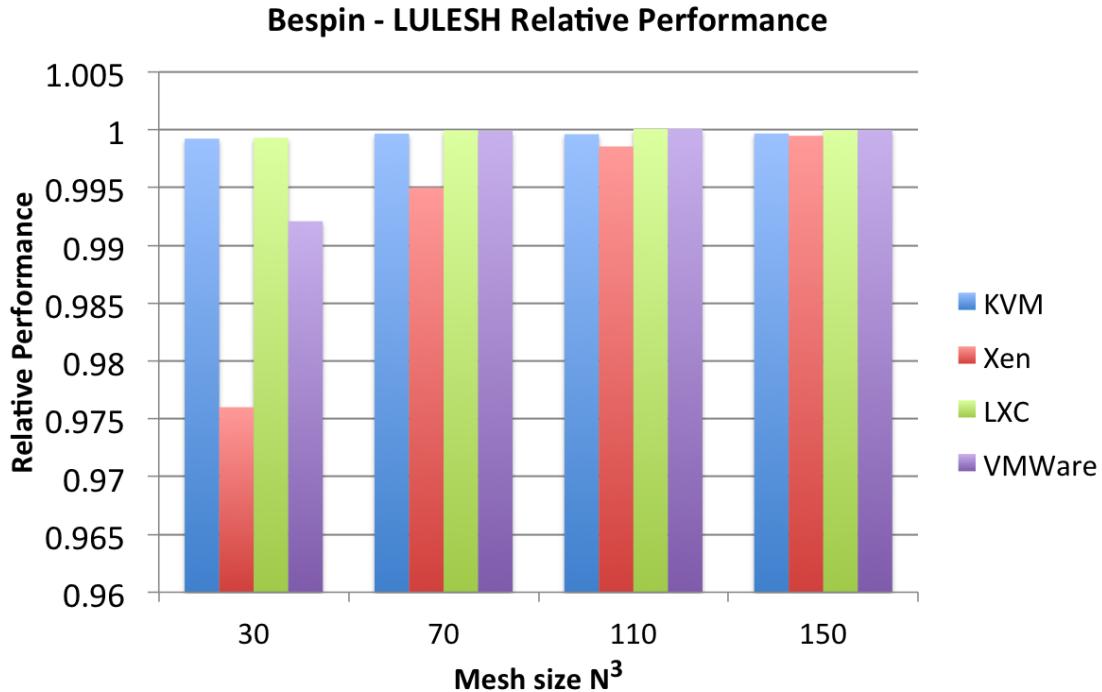


Figure 5.9 LULESH relative performance on Bespin. Higher is better.

2227 5.5 Lessons Learned

2228 Virtualizing performance-critical workloads has always proven controversial, whether
 2229 the workload is CPU-only [169] or CPU with GPU. From our Westmere results, we
 2230 can see that this criticism is in part legitimate, at times resulting in a performance
 2231 penalty of greater than 10%, especially in the case of the VMWare ESXi and Xen
 2232 hypervisors. We believe much of this to be fixable, especially in the case of the Xen
 2233 hypervisor.

2234 At the same time, however, we have shown that the Sandy Bridge processor genera-
 2235 tion has nearly erased those performance overheads, suggesting that old arguments
 2236 against virtualization for performance-critical tasks should be reconsidered. In light of
 2237 this, the primary lesson from this study is that VGA-passthrough to virtual machines
 2238 is achievable at low overhead, and across a variety of hypervisors and virtualization

2239 platforms. Virtualization performance remains inconsistent across hypervisors for the
2240 Westmere generation of processors, but starting with the Sandy Bridge architecture,
2241 performance and consistency increase dramatically. In the case of the Sandy Bridge
2242 architecture, even the lowest performing hypervisor, open source Xen, typically per-
2243 forms within 95% of the base case.

2244 This study has also yielded valuable insight into the merits of each hypervisor.
2245 KVM consistently yielded near-native performance across the full range of bench-
2246 marks. Its support for transparent hugepages resulted in slight performance boosts
2247 over-and-above even the base CentOS system in the case of the Delta system.

2248 VMWare's performance proved inconsistent across architectures, performing well
2249 in the case of Bespin, and relatively poorly in the case of the Delta system. Because
2250 hypervisor configurations were identical across systems, we can only speculate that
2251 VMWare's performance is aided by the virtualization improvements offered by the
2252 Sandy Bridge microarchitecture.

2253 The Xen hypervisor was consistently average across both architectures, perform-
2254 ing neither poorly nor extraordinarily well in any individual benchmark. Xen and
2255 VMWare ESXi are the only two hypervisors from this study that officially support
2256 VGA passthrough. As a result, PCI passthrough support in both Xen and VMWare is
2257 more robust than KVM. We expect that this advantage will not last long, as commer-
2258 cial solutions targeting PCI passthrough in KVM are becoming common, particularly
2259 with regard to SR-IOV and networking adapters.

2260 Linux Containers (LXC), consistently performed closest to the native case. This,
2261 of course, is not surprising given that LXC guests share a single kernel with their
2262 hosts. This performance comes at the cost of both flexibility and security, however.
2263 LXC is less flexible than its full virtualization counterparts, offering support for only
2264 Linux guests. More importantly, LXC device passthrough has security implications

2265 for multi-GPU systems. In the case of a multi-GPU-enabled host with NVIDIA
2266 hardware, both GPUs must be passed to the LXC guest in order to initialize the
2267 driver. This limitation may be addressable in future revisions to the NVIDIA driver.

2268 5.6 Directions for Future Work

2269 In this chapter we have characterized the performance of 4 common hypervisors across
2270 two generations of GPUs and two host microarchitectures, and across 3 sets of bench-
2271 marks. We showed the dramatic improvement in virtualization performance between
2272 the Fermi/Westmere and the Kepler/Sandy Bridge system, with the Sandy Bridge
2273 system typically performing within 1% of the base system. Finally, this study sought
2274 to characterize the GPU and CPU+GPU performance with carefully tuned hypervi-
2275 sor and guest configurations, especially with respect to NUMA. Improvements must
2276 be made to today’s hypervisors in order to improve virtual NUMA support. Finally,
2277 cloud infrastructure, such as OpenStack, must be capable of automatically allocating
2278 virtual machines in a NUMA-friendly manner in order to achieve acceptable results
2279 at cloud-scale.

2280 The next step in this work is to move beyond the single node to show that clus-
2281 ters of accelerators can be efficiently used with minimal overhead. This will require
2282 studies in high speed networking, particularly SR-IOV-enabled ethernet and Infini-
2283 band. Special attention is needed to ensure that latencies remain tolerable within
2284 virtual environments. Some studies have begun to examine these issues [198], but
2285 open questions remain.

2286 **Chapter 6**

2287 **Supporting High Performance**

2288 **Molecular Dynamics in Virtualized**

2289 **Clusters using IOMMU, SR-IOV,**

2290 **and GPUDirect**

2291 **6.1 Introduction**

2292 At present we stand at the inevitable intersection between High Performance Com-
2293 puting (HPC) and clouds. Various platform tools such as Hadoop and MapReduce,
2294 among others, have already percolated into data intensive computing within HPC [22].

2295 In addition, there are efforts to support traditional HPC-centric scientific computing
2296 applications in virtualized cloud infrastructure. There are a multitude of reasons for
2297 supporting parallel computation in the cloud [59], including features such as dynamic
2298 scalability, specialized operating environments, simple management interfaces, fault
2299 tolerance, and enhanced quality of service, to name a few. The growing importance

2300 of supporting advanced scientific computing using virtualized infrastructure can be
2301 seen by a variety of new efforts, including the NSF-funded Comet resource part of
2302 XSEDE at San Diego Supercomputer Center [199].

2303 Nevertheless, there exists a past notion that virtualization used in today's cloud
2304 infrastructure is inherently inefficient. Historically, cloud infrastructure has also done
2305 little to provide the necessary advanced hardware capabilities that have become al-
2306 most mandatory in supercomputers today, most notably advanced GPUs and high-
2307 speed, low-latency interconnects. The result of these notions has hindered the use
2308 of virtualized environments for parallel computation, where performance must be
2309 paramount.

2310 A growing effort is currently underway that looks to systematically identify and
2311 reduce any overhead in virtualization technologies. This effort has, thus far, proven
2312 to be a qualified success [169, 200], though further research is needed to address
2313 issues of scalability and I/O. Thus, we see a constantly diminishing overhead with
2314 virtualization, not only with traditional cloud workloads [201] but also with HPC
2315 workloads. While virtualization will almost always include some additional overhead
2316 in relation to its dynamic features, the eventual goal for supporting HPC in virtualized
2317 environments is to minimize what overhead exists whenever possible. To advance
2318 the placement of HPC applications on virtual machines, new efforts are emerging
2319 which focus specifically on key hardware now commonplace in supercomputers. By
2320 leveraging new virtualization tools such as IOMMU device passthrough and SR-IOV,
2321 we can now support the such advanced hardware as the latest Nvidia Tesla GPUs [202]
2322 as well as InfiniBand fabrics for high performance networking and I/O [203, 204].

2323 With the advances in hypervisor performance coupled with the newfound avail-
2324 ability of HPC hardware in virtual machines analogous to the most powerful super-
2325 computers used today, we see can see the possibility of a high performance cloud in-

2326 frastructure using virtualization. While our previous efforts in this area have focused
2327 on single-node advancements, it is now imperative to ensure real-world applications
2328 can also operate in distributed environments as found in today’s cluster and cloud
2329 infrastructures.

2330 Efforts to improve power efficiency and performance in data centers has led to
2331 more heterogeneous architectures. That move toward heterogeneity has, in turn, led
2332 to support for heterogeneity in the cloud. For example, Amazon EC2 supports GPU
2333 accelerators in EC2 [205], and OpenStack supports heterogeneity using flavors [206].
2334 These advancements in cloud-level support for heterogeneity combined with better
2335 support for high-performance virtualization makes the use of cloud for HPC much
2336 more feasible for a wider range of applications and platforms.

2337 In this paper we describe background a related work. Then, we describe a heteroge-
2338 neous cloud platform, based on OpenStack. This effort has been under development
2339 at USC/ISI since 2011 [163]. We describe our work towards integrating GPU and
2340 InfiniBand support into OpenStack, and we describe the heterogeneous scheduling
2341 additions that are necessary to support not only attached accelerators, but any cloud
2342 composed of heterogeneous elements.

2343 We then demonstrate running two molecular dynamics simulations, LAMMPS
2344 and HOOMD, in a virtual infrastructure complete with both Kepler GPUs and QDR
2345 InfiniBand. Both HOOMD and LAMMPS are used extensively in some of the world’s
2346 fastest supercomputers and represent example simulations that HPC supports today.
2347 We show that these applications are able to run at near-native speeds within a com-
2348 pletely virtualized environment, demonstrating just small performance impacts that
2349 are usually acceptable by many users. Furthermore, we demonstrate the ability of
2350 such a virtualized environment to support cutting edge software tools such as RDMA
2351 GPUDirect, illustrating how cutting-edge HPC technologies are also possible in a

2352 virtualized environment.

2353 Following these efforts, we hope to ensure upstream infrastructure projects such
2354 as OpenStack [128, 207] are able to make effective and quick use of these features,
2355 allowing users to build private cloud infrastructure to support high performance dis-
2356 tributed computational workloads.

2357 6.2 Background and Related Work

2358 Virtualization technologies and hypervisors have been seen widespread deployment
2359 in support of a vast array of applications. This ranges from public commercial Cloud
2360 deployments such as Amazon EC2 [208, 209], Microsoft Azure [210], and Google’s
2361 Cloud Platform [211] to private deployments within colocation facilities, corporate
2362 data centers, and even national scale cyber infrastructure initiatives. All these sup-
2363 port look to support various use cases and applications such as web servers, ACID
2364 and BASE databases, online object storage, and even distributed systems, to name a
2365 few.

2366 The use of virtualization and hypervisors specifically support various HPC so-
2367 lutions has been studied with mixed results. In [169], it is found that there is a
2368 great deal of variance between hypervisors when running various distributed memory
2369 and MPI applications, finding that KVM overall performed well across an array of
2370 HPC benchmarks. Furthermore, some applications may not fit well into default
2371 virtualized environments, such as High Performance Linpack [200]. Other studies
2372 have specifically looked at interconnect performance in virtualization and found the
2373 best-case scenario to be lacking [212] with up to 60% performance penalties with
2374 conventional techniques.

2375 Recently, various CPU architectures have added support for I/O virtualization

2376 mechanisms in the CPU ISA through the use of an I/O memory management unit
2377 (IOMMU). Often, this is referred to as PCI Passthrough, as it enabled devices on the
2378 PCI-Express bus to be passed directly to a specific virtual machine (VM). Specific
2379 hardware implementations include Intel’s VT-d [213], AMD’s IOMMU [214] from
2380 x86_64 architectures, and even more recently ARM System MMU [215]. All of these
2381 implementations effectively look to aid in the usage of DMA-capable hardware to be
2382 used within a specific virtual machine. Using these features, a wide array of hardware
2383 can be utilized directly within VMs and enable fast and efficient computation and
2384 I/O capabilities.

2385 With PCI Passthrough, a PCI device is handed directly to a running (or booting)
2386 VM, thereby relinquishing control of the device within the host entirely. This is
2387 different from typical VM usage where hardware is emulated in the host and used
2388 in a guest VM, such as with bridged ethernet adapters or emulated VGA devices.
2389 Performing PCI Passthrough requires the host to seize the device upon boot using a
2390 specialized driver to effectively block normal driver initialization. In the instance of
2391 the KVM hypervisor, this is done using the *vfio* and *pci_stub* drivers. Then, this driver
2392 relinquishes control to the VM, whereby normal device drivers initiate the hardware
2393 and enable the device for use by the guest OS.

2394 6.2.1 GPU Passthrough

2395 Nvidia GPUs comprise the single most common accelerator in the Nov 2014 Top 500
2396 List [8] and represent an increasing shift towards accelerators for HPC applications.
2397 Historically, GPU usage in a virtualized environment has been difficult, especially
2398 for scientific computation. Various front-end remote API implementations have been
2399 developed to provide CUDA and OpenCL libraries in VMs, which translate library
2400 calls to a back-end or remote GPU. One common use case of this is rCUDA [51], which

2401 provides a front-end CUDA API within a VM or any compute node, and then sends
2402 the calls via Ethernet or InfiniBand to a separate node with 1 or more GPUs. While
2403 this method is valid, it has the drawback of relying on the interconnect itself and the
2404 bandwidth available, which can be especially problematic on Ethernet. Furthermore,
2405 as this method consumes bandwidth, it can leave little remaining for MPI or RDMA
2406 routines, thereby constructing a bottleneck for some MPI+CUDA applications that
2407 depend on inter-process communication.

2408 Recently efforts have been seen to support such GPU accelerators within VMs
2409 using IOMMU technologies, with implementations now available with KVM [202],
2410 Xen [216] and VMWare [217]. These efforts have shown that GPUs can achieve up
2411 to 99% of their bare metal performance when passed to a virtual machine using PCI
2412 Passthrough. VMWare specifically shows how the such PCI Passthrough solutions
2413 perform well and are likely to outperform front-end Remote API solutions such as
2414 rCUDA within a VM [217]. While these works demonstrate PCI Passthrough perfor-
2415 mance across a range of hypervisors and GPUs, they have been limited to investigating
2416 single node performance until now.

2417 **6.2.2 SR-IOV and InfiniBand**

2418 With almost all parallel HPC applications, the interconnect fabric which enables fast
2419 and efficient communication between processors becomes a central requirement to
2420 achieving good performance. Specifically, a high bandwidth link is needed for dis-
2421 tributed processors to share large amounts of data across the system. Furthermore,
2422 low latency becomes equally important for ensuring quick delivery of small mes-
2423 sage communications and resolving large collective barriers within many parallelized
2424 codes. One such interconnect, InfiniBand, has become the most common implemen-
2425 tation used within the Top500 list. However previously InfiniBand was inaccessible

2426 to virtualized environments.

2427 Supporting I/O interconnects in VMs has been aided by Single Root I/O Virtu-
2428 alization (SR-IOV), whereby multiple virtual PCI functions are created in hardware
2429 to represent a single PCI device. These virtual functions (VFs) can then be passed
2430 to a VM and used as by the guest as if it had direct access to that PCI device. SR-
2431 IOV allows for the virtualization and multiplexing to be done within the hardware,
2432 effectively providing higher performance and greater control than software solutions.

2433 SR-IOV has been used in conjunction with Ethernet devices to provide high perfor-
2434 mance 10Gb TCP/IP connectivity within VMs [218], offering near-native bandwidth
2435 and advanced QoS features not easily obtained through emulated Ethernet offerings.
2436 Currently Amazon EC2 offers a high performance VM solution utilizing SR-IOV en-
2437 abled 10Gb Ethernet adapters. While SR-IOV enabled 10Gb Ethernet solutions offers
2438 a big forward in performance, Ethernet still does not offer the high bandwidth or low
2439 latency typically found with InfiniBand solutions.

2440 Recently SR-IOV support for InfiniBand has been added by Mellanox in the Con-
2441 nectX series adapters. Initial evaluation of SR-IOV InfiniBand within KVM VMs
2442 has proven has found point-to-point bandwidth to be near-native, but up to 30%
2443 latency overhead for very small messages [203, 219]. However, even with the noted
2444 overhead, this still signifies up to an order of magnitude difference in latency between
2445 InfiniBand and Ethernet with VMs. Furthermore, advanced configuration of SR-IOV
2446 enabled InfiniBand fabric is taking shape, with recent research showing up to a 30%
2447 reduction in the latency overhead [204]. However, real application performance has
2448 not yet been well understood until now.

2449 **6.2.3 GPUDirect**

2450 NVIDIA’s GPUDirect technology was introduced to reduce the overhead of data
2451 movement across GPUs [220, 221]. GPUDirect supports both networking as well as
2452 peer-to-peer interfaces for single node multi-GPU systems. The most recent imple-
2453 mentation of GPUDirect, version 3, adds support for RDMA over InfiniBand for
2454 Kepler-class GPUs.

2455 The networking component of GPUDirect relies on three key technologies: CUDA
2456 5 (and up), a CUDA-enabled MPI implementation, and a Kepler-class GPU (RDMA
2457 only). Both MVAPICH and OpenMPI support GPUDirect. Support for RDMA over
2458 GPUDirect is enabled by the MPI library, given supported hardware, and does not
2459 depend on application-level changes to a user’s code.

2460 In this paper, our GPUDirect work focuses on GPUDirect v3 for multi-node
2461 RDMA support. We demonstrate scaling for up to 4 nodes connected via QDR
2462 InfiniBand and show that GPUDirect RDMA improves both scalability and overall
2463 performance by approximately 9% at no cost to the end user.

2464 **6.3 A Cloud for High Performance Computing**

2465 With support for GPU Passthrough, SR-IOV, and GPUDirect, we have the building
2466 blocks for a high performance, heterogeneous cloud. In addition, other common
2467 accelerators (e.g. Xeon Phi [222]) have similarly been demonstrated in virtualized
2468 environments. Our vision is of a heterogeneous cloud, supporting both high speed
2469 networking and accelerators for tightly coupled applications.

2470 To this end we have developed a heterogeneous cloud based on OpenStack [128].
2471 In our previous work, we have demonstrated the ability to rapidly provision GPU,
2472 bare metal, and other heterogeneous resources within a single cloud [163]. Building

2473 on this effort we have added support for GPU passthrough to OpenStack as well as
2474 SR-IOV support for both ConnectX-2 and ConnectX-3 Infiniband devices. Mellanox
2475 separately supports an OpenStack InfiniBand networking plugin for OpenStack’s Neu-
2476 tron service [223], however the Mellanox plugin depends on the ConnectX-3 adapter.
2477 Our institutional requirements depend on ConnectX-2 SR-IOV support, requiring
2478 an independent implementation.

2479 OpenStack supports services for networking (Neutron), compute (Nova), identity
2480 (Keystone), storage (Cinder, Swift), and others. Our work focuses entirely on the
2481 compute service.

2482 Scheduling is implemented at two levels: the cloud-level and the node-level. In our
2483 earlier work, we have developed a cloud-level heterogeneous scheduler for OpenStack,
2484 allowing scheduling based on architectures and resources [163]. In this model, the
2485 cloud-level scheduler dispatches jobs to nodes based on resource requirements (e.g.
2486 Kepler GPU) and node-level resource availability.

2487 At the node, a second level of scheduling occurs to ensure that resources are
2488 tracked and not over-committed. Unlike traditional cloud paradigms, devices passed
2489 into VMs cannot be over-committed. We treat devices, whether GPUs or InfiniBand
2490 virtual functions, as schedulable resources. Thus, it is the responsibility of the indi-
2491 vidual node to track resources committed and report availability to the cloud-level
2492 scheduler. For reporting, we piggyback on top of OpenStack’s existing reporting
2493 mechanism to provide a low overhead solution.

2494 6.4 Benchmarks

2495 We selected two molecular dynamics (MD) applications for evaluation in this study:
2496 LAMMPS and HOOMD [224, 225]. These MD simulations are chosen to represent a

2497 subset of advance parallel computation for a number of fundamental reasons:

- 2498 ● MD simulations provide a practical representation of N-Body simulations, which
2499 is one of the major computational *Dwarfs* [226] in parallel and distributed com-
2500 puting.
- 2501 ● MD simulations are one of the most widely deployed applications on large scale
2502 supercomputers today.
- 2503 ● Many MD simulations have a hybrid MPI+CUDA programming model, which
2504 has often become commonplace in HPC as the use of accelerators increases.

2505 As such, we look to LAMMPS and HOOMD to provide a real-world example for
2506 running cutting-edge parallel programs on virtualized infrastructure. While these
2507 applications by no means represent all parallel scientific computing efforts (as justi-
2508 fied by the 13 Dwarfs defined in [226]), we hope these MD simulators offer a more
2509 pragmatic viewpoint than traditional synthetic HPC benchmarks such as High Per-
2510 formance Linpack.

2511 **LAMMPS** The Large-scale Atomic/Molecular Parallel Simulator is a well-understood
2512 highly parallel molecular dynamics simulator. It supports both CPU and GPU-
2513 based workloads. Unlike many simulators, both MD and otherwise, LAMMPS is
2514 heterogeneous. It will use both GPUs and multicore CPUs concurrently. For this
2515 study, this heterogeneous functionality introduces additional load on the host, allow-
2516 ing LAMMPS to utilize all available cores on a given system. Networking in LAMMPS
2517 is accomplished using a typical MPI model. That is, data is copied from the GPU
2518 back to the host and sent over the InfiniBand fabric. No RDMA is used for these
2519 experiments.

2520 **HOOMD-blue** The Highly Optimized Object-oriented Many-particle Dynamics
2521 – Blue Edition is a particle dynamics simulator capable of scaling into the thou-
2522 sands of GPUs. HOOMD supports executing on both CPUs and GPUs. Unlike
2523 LAMMPS, HOOMD is homogeneous and does not support mixing of GPUs and
2524 CPUs. HOOMD supports GPUDirect using a CUDA-enabled MPI. In this paper
2525 we focus on HOOMD’s support for GPUDirect and show its benefits for increasing
2526 cluster sizes.

2527 6.5 Experimental Setup

2528 Using two molecular dynamics tools, LAMMPS [224] and HOOMD [225], we demon-
2529 strate a high performance *system*. That is, we combine PCI passthrough for Nvidia
2530 Kepler-class GPUs with QDR Infiniband SR-IOV and show that high performance
2531 molecular dynamics simulations are achievable within a virtualized environment.

2532 For the first time, we also demonstrate Nvidia GPUDirect technology within such
2533 a virtual environment. Thus, we look to not only illustrate that virtual machines
2534 provide a flexible high performance infrastructure for scaling scientific workloads in-
2535 cluding MD simulations, but also that the latest HPC features and programming
2536 environments are also available in this same model.

2537 6.5.1 Node configuration

2538 To support the use of Nvidia GPUs and InfiniBand within a VM, specific and exact
2539 host configuration is needed. This node configuration is illustrated in Figure 6.1.
2540 While our implementation is specific to the KVM hypervisor, this setup represents a
2541 design that can be hypervisor agnostic.

2542 Each node in the testbed uses CentOS 6.4 with a 3.13 upstream Linux kernel for

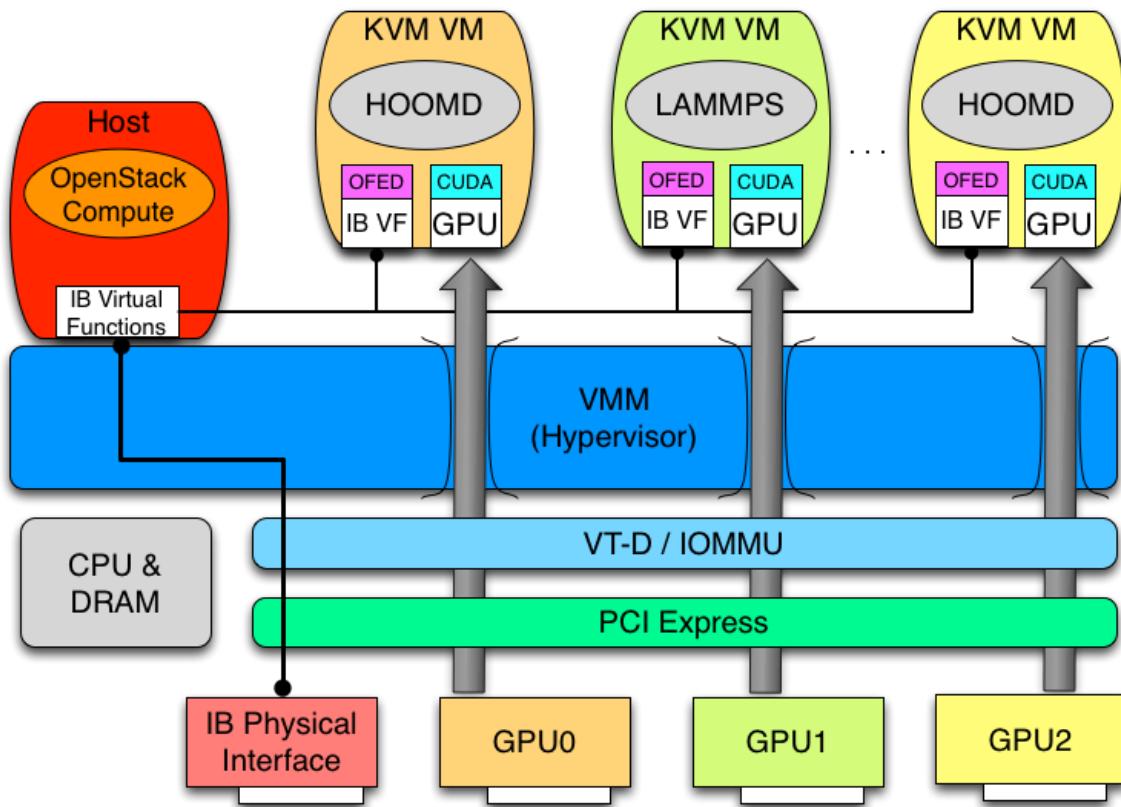


Figure 6.1 Node PCI Passthrough of GPUs and InfiniBand

2543 the host OS, along with the latest KVM hypervisor, QEMU 2.1, and the *vfio* driver.
 2544 Each Guest VM runs CentOS 6.4 with a stock 2.6.32-358.23.2 kernel. A Kepler GPU
 2545 is passed through using PCI Passthrough and directly initiated within the VM via
 2546 the Nvidia 331.20 driver and CUDA release 5.5. While this specific implementation
 2547 used only a single GPU, it is also possible to include as many GPUs as one can fit
 2548 within the PCI Express bus if desired. As the GPU is used by the VM, an on-board
 2549 VGA device was used by the host and a standard Cirrus VGA was emulated in the
 2550 guest OS.

2551 With using SR-IOV, the OFED drivers version 2.1-1.0.0 are used with Mellanox
 2552 ConnectX-3 VPI adapter with firmware 2.31.5050. The host driver initiates 4 VFs,
 2553 one of which is passed through to the VM where the default OFED mlnx_ib drivers

2554 are loaded.

2555 6.5.2 Cluster Configuration

2556 Our test environment is composed of 4 servers each with a single Nvidia Kepler-
2557 class GPU. Two servers are equipped with K20 GPUs, while the other two servers
2558 are equipped with K40 GPUs, demonstrating the potential for a more heterogeneous
2559 deployment. Each server is composed of 2 Intel Xeon E5-2670 CPUs, 48GB of DDR3
2560 memory, and Mellanox ConnectX-3 QDR InfiniBand. CPU sockets and memory are
2561 split evenly between the two NUMA nodes on each system. All InfiniBand adapters
2562 use a single Voltaire 4036 QDR switch with a software subnet manager for IPoIB
2563 functionality.

2564 For these experiments, both the GPUs and InfiniBand adapters are attached to
2565 NUMA node 1 and both the guest VMs and the base system utilized identical software
2566 stacks. Each guest was allocated 20 GB of RAM and a full socket of 8 cores, and
2567 pinned to NUMA node 1 to ensure optimal hardware usage. While all VMs are
2568 capable of login via the InfiniBand IPoIB setup, a 1Gb Ethernet network was used
2569 for all management and login tasks.

2570 For a fair and effective comparison, we also use a native environment without any
2571 virtualization. This native environment employs the same hardware configuration,
2572 and like the Guest OS runs CentOS 6.4 with the stock 2.6.32-358.23.2 kernel.

2573 6.6 Results

2574 In this section, we discuss the performance of both the LAMMPS and HOOMD
2575 molecular dynamics simulation tools when running within a virtualized environment.
2576 Specifically, we scale each application to 32 cores and 4 GPUs, both in a native bare-

2577 metal and virtualized environments. Each application set was run 10 times, with the
 2578 results averaged accordingly.

2579 **6.6.1 LAMMPS**

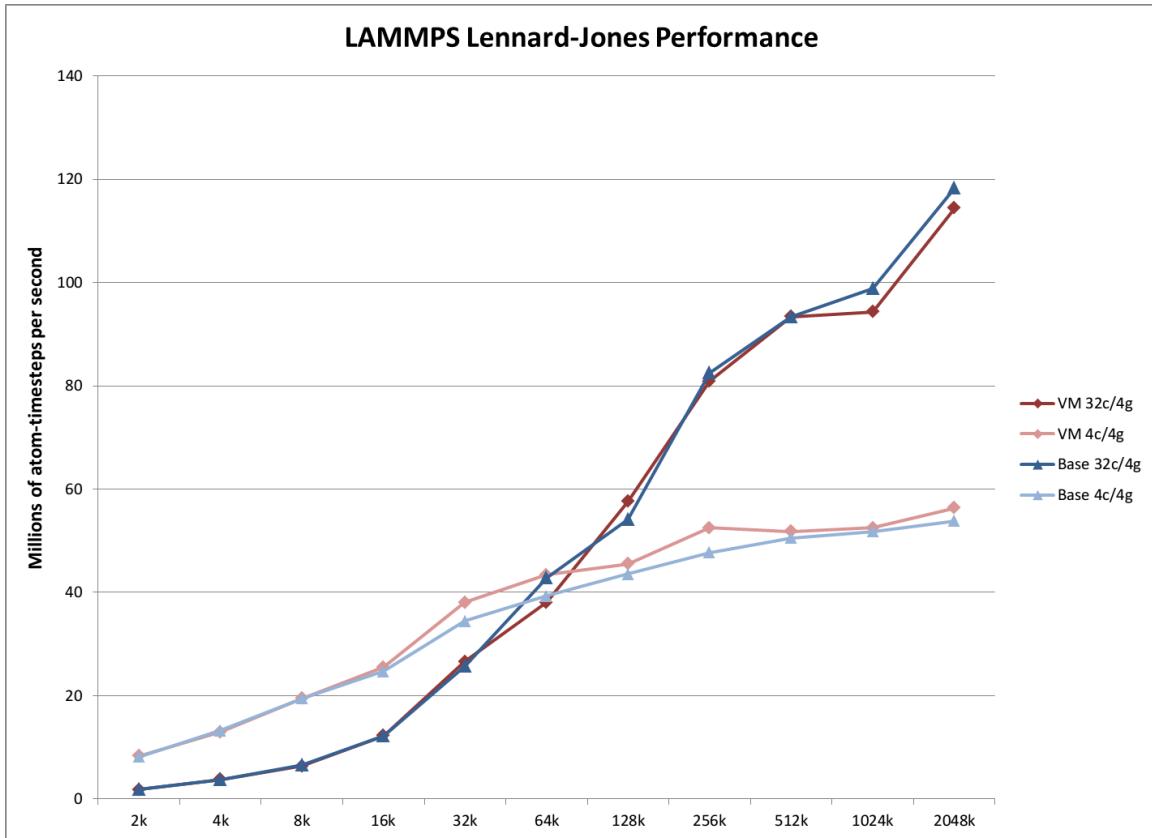


Figure 6.2 LAMMPS LJ Performance

2580 Figure 6.2 shows one of the most common LAMMPS algorithms used; the Lennard-
 2581 Jones potential (LJ). This algorithm is deployed in two main configurations - a 1:1
 2582 core to GPU mapping, and a 8:1 core to GPU mapping. With the LAMMPS GPU
 2583 implementation, a delicate balance between GPUs and CPUs is required to find the
 2584 optimal ratio for fastest computation, however here we just look at the two most
 2585 obvious choices. With small problem sizes, the 1:1 mapping outperforms the more
 2586 complex core deployment, as the problem does not require the additional complexity

2587 provided with multi-core solution. As expected the multi-core configuration quickly
2588 offers better performance for larger problem sizes, achieving roughly twice the perfor-
2589 mance with all 8 available cores. This is largely due to the availability of all 8 cores
2590 to keep the GPU running 100% with continual computation.

2591 The important factor for this manuscript is the relative performance of the virtu-
2592 alized environment. From the results, it is clear the VM solution performs very well
2593 compared to the best-case native deployment. For the multi-core configuration across
2594 all problem sizes, the virtualized deployment averaged 98.5% efficiency compared to
2595 native. The single core per GPU deployment reported better-than native perfor-
2596 mance at 100% native. This is likely due to caching effects, but further investigation
2597 is needed to fully identify this occurrence.

2598 Another common LAMMPS algorithm, the Rhodopsin protein in solvated lipid
2599 bilayer benchmark (Rhodo), was also run with results given in Figure 6.3. As with
2600 the LJ runs, we see the multi-core to GPU configuration resulting in higher computa-
2601 tional performance for the larger problem sizes compared to the single core per GPU
2602 configuration, as expected.

2603 Again, the overhead of the virtualized configuration remains low across all con-
2604 figurations and problem sizes, with an average 96.4% efficiency compared to native.
2605 Interestingly enough, we also see the performance gap decrease as the problem size
2606 increases, with the 512k problem size in yielding 99.3% of native performance. This
2607 finding leads us to extrapolate that a virtualized MPI+CUDA implementation would
2608 scale to a larger computational resource with similar success.

2609 **6.6.2 HOOMD**

2610 In Figure 6.4 we show the performance of a Lennard-Jones liquid simulation with 256K
2611 particles running under HOOMD. HOOMD includes support for CUDA-aware MPI

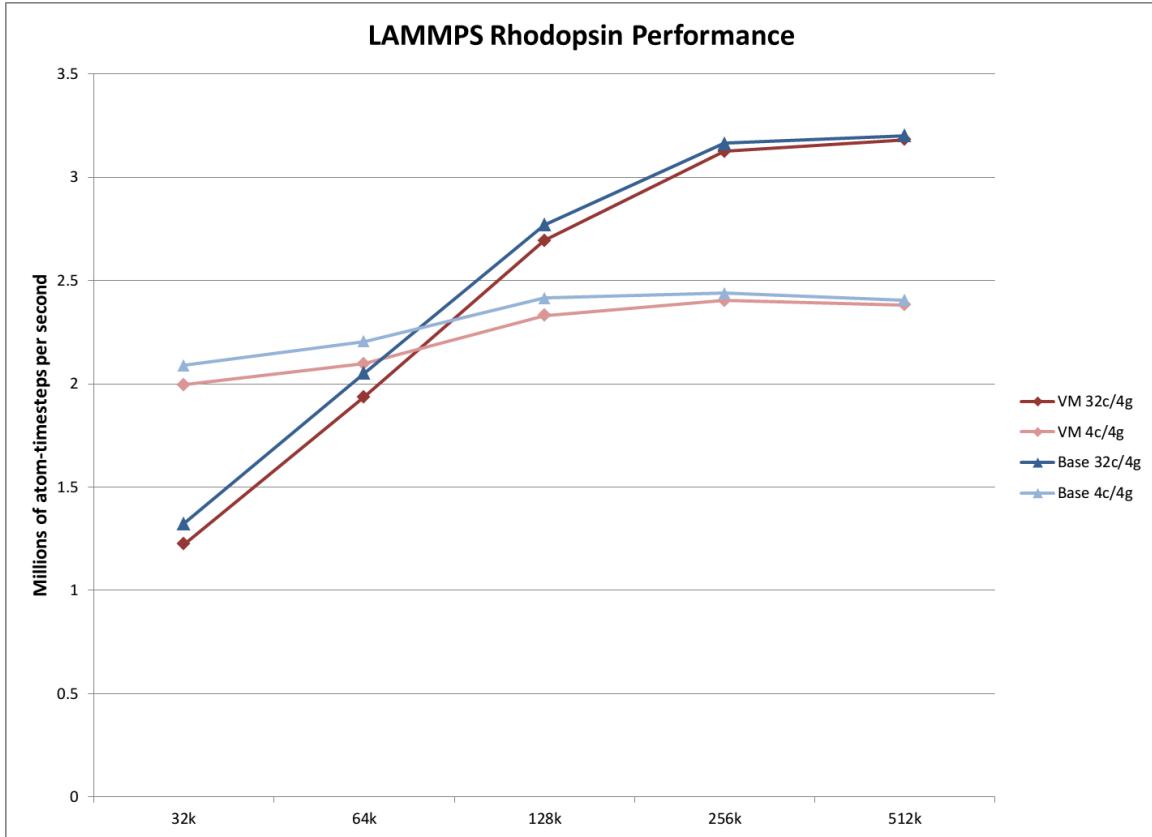


Figure 6.3 LAMMPS RHODO Performance

2612 implementations via GPUDirect. The MVAPICH 2.0 GDR implementation enables
 2613 a further optimization by supporting RDMA for GPUDirect. From Figure 6.4 we
 2614 can see that HOOMD simulations, both with and without GPUDirect, perform very
 2615 near-native. The GPUDirect results at 4 nodes achieve 98.5% of the base system's
 2616 performance. The non-GPUDirect results achieve 98.4% efficiency at 4 nodes. These
 2617 results indicate the virtualized HPC environment is able to support such complex
 2618 workloads. While the effective testbed size is relatively small, it indicates that such
 2619 workloads may scale equally well to hundreds or thousands of nodes.

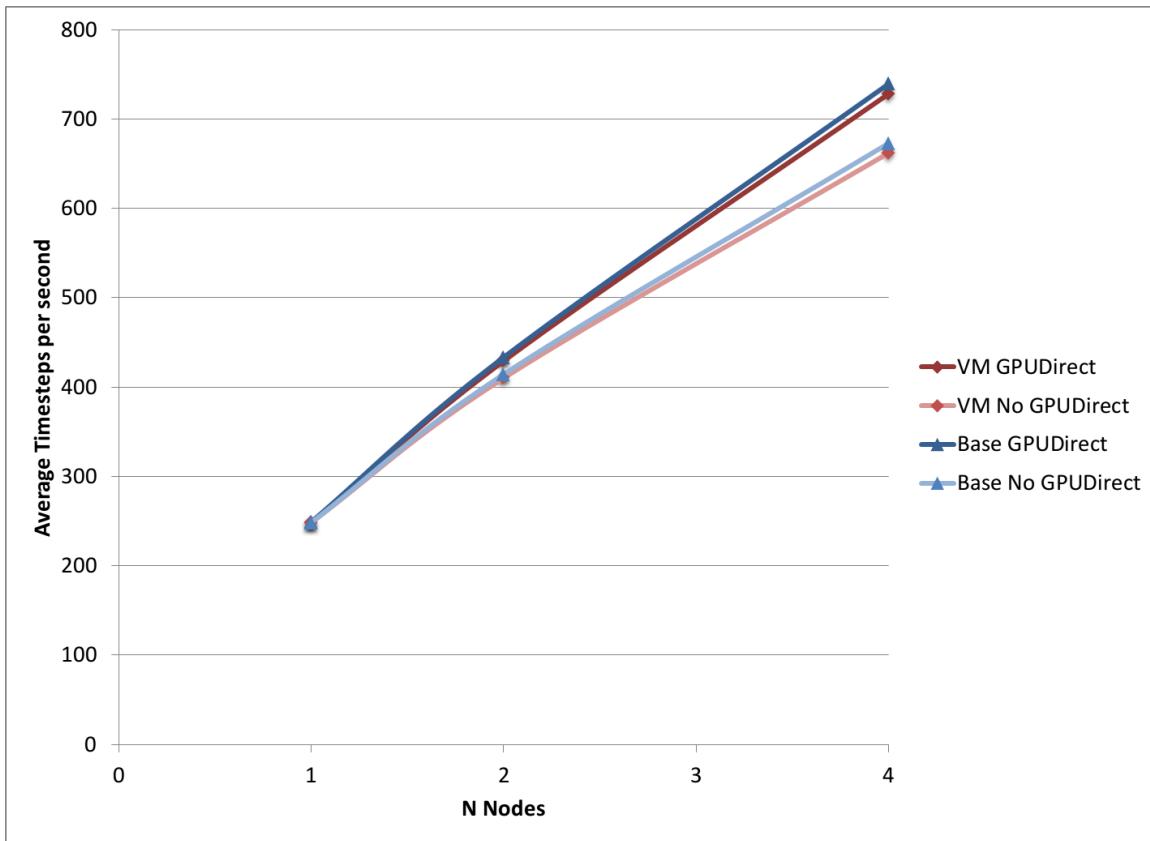


Figure 6.4 HOOMD LJ Performance with 256k Simulation

6.7 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

2630 the non-GPUDirect implementation, achieving a 9% performance boost in both the
2631 native and virtualized experiments. While GPUDirect's performance impact has been
2632 well evaluated previously [220], it is the author's belief that this manuscript represents
2633 the first time GPUDirect has been utilized in a virtualized environment.

2634 Another interesting finding of running LAMMPS and HOOMD in a virtualized
2635 environment is as workload scales from a single node to 32 cores, the overhead does
2636 not increase. These results lend credence to the notion that this solution would also
2637 work for a much larger deployment. Specifically, it would be possible to expand
2638 such computational problems to a larger deployment in FutureGrid [227], Chameleon
2639 Cloud [228], or even the planned NSF Comet machine at SDSC, scheduled to provide
2640 up to 2 Petaflops of computational power. Effectively, these results help support
2641 the theory that a majority of HPC computations can be supported in virtualized
2642 environment with minimal overhead.

2643 6.8 Chapter Summary

2644 With the advent of cloud infrastructure, the ability to run large-scale parallel sci-
2645 entific applications has become possible but limited due to both performance and
2646 hardware availability concerns. In this work we show that advanced HPC-oriented
2647 hardware such as the latest Nvidia GPUs and InfiniBand fabric are now available
2648 within a virtualized infrastructure. Our results find MPI + CUDA applications such
2649 as molecular dynamics simulations run at near-native performance compared to tra-
2650 ditional non-virtualized HPC infrastructure, with just an averaged 1.9% and 1.5%
2651 overhead for LAMMPS and HOOMD, respectively. Moving forward, we show the
2652 utility of GPUDirect RDMA for the first time in a cloud environment with HOOMD.
2653 Effectively, we look to pave the way for large-scale virtualized cloud Infrastructure

2654 to support a wide array of advanced scientific computation commonly found running
2655 on many supercomputers today. Our efforts leverage these technologies and provide
2656 them in an open source Infrastructure-as-a-Service framework using OpenStack.

2657 **Chapter 7**

2658 **Virtualization advancements to
2659 support HPC applications**

2660 Many of the previous chapters have focused on identifying and decreasing the per-
2661 formance gap that exists with running HPC workloads in virtualized infrastructure,
2662 as compared to native HPC environments without virtualization. While this is a
2663 significant technical challenge to overcome, the use of virtualization itself has the
2664 potential to offer additional benefits to HPC infrastructure. In this chapter, we look
2665 to identify research, design, and future implementation of advanced virtualization
2666 techniques to enable a new class of HPC infrastructure with added performance and
2667 features that has yet to be realized. It may be possible for these advancements, once
2668 implemented to have an impact not only on cloud infrastructure, but also a dedicated
2669 HPC environment for added usability and performance.

2670 7.1 Memory Page Table Optimizations

2671 As we've seen both in Chapter 3 as well as in other supported literature [43], virtual-
2672 ization of memory structures becomes a point of contention and potential overhead.
2673 This is often due to the extensive effort a hypervisor has to perform in order to
2674 translate memory addresses from guest-virtual addresses to host-virtual addresses,
2675 and then again to machine-physical addresses. Historically, this was handled using
2676 Shadow Page tables [229], which eliminate the need for emulation of physical memory
2677 inside the VM by creating a page table mapping from guest virtual to machine mem-
2678 ory. However, these page tables are not walkable by hardware such as a transition
2679 lookaside buffer (TLB), and as such guest OS page tables require updating of the
2680 shadow page table. This can be costly not only in the additional management, but
2681 also notably by the cost of VMexit and VMentry calls.

2682 Recently, Intel and AMD have implemented Extended Page Tables (EPT) and
2683 nested paging, respectively. With EPT, support is added to allow the TLB hardware
2684 to keep track of both guest pages and hypervisor pages concurrently, effectively re-
2685 moving the need for shadow page table. The downside of EPT and nested paging
2686 occurs when there is a TLB miss (the page isn't in the TLB), and as such each miss
2687 requires a walk through each VMM nested paging, effectively creating TLB miss cost
2688 of 16 table walks (instead of just 3-4) for 4k pages. While many applications find
2689 this TLB miss additional cost less much less than that of managing shadow page
2690 tables, it still can lead to a significant gap in performance between non-virtualized
2691 applications, especially as VM count or an application's memory footprint increase.

2692 One potential way to decrease the chance of a TLB miss (and therefore the cost
2693 of a miss) is by using a larger page size. By default, x86 hardware uses 4k pages
2694 sizes, but newer hardware can support 2M and 1G page sizes as well, effectively

2695 named *transparent huge pages* or THP. Using the KVM hypervisor with transparent
2696 hugepages enabled, we can create guest VMs backed entirely 2M hugepages [197]. We
2697 can also enable transparant hugepage support within the guest as well, to have the
2698 entire guest OS (including kernel and modules) using 2M pages.

2699 The result of THP-enabled guest VMs can be significant. With huge pages on Intel
2700 x86 CPUs with EPT, there exists an entirely separate TLB for hugepages as well. This
2701 will naturally alleviate TLB pressure and therefore reduce TLB contention between
2702 guest and host operating systems. Because 2M pages provide larger addressable
2703 memory, the size of the page tables themselves are also decreased. Effectively, this
2704 reduces the TLB miss cost from 4 to 3 page table walks, which when handling a VM
2705 TLB miss, then requires only 15 walks to the default 24. This in effect can have a
2706 significant improvement in VM performance, as hypothesized in [197].

2707 To evaluate the effect of 2M transparent huge pages on guest performance, we
2708 leverage the same KVM setup in Chapter 5 on the Bespin hardware. Specifically, THP
2709 is switched both in the host as well as the guest OS kernels, and the same LibSVM
2710 application using GPUs is re-run. The libSVM GPU application can have significant
2711 memory requirements, as large chunks of the support vector machine datasets are
2712 stored in CPU memory and tranferred in a sudo-random order to and from GPU
2713 memory, making it an ideal application to use for evaluating THP.

2714 The results of running libSVM in a THP-enabled VM, a VM with no THP, and
2715 natively without virtualization are displayed in Figure 7.1. Comparing first just
2716 the KVM results without THP to the native solution, we can see the impact of
2717 THP the overal application runtime, especially at larger problem sizes (6000 training
2718 sets). However, when TLB is enabled in the guest and host, we actually see the
2719 KVM VM solution *outperform* the native solution. This is because guest privileged
2720 OS memory used to buffer to/from GPU memory is backed by 2M pages, instead



Figure 7.1 Transparent Huge Pages with KVM

of the normal 4k pages as in the native solution. The result is less TLB misses during application runtime compared to 4k TLB misses in a native solution, resulting in improved performance. While this is likely a special case for THP usage with the libSVM application, 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 workloads.

2728 7.2 Live Migration Mechanisms

2729 Migration of VMs represents one of the fundamental advantages to virtualization, and
2730 also one of the greatest challenges to efficiency. With VM migration, the complete VM
2731 state is copied from a source to a unallocated destination host, where disk, memory,
2732 and network connections are kept intact. For disk continuity, a distributed and/or
2733 shared filesystem is utilized, most commonly but not exclusively NFS, where both
2734 the source and destination hosts have access to the VM disk. Network continuity is
2735 preserved so long as the destination guest is within the same LAN and generates an
2736 unsolicited ARP reply to maintain the original IP after migration. VM vCPU states
2737 and machine states are recorded from the source and quickly sent to the destination
2738 host when the VM is paused. For live migration, the source VM is paused only
2739 after all state and memory contents are copied to the destination. The last of the
2740 dirtied memory pages are copied over, and the newly formed destination VM is then
2741 un-paused. This pause and transfer time represents the entirety of a VM downtime
2742 during live migration, and is often at or under 100 milliseconds across commodity
2743 Ethernet networks (given a low memory utilization).

2744 VM memory transfers are often the main performance consideration for overhead
2745 during live migration. This is not only due to the potentially large amount of memory
2746 to be sent across the network, but the veracity at which the memory is changed. This
2747 is defined directly by the amount of main memory allocated (or in use) by the source
2748 VM. However, as a VM's memory is sent, the VM is still running and therefore memory
2749 pages can be dirtied, creating the need for any written page to be retransmitted.
2750 Given a small network and a memory bound processes running within a VM, this
2751 can be an infinitely long process of page dirtying. Many live migration strategies
2752 provide an iterative timeout mechanism to avoid this infinite state, but this will lead

2753 to increased downtime during migration.

2754 The copying of memory pages for live migration can take multiple implemen-
2755 tations. Three common options are summarized below:

- 2756 ● Pre-copy Migration: All memory pages are transmitted to the destination before
2757 the VM is paused. The hypervisor will note and track all dirtied memory pages,
2758 and retransmit those pages in iterative rounds. The rounds end when either no
2759 dirtied pages exist or a max iteration count has been reached. The VM state
2760 is then transmitted and resumed on the destination. This method was the first
2761 live migration technique used in [230].
- 2762 ● Post-copy Migration: The Vm state is immediately paused and sent to the
2763 destination hypervisor, and immediately resumed. If the new destination VM
2764 generates a page fault, the VM is paused, and faulted pages are transmitted
2765 across the network on demand from the source and the VM resumed. This
2766 methodology is proposed for use in the Xen hypervisor by Hines et al [231].
- 2767 ● Hybrid-copy Migration: Provides a compromise solution to memory paging.
2768 First, single copy of the VM memory pages, or a subset of known-necessary
2769 memory pages are sent to the destination. Then the source VM is paused,
2770 its VM state sent to the destination, and resumed on the destination. Known
2771 dirtied source pages, or missing pages are then copied to the destination upon a
2772 triggered page fault utilizing the same mechanism as post-copy migration. An
2773 example of hybrid migration can be found via Lu et al [232].

2774 While pre-copy migration is the traditional and most used live migration tech-
2775 nique, there are opportunities about to implement other migration techniques to
2776 advance the mobility of distributed computing in high performance virtual clusters
2777 with virtualization.

2778 7.2.1 RDMA-enabled VM Migration

2779 Currently, most live migration in production environments occur over TCP/IP con-
2780 nections, largely due to the prevalence of commodity Ethernet connections within
2781 cloud infrastructure. However even if RDMA-capable interconnects are available in
2782 such infrastructure as described with InfiniBand in Chapter 6, live migration still
2783 usually occurs over TCP/IP. For the case of InfiniBand, this is via IP over InfiniBand
2784 (IPoIB) [233], which can be an inefficient use of the interconnect bandwidth and add
2785 extra latency [234].

2786 The time it takes to migrate the memory contents from a source to destination
2787 VM can be significantly decreased by using an RDMA based mechanisms. Huang et
2788 al first provided a proposed pre-copy method for RDMA-based migration using the
2789 Xen hypervisor [235]. Specifically, they found a 80% decrease in migration time with
2790 RDMAwrite operations. This speedup is largely due to the removal of overhead nec-
2791 essary for processing TCP/IP communications, largely in CPU utilization for copying
2792 buffers, packet processing, and the included context switch overhead when competing
2793 for resources in a CPU-bound application (which are common in HPC environments).

2794 The live migration algorithm proposed in [235] uses the standard pre-copy mecha-
2795 nism that sends the entire memory contents across the network as RDMA operations,
2796 then iteratively copies dirtied pages before switching the running states. As discussed
2797 in the previous section, a post-copy migration strategy may have benefits for quick
2798 VM migration in high performance virtual clusters, or even for VM cloning as de-
2799 scribed later in Section 7.3. Furthermore, efforts in Chapter 3 have found that the
2800 Xen hypervisor is not best suited for HPC workloads. As such, there is a need to
2801 redefine the use of RDMA for VM migration using a hybrid post-copy mechanism in
2802 a high performance hypervisor.

- 2803 ● Transfer initial CPU state, registers
- 2804 ● Start the page table pages translation process: MFN to PFN and use copy-base
2805 approach
- 2806 ● Concurrently, allocate remote memory on destination VM.
- 2807 ● Set up other machine state settings in destination
- 2808 ● Start destination VM, pause source VM.
- 2809 ● Initiate RDMAwrite of entire memory contents from source to destination
- 2810 ● As page faults occur in destination VM, catch faults and perform RDMAread
2811 requesting pages.

2812 Currently efforts are underway to provide post-copy live migration in KVM/QEMU,
2813 using the `migrate_set_capability x-postcopy-ram` on mode within KVM [236].
2814 This method uses the Linux `userfaultfd` kernel mechanisms from a kernel 4.3 or newer.
2815 At the start, all memory blocks are registered as `userfaultfd` so all faults cause the
2816 running thread to pause. In kernel space, the missing page is requested from the
2817 sender, which prioritized over other pages being sent and is returned and mapped
2818 to the destination guest memory space and the thread or process is unpause. This
2819 mechanism operates asynchronously, so that multiple outstanding page faults will not
2820 stop other executables within the VM. As such, the `xpostcopy-ram` KVM extention
2821 has been identified as an ideal place to implement an RDMA based implementation.

2822 To provide RDMA functionality, the InfiniBand interconnect could first be used
2823 as a proof-of-concept. This choice is due to InfiniBand's rise in popularity, increased
2824 prevalence in virtualized systems with SR-IOV as noted in Chapter 6, and RMDA
2825 functionality. However, other interconnect options exist that may be better suited

2826 for enhanced functionality and performance, such as Intel's new Omnipath [237], or
2827 an Ethernet solution such as RoCE [238], to name a few alternatives. While RDMA
2828 can be managed through the kernel level, it is often used in user-level APIs, such
2829 as MPI, or in the case of InfiniBand, ibVerbs. However, it may be best to select a
2830 interconnect-agnostic middleware for RDMA, such as Photon [239].

2831 RDMA semantics can be used to either read or write contents of remote mem-
2832 ory, in this case VM guest memory pages. However before such operations can take
2833 place, the target side of the operation must register the remote memory buffers and
2834 send the remote key to the initiator, effectively providing the DMA addressing to
2835 be used. Beyond the increased bandwidth and decreased latency benefits provided
2836 by an advanced interconnect with InfiniBand, RDMA also allows VM memory to be
2837 sent without involving the OS. This is due to InfiniBand's zero-copy, kernel bypass
2838 mechanisms, and utilizing asynchronous operations. This keeps the CPU load down,
2839 and focused on the computation at hand rather than the I/O transfer, as it often has
2840 to when utilizing a TCP/IP stack.

2841 Selecting the proper RDMA based communication operations to use can make a
2842 notable difference in the overall performance of the migration. Some RDMA oper-
2843 ations are largely a one-sided movement, which can have performance impact and
2844 should be carefully considered in using for post-copy live migration. The RDMAread
2845 operation requires more effort at the destination host, while RDMA write operation
2846 puts burden on the source host. One way to determine which method is best is by
2847 evaluating both source and sink CPU loads. However this method likely will not
2848 be as obvious when we consider VMs will be not be running in an over-subscribed
2849 virtualized environment, but rather a highly optimized one.

2850 When the destination VM page faults, it is necessary to retrieve the page with
2851 as little latency as possible, as the running thread is paused. Using the method

2852 developed in [236], the *userfaultfd* will trigger an RDMAread to retreive the missing
2853 page. RDMAread requires no interaction from the source VM, as RDMAread is one
2854 sided and the memory buffers have been passed in the setup phase. To further enable
2855 quick return time for the missing page, enabling polling on the source host may further
2856 reduce latency, however verification of this will be necessary. When the RDMAread
2857 operation is finished, the running thread will resume and execution will continue.

2858 **7.2.2 Moving the Compute to the Data**

2859 While pre-copy migration is dominant in the live migration techniques of nearly every
2860 mainstream hypervisor today, it is proposed that for HPC applications, post-copy
2861 migration could provide some key new advances. One particular use case would be to
2862 send a lightweight VM to directly act on a large or set of large datasets, and return a
2863 slimmed down result set. This would reduce the requirement of transmitting the data
2864 across a network entirely, and potentially speed up data access latency drastically, as
2865 on result information in the form of memory pages and VM state are transmitted.

2866 With post-copy migration, one could move the computation at hand close to a
2867 data source in significantly less time than full pre-copy live migration. This data
2868 source, and lightweight VM sink, could potentially be something similar to a Burst
2869 Buffer system [240, 241] or a classic HPC I/O node with a distribute filesystem such
2870 as Lustre or GPFS [242]. This data source could even potentially be a remote sci-
2871 entific instrument completely separate from the HPC infrastructure itself, especially
2872 if the network at hand is capable of RoCE [238] or iWARP [243]. A VM would ini-
2873 tiate post-copy live migration, transmitting only the necessary cpu state, registers,
2874 and non-paged memory rather than the full VM memory state. Once migrated, the
2875 VM could connect to a (now local) I/O or storage device, accessing data fast and
2876 perofrming the necessary calculations. The VM could potentially even forgo the copy

2877 of the majority of its memory. During this time, only the necessary memory pages
2878 required to complete the immediate calculation would generate a fault and trigger
2879 their transmission from the source. The VM could even return the result (rather
2880 than a very large dataset) to the original source VM.

2881 This post-copy live migration technique for remote data computation avoids the
2882 cost of spawning a whole new job and/or process with associated running parameters,
2883 as well as the extremely high cost of a full VM live migration using the pre-copy
2884 method. However, one potential downside of post-copy live migration would be the
2885 non-deterministic runtime, as it would be unknown how much remote memory paging
2886 would be required. This could lead to more time spent with the destination VM in
2887 a paused state awaiting remote memory pages, rather than if the entire VM memory
2888 contents were transmitted completely. Careful analysis of memory usage, or a hybrid
2889 copy method based on predetermined memory sections could help overcome this issue,
2890 but require a more advanced migration architecture.

2891 7.3 Fast VM Cloning

2892 In many distributed system environments, concurrency is achieved through the use
2893 of homogeneous compute nodes that handle the bulk of the computational load in
2894 parallel. This can take many forms, including master/slave configurations, or even a
2895 traditional HPC cluster with identical compute nodes, often used to support Single
2896 Process Multiple Data (SPMD) computational models [244]. With high performance
2897 virtual clusters, there is a need to efficiently deploy and manage near identical virtual
2898 machines for distributed computation.

2899 In Snowflock [245,246], the notion of VM cloning is given. Specifically, Lagar et al.
2900 define the notion of VM Fork, where VMs are treated similar to a fork system call for

2901 processes. This process is conceptually similar to VM migration, with the exception
2902 that the source VM is not destroyed after the migration. Starting with a master
2903 VM, an impromptu cluster can be created across a network using the Xen hypervisor.
2904 Snowflock specifically uses Multicast to linearly scale out VM creation to many hosts,
2905 only coping a minimal state and then remotely coping memory pages when requested.
2906 This fetched memory on-demand is similar in principal to the post-copy live migration
2907 technique described in the previous section. This is further augmented with blocktap-
2908 based virtual disks with Copy-on-Write (CoW) functionally, delivering CoW slices for
2909 each child VM.

2910 While Snowflock provides an excellent framework for VM cloning, it is not suitable
2911 for the current implementation. First, it uses Xen, which in previous research de-
2912 scribed in Chapter 3 has found to have limited performance for HPC workloads [169].
2913 Second, SnowFlock is designed for Ethernet and IP based networks, which have signif-
2914 icantly higher latency and lower bandwidth when compared to InfiniBand solutions.
2915 Developing analogous mechanisms with a high performance hypervisor such as KVM
2916 or Palacios [27] to use RDMA for VM memory paging. RDMA could even be utilized
2917 in conjunction with CoW disc blocks.

2918 Potential VM Cloning RDMA mechanism:

- 2919 ● Prepare parent VM state, including registers and info.
- 2920 ● Prepare CoW disk images.
- 2921 ● Create large buffers for all VM memory on child hosts.
- 2922 ● Send vmstate via RDMAwrite or IB Send to N child nodes, where N is the clone
2923 size.
- 2924 ● Resume/start child VMs in tandem.

- 2925 ● Parent VM set up RDMA multicast (unreliable connection) to send memory
2926 pages to all childs efficiently
- 2927 ● Child clone VM joins RDMA mlticast.
- 2928 ● If child page faults, perform RDMAread operation for specific page.

2929 This method allows for efficient cloning of VMs based on a running parent VM.
2930 First, the VM state and images are copied to all child nodes to recieve the VM,
2931 and blank memory is allocated. Then each child VM is started, and page faults are
2932 handled via RDMA read. This will result in an initial slowdown, but as pages are
2933 received the child VMs will start to run. The rest of the memory is eventually sent
2934 via multicast to all VMs simultaniously. As multicast is unreliable, delivery failure
2935 will just trigger a page fault and subsequent page sending via RDMA. It allows for
2936 direct page fault handling, while allowing child VMs to start and run immediately.
2937 As RDMA multicast mechanisms are relatively questionable, more investigation is
2938 needed to evaluate the feasibility of this situation.

2939 It is expected for post-copy VM cloning to also work most efficiently if used in
2940 conjunction with guest VMs backed with hugepages. Transferring memory in 2M
2941 chunks will more effectively utilize network bandwdith by eliminating send/receive
2942 overhead. This also will hide the latency found in SR-IOV enabled interconnects
2943 for small messages. Furthermore, it will reduce the overhead of page fault handling
2944 mechanisms, as less overall pages will fault and be transferred. While hugepages are
2945 expected to improve VM cloning efficency, emperical testing will still be necessary to
2946 properly evaluate their viability.

2947 While a VM fork mechanism leveraging post-copy live migration in KVM will
2948 quickly spool up cloned VMs, the eventual memory transfer will eventually fail to scale
2949 past the network's capacity. This could happen if hundreds or thousands of child clone

2950 VMs are being started simultaneously, as is likely in large scale deployments. As such,
2951 a higherarchical distribution may be necessary. One possible method for this would be
2952 a two-stage cloning mechanism, where child VMs are cloned one to each cabinet, and
2953 the entire memory contents copied using the pre-copy migration mechanism. From
2954 there, further cloning occurs to deploy many cloned VMs to individual nodes within
2955 the cabinet. Organization of mid-tier cloned VMs would likely be determined based
2956 on RDMA fabric configurations within cabinets, as this is a network-bound process.

2957 7.4 Virtual Cluster Scheduling

2958 Historically, cloud infrastructure has taken a simplistic approach when it comes to VM
2959 and workload scheduling. Often, a simple round-robin or greedy [247] scheduling
2960 algorithm is appropriated. With round robin scheduling, a simple list of host ma-
2961 chines are used and iterated over as VM requests are made. This essentially scatters
2962 the VMs without regard to their locality, and oversubscription becomes commonplace
2963 regardless of the number of requested VMs or their interconnection. A greedy algo-
2964 rithm can help keep spacial locality, but focuses specifically on over-subscription as
2965 well to help consolidate VM allocations. While this oversubscription aspect is very
2966 good for public cloud providers such as Amazon EC2, it becomes counterproductive
2967 for high performance virtual clusters.

2968 With high performance virtual clusters, a number of VM instances are needed
2969 that can gain near-native performance. Furthermore, these VMs will be running
2970 tightly coupled applications, and as such need to be allocated in a way in which
2971 communication latency is minimized and bandwidth is maximized for all VMs. This
2972 will help insure the entire virtual clusters can perform optimally. However, given
2973 off-the-shelf private cloud providers or, worse still, public cloud infrasturcture, these

2974 requirements are at best opaque to the user, and at worst extremely suboptimal.

2975 One way in which high performance virtual clusters can operate efficiently is
 2976 by defining specific instance types, or *flavors* within OpenStack, that define what
 2977 resources a VM has allocated. Given previous research on the NUMA effects of
 2978 VMs [248], these specialized flavors should be defined to fit within a NUMA socket.
 2979 Furthermore, CPU pinning should be used to specifically keep the VM within the
 2980 NUMA socket itself. Using libvirt, a common API utilized in many cloud infrastruc-
 2981 ture deployments (including OpenStack), we can specify CPU pinning directly in the
 2982 XML config.

```
2983 <cputune>
2984   <vcpupin vcpu="0" cpuset="0"/>
2985   <vcpupin vcpu="1" cpuset="1"/>
2986   <vcpupin vcpu="2" cpuset="4"/>
2987   <vcpupin vcpu="3" cpuset="5"/>
2988   <vcpupin vcpu="4" cpuset="6"/>
2989   <emulatorpin cpuset="2"/>
2990 </cputune>
```

<input type="checkbox"/>	Key	Value	Actions
<input type="checkbox"/>	pci_passthrough:labels	["gpu", "infiniband"]	<button>Edit</button> <button>More ▾</button>

Displaying 1 item

Figure 7.2 Adding extra specs to a VM flavor in OpenStack

2991 With OpenStack, this NUMA configuration can be executed through the KVM

2992 Nova plugin and a scheduling filter, which defines how to place VMs effectively. Fur-
2993 themore, the instance flavor can also specify the addition of

2994 `instance_type_extra_specs`

2995 within Nova, whereby specialized hardware such as GPUs and InfiniBand intercon-
2996 nects (as described in Chapters 5 and 6) can be passed through to the VMs directly.

2997 Once defined, the same specialized high performance flavors in OpenStack simply have
2998 to add the labels (such as 'gpu' or 'infiniband') to the flavor to gain the requested
2999 hardware, as illustrated in Figure 7.2 with OpenStack Horizon's UI interface. The
3000 implementation put forth in the OpenStack Havana build has since been upgraded by
3001 Intel with Sr-IOV support and additional scheduling filter additions, and is available
3002 in the latest OpenStack releases [249].

```
3003 pci_passthrough_devices=[{"label":"gpu","address":"0000:08:00.0"},  
3004  {"label":"infiniband","address":"0000:21:00.0"}]  
3005 instance_type_extra_specs={'pci_passthrough:labels': '["gpu"]'}  
3006 instance_type_extra_specs={'pci_passthrough:labels': '["infiniband"]'}
```

3007 To provide a space for high performance virtual clusters, we need a scheduling
3008 mechanism within a cloud infrastructure to support the effective and proper plate-
3009 ment beyond controlling for NUMA characteristics. While there are many effective
3010 scheduling algorithms for workload placement within an environment, a Proximity
3011 Scheduler [37], as defined in OpenStack, may work well. Specifically, one could either
3012 define or compute the underlying cloud infrastructure network topology. This could
3013 be as simple as a YAML file that defines an underlying InfiniBand interconnect 2-1 Fat
3014 Tree topology, or a more complex solution that utilizes network performance metrics
3015 to measure bandwidth and latency between disjoint nodes to build a weighted prox-
3016 mity graph. As it is possible that such network parameters could change due to

3017 other usage or to reconfiguration, a method of periodically monitoring and updating
3018 this proximity network using measurement tools [250] may also help keep an effec-
3019 tive proximity metric between hosts. With a metric, one can then apply a proximity
3020 scheduler to handle high performance virtual cluster allocation requests effectively
3021 and in a way that will be far more optimal than a simple round robin scheduling
3022 mechanism. The OpenStack private cloud IaaS framework is proposed for this ef-
3023 fort, however, further development is needed to bring such a scheidlung mechanism
3024 to fruition.

3025 The use of service level agreements (SLAs) within cloud infrastructure allocation is
3026 a well studied aspect [251]. It is also possible that certain SLAs could be incorporated
3027 into a private cloud infrasturcture such as OpenStack to simultaniously gaurantee
3028 performance for virtual clusters with the above defined methods, while concurrently
3029 offering "classical" VM workload scheduling for HTC, big data, or other cloud usage
3030 models. This would provide the same user experience yet support diverse workloads
3031 and performance expectations as defined by a given SLA

3032 7.5 Chapter Summary

3033 In summary, we expect the combination of transparant huge pages, an RDMA-capable
3034 interconnect multiplexed in hardware for use by both guest and hosts, a high perfor-
3035 mant hypervisor, and post-copy migration and cloning mechanisms to enable a novel
3036 architecture for high perofrmance virtual clusters. These mechanisms, if implemented
3037 and combined with an efficient OS, such as Kitten [252], could help enable a new run-
3038 time system to support extreme scale distirbuted memory computations with in-situ
3039 analysis [253, 254] for large scale scientific systems.

3040 **Chapter 8**

3041 **Conclusion**

3042 With the advent of virtualization and the availability of virtual machines through
3043 cloud infrastructure, a paradigm shift in distributed systems has occurred. Many
3044 services and applications once deployed on workstations, private servers, personal
3045 computers, and even some supercomputers have migrated to a cloud infrastructure.
3046 The reasons for this change are vast, however these reasons are not enough to support
3047 all computational challenges within such a virtualized infrastructure.

3048 The use of tightly coupled, distributed memory applications common in High
3049 Performance Computing communities, has seen a number of problems and compli-
3050 cations when deployed in virtualized infrastructure. While the reasons for this can
3051 be vast, many challenges stem from the performance impact and overhead associated
3052 with virtualization, along with a lack of hardware necessary to support such concur-
3053 rent environments. This dissertation looks to evaluate virtualization for supporting
3054 mid-tier scientific HPC applications and provide potential solutions to these issues.

3055 From the beginning, this dissertation proposes the advent of high performance
3056 virtual clusters to support a wide array of scientific computation, including mid-tier
3057 HPC applications, as well as a framework for building such an environment. This

3058 framework aims at identifying virtualization overhead and finding solutions and best
3059 practices with performant hypervisors, providing support for advanced accelerators
3060 and interconnects to enable new class of applications, and evaluating potential meth-
3061 ods using benchmarks and real-world applications.

3062 Chapter 2 studied the related research necessary for defining not only the context
3063 for virtualization and cloud computing, but also virtual clusters, and their history
3064 through supercomputing. Chapter 3 looked to study the applicability of various hy-
3065 pervisors for supporting common HPC workloads through the use of benchmarks from
3066 a single-node aspect. This found challenges and some solutions to these workloads,
3067 and identified missing gaps that exist.

3068 Chapter 4 started the investigation of the utility of GPUs to support mid-tier
3069 scientific applications using the Xen hypervisor. This chapter provided a proof-of-
3070 concept that with proper configuration and utilizing the latest in hardware support,
3071 GPU passthrough was possible and a viable model for supporting CUDA-enabled
3072 applications, a fast-growing application set. Chapter 5 provides an in-depth com-
3073 parison of multiple hypervisors using the SHOC GPU benchmark suite, as well as
3074 GPU-enabled HPC applications. Here we discover our KVM implementation per-
3075 forms at near-native speeds and allows for effective GPU utilization.

3076 Chapter 6 takes the lessons learned with KVM in GPU passthrough and adds in
3077 SR-IOV InfiniBand support, a critical tool for supporting tightly coupled distributed
3078 memory applications, to build a small virtual cluster. This environment supports
3079 two class-leading Molecular Dynamics simulations, LAMMPS and HOOMD-blue, and
3080 shows how both applications can not only perform at near-native speeds, but also
3081 leverage the latest HPC technologies such as GPUDirect for efficient GPU-to-GPU
3082 communication. This framework is also enveloped in an OpenStack environment.

3083 Chapter 7 is an introspective look at other advancements that can be made in

3084 virtualization to support high performance virtual clusters. Specifically, this chap-
3085 ter details the utility of virtual clusters backed with hugepages, added support for
3086 specialized live migration techniques leveraging high speed RDMA-capable intercon-
3087 nects, VM cloning for fast deployment of virtual clusters themselves, and scheduling
3088 considerations for integration of high performance virtual clusters in OpenStack.

3089 *TODO: Answer the research question, can we support HPC with virtual clusters?*
3090 *How could this help with big data convergence?*

3091 8.1 Impact

3092 TBD

3093 Bibliography

- 3094 [1] B. Alexander, “Web 2.0: A New Wave of Innovation for Teaching and Learn-
3095 ing?” *Learning*, vol. 41, no. 2, pp. 32–44, 2006.
- 3096 [2] R. Buyya, C. S. Yeo, and S. Venugopal, “Market-oriented cloud computing:
3097 Vision, hype, and reality for delivering it services as computing utilities,” in
3098 *Proceedings of the 10th IEEE International Conference on High Performance*
3099 *Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos,*
3100 *CA, USA)*, 2008, pp. 5–13.
- 3101 [3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Comput-
3102 ing 360-Degree Compared,” in *Grid Computing Environments Workshop, 2008.*
3103 *GCE’08*, 2008, pp. 1–10.
- 3104 [4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski,
3105 G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above
3106 the clouds: A berkeley view of cloud computing,” University of
3107 California at Berkeley, Tech. Rep., February 2009. [Online]. Available:
3108 <http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html>
- 3109 [5] T. Kuhn, *The structure of scientific revolutions*. University of Chicago press
3110 Chicago, 1970.

- 3111 [6] T. Sterling, *Beowulf cluster computing with Linux*. The MIT Press, 2001.
- 3112 [7] T. Hoare and R. Milner, “Grand challenges for computing research,” *The Computer Journal*, vol. 48, no. 1, pp. 49–52, 2005.
- 3113
- 3114 [8] J. Dongarra, H. Meuer, and E. Strohmaier, “Top 500 supercomputers,” website, November 2013.
- 3115
- 3116 [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.
- 3117
- 3118
- 3119 [10] L.-W. Wang, “A survey of codes and algorithms used in nersc material science allocations,” *Lawrence Berkeley National Laboratory*, 2006.
- 3120
- 3121 [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.
- 3122
- 3123
- 3124
- 3125 [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].
- 3126
- 3127
- 3128
- 3129 Available: <http://doi.acm.org/10.1145/1851476.1851535>
- 3130 [13] 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.
- 3131
- 3132

- 3133 [14] K. Antypas, “Nersc-6 workload analysis and benchmark selection process,”
3134 *Lawrence Berkeley National Laboratory*, 2008.
- 3135 [15] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally,
3136 J. Meredith, J. Rogers, P. Roth, K. Spafford *et al.*, “Keeneland: Bringing het-
3137 erogeneous gpu computing to the computational science community,” *Comput-
3138 ing in Science and Engineering*, vol. 13, no. 5, pp. 90–95, 2011.
- 3139 [16] P. Pacheco, *Parallel Programming with MPI*, ser. ISBN. Morgan Kaufmann,
3140 October 1996, no. 978-1-55860-339-4.
- 3141 [17] D. Agrawal, S. Das, and A. El Abbadi, “Big data and cloud computing: cur-
3142 rent state and future opportunities,” in *Proceedings of the 14th International
3143 Conference on Extending Database Technology*. ACM, 2011, pp. 530–533.
- 3144 [18] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large
3145 clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- 3146 [19] S. Kamburugamuve, G. Fox, D. Leake, and J. Qiu, “Survey of apache big data
3147 stack,” Ph.D. dissertation, Ph. D. Qualifying Exam, Dept. Inf. Comput., Indi-
3148 ana Univ., Bloomington, IN, 2013.
- 3149 [20] M. Chen, S. Mao, and Y. Liu, “Big data: a survey,” *Mobile Networks and
3150 Applications*, vol. 19, no. 2, pp. 171–209, 2014.
- 3151 [21] D. A. Reed and J. Dongarra, “Exascale computing and big data,” *Communi-
3152 cations of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.
- 3153 [22] S. Jha, J. Qiu, A. Luckow, P. K. Mantha, and G. C. Fox, “A tale of two
3154 data-intensive paradigms: Applications, abstractions, and architectures,” in
3155 *Proceedings of the 3rd International Congress on Big Data*, 2014.

- 3156 [23] J. Qiu, S. Jha, A. Luckow, and G. C. Fox, “Towards hpc-abds: An initial high-
3157 performance big data stack,” *Building Robust Big Data Ecosystem ISO/IEC
3158 JTC 1 Study Group on Big Data*, pp. 18–21, 2014.
- 3159 [24] M. W. ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and
3160 D. K. D. Panda, “High-performance rdma-based design of hadoop mapreduce
3161 over infiniband,” in *Parallel and Distributed Processing Symposium Workshops
3162 PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1908–
3163 1917.
- 3164 [25] S. Ekanayake, S. Kamburugamuve, and G. Fox, “Spidal: High performance data
3165 analytics with java and mpi on large multicore hpc clusters,” in *Proceedings of
3166 24th High Performance Computing Symposium (HPC 2016)*, 2016.
- 3167 [26] F. Tian and K. Chen, “Towards optimal resource provisioning for running
3168 mapreduce programs in public clouds,” in *Cloud Computing (CLOUD), 2011
3169 IEEE International Conference on*. IEEE, 2011, pp. 155–162.
- 3170 [27] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke,
3171 S. Jaconette, M. Levenhagen *et al.*, “Palacios and kitten: New high performance
3172 operating systems for scalable virtualized and native supercomputing,” in *Par-
3173 allel & Distributed Processing (IPDPS), 2010 IEEE International Symposium
3174 on*. IEEE, 2010, pp. 1–12.
- 3175 [28] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang,
3176 “Virtual clusters for grid communities,” *Cluster Computing and the Grid, IEEE
3177 International Symposium on*, vol. 0, pp. 513–520, 2006.
- 3178 [29] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and
3179 S. Tuecke, “A resource management architecture for metacomputing systems,”

- 3180 in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer,
3181 1998, pp. 62–82.
- 3182 [30] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, “Cost-
3183 benefit analysis of cloud computing versus desktop grids,” in *Parallel & Dis-*
3184 *tributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*.
3185 IEEE, 2009, pp. 1–12.
- 3186 [31] T. Bell, B. Bompastor, S. Bukowiec, J. C. Leon, M. Denis, J. van Eldik, M. F.
3187 Lobo, L. F. Alvarez, D. F. Rodriguez, A. Marino *et al.*, “Scaling the cern
3188 openstack cloud,” in *Journal of Physics: Conference Series*, vol. 664, no. 2.
3189 IOP Publishing, 2015, p. 022003.
- 3190 [32] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, “Mapreduce in the clouds for
3191 science,” in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE*
3192 *Second International Conference on*. IEEE, 2010, pp. 565–572.
- 3193 [33] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema,
3194 “A performance analysis of ec2 cloud computing services for scientific comput-
3195 ing,” in *International Conference on Cloud Computing*. Springer, 2009, pp.
3196 115–131.
- 3197 [34] J. Dongarra *et al.*, “The international exascale software project roadmap,”
3198 *International Journal of High Performance Computing Applications*, p.
3199 1094342010391989, 2011.
- 3200 [35] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau,
3201 P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. K. and Dean Klein,
3202 R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S.
3203 Williams, and K. Yelick, “Exascale computing study: Technology challenges

- 3204 in achieving exascale systems,” *Defense Advanced Research Projects Agency*
3205 *Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15,
3206 2008.
- 3207 [36] J. Shalf, S. Dosanjh, and J. Morrison, “Exascale computing technology chal-
3208 lenges,” in *International Conference on High Performance Computing for Com-*
3209 *putational Science*. Springer, 2010, pp. 1–25.
- 3210 [37] J. Suh, “Proximity scheduler in openstack,” Webpage. [Online]. Available:
3211 <https://wiki.openstack.org/wiki/ProximityScheduler>
- 3212 [38] S. Kamburugamuve, S. Ekanayake, M. Pathirage, and G. Fox, “Towards high
3213 performance processing of streaming data in large data centers,” in *HPBDC*
3214 *2016 IEEE International Workshop on High-Performance Big Data Comput-*
3215 *ing in conjunction with The 30th IEEE International Parallel and Distributed*
3216 *Processing Symposium (IPDPS 2016), Chicago, Illinois USA, Friday*, 2016.
- 3217 [39] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, and
3218 G. Pike, “Design of the FutureGrid Experiment Management Framework,”
3219 in *Proceedings of Gateway Computing Environments 2010 at Supercomputing*
3220 *2010*. New Orleans, LA: IEEE, Nov 2010. [Online]. Available: <http://grids.ucs.indiana.edu/ptliupages/publications/vonLaszewski-10-FG-exp-GCE10.pdf>
- 3221 [40] S. Drake and O. development team, “Heat: OpenStack Orchestration,”
3222 Webpage. [Online]. Available: <https://wiki.openstack.org/wiki/Heat>
- 3223 [41] G. Von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, “Accessing
3224 multiple clouds with cloudmesh,” in *Proceedings of the 2014 ACM international*
3225 *workshop on Software-defined ecosystems*. ACM, 2014, pp. 21–28.
- 3226

- 3227 [42] “The magellan project,” Webpage. [Online]. Available: <http://magellan.alcf.anl.gov/>
- 3228
- 3229 [43] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon, “The Magellan Report on
- 3230 Cloud Computing for Science,” U.S. Department of Energy Office of Science
- 3231 Office of Advanced Scientific Computing Research (ASCR), Tech. Rep., Dec.
- 3232 2011.
- 3233 [44] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf,
- 3234 H. Wasserman, and N. Wright, “Performance Analysis of High Performance
- 3235 Computing Applications on the Amazon Web Services Cloud,” in *2nd IEEE International Conference on Cloud Computing Technology and Science*. IEEE,
- 3236 2010, pp. 159–168.
- 3237
- 3238 [45] D. Merkel, “Docker: lightweight linux containers for consistent development
- 3239 and deployment,” *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.
- 3240 [46] D. M. Jacobsen and R. S. Canon, “Contain this, unleashing docker for hpc,”
- 3241 *Proceedings of the Cray User Group*, 2015.
- 3242 [47] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A.
- 3243 De Rose, “Performance evaluation of container-based virtualization for high
- 3244 performance computing environments,” in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE,
- 3245 2013, pp. 233–240.
- 3246
- 3247 [48] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann, 2013.
- 3248
- 3249 [49] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neuge-
- 3250 bauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in *Pro-*

- 3251 *ceedings of the 19th ACM Symposium on Operating Systems Principles*, New
3252 York, U. S. A., Oct. 2003, pp. 164–177.
- 3253 [50] “Vmware esx server,” [Online], <http://www.vmware.com/de/products/vi/esx/>.
- 3254 [51] J. Duato, A. J. Pena, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-
3255 Orti, “Enabling CUDA acceleration within virtual machines using rCUDA,” in
3256 *High Performance Computing (HiPC), 2011 18th International Conference on*.
3257 IEEE, 2011, pp. 1–10.
- 3258 [52] L. Shi, H. Chen, J. Sun, and K. Li, “vCUDA: GPU-accelerated high-
3259 performance computing in virtual machines,” *Computers, IEEE Transactions*
3260 on, vol. 61, no. 6, pp. 804–816, 2012.
- 3261 [53] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux
3262 virtual machine monitor,” in *Proceedings of the Linux Symposium*, vol. 1, 2007,
3263 pp. 225–230.
- 3264 [54] I. Foster, C. Kesselman, and S. Tuecke, “The Anatomy of the Grid: Enabling
3265 Scalable Virtual Organizations,” *Intl. J. Supercomputer Applications*, vol. 15,
3266 no. 3, 2001.
- 3267 [55] I. Foster, C. Kesselman *et al.*, “The Physiology of the Grid:An Open Grid
3268 Services Architecture for Distributed Systems Integration,” Argonne National
3269 Laboratory, Chicago, Tech. Rep., Jan. 2002.
- 3270 [56] D. DiNucci, “Fragmented future,” *AllBusiness-Champions of Small Business*,
3271 1999. [Online]. Available: <http://www.cdinucci.com/Darcy2/articles/Print/Printarticle7.html>

- 3273 [57] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard,
3274 S. Poglani, K. Riemer, S. Struble, P. Takacs-Nagy, I. Trickovic, and S. Zimek,
3275 “Web Services Choreography Interface,” Jun. 2002, version 1.0. [Online].
3276 Available: <http://wwws.sun.com/software/xml/developers/wsci/index.html>
- 3277 [58] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Archi-*
3278 *ture Best Practices (The Coad Series)*. Prentice Hall PTR Upper Saddle
3279 River, NJ, USA, 2004.
- 3280 [59] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee,
3281 D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Com-*
3282 *munications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- 3283 [60] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, and J. Tao,
3284 “Cloud Computing: a Perspective Study,” *New Generation Computing*, vol. 28,
3285 pp. 63–69, Mar 2010. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/10-cloudcomputing-NGC/vonLaszewski-10-NGC.pdf>
- 3286 [61] G. von Laszewski, A. J. Younge, X. He, K. Mahinthakumar, and
3287 L. Wang, “Experiment and Workflow Management Using Cyberaide
3288 Shell,” in *Proceedings of the 4th International Workshop on Workflow*
3289 *Systems in e-Science (WSES 09) with 9th IEEE/ACM International*
3290 *Symposium on Cluster Computing and the Grid (CCGrid 09)*. IEEE,
3291 May 2009. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/09-gridshell-ccgrid/vonLaszewski-ccgrid09-final.pdf>
- 3292 [62] G. von Laszewski, F. Wang, A. J. Younge, X. He, Z. Guo, and M. Pierce,
3293 “Cyberaide JavaScript: A JavaScript Commodity Grid Kit,” in *Proceedings*
3294 *of the Grid Computing Environments 2007 at Supercomputing 2008*. Austin,

- 3297 TX: IEEE, Nov 2008. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/08-javascript/vonLaszewski-08-javascript.pdf>
- 3298
- 3299 [63] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, “Comparison of multiple
3300 cloud frameworks,” in *Cloud Computing (CLOUD), 2012 IEEE 5th Interna-*
3301 *tional Conference on*, June 2012, pp. 734–741.
- 3302 [64] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing
3303 systems,” *INC, IMS and IDC*, pp. 44–51, 2009.
- 3304 [65] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Virtual infrastruc-
3305 ture management in private and hybrid clouds,” *IEEE Internet Computing*,
3306 vol. 13, no. 5, pp. 14–22, 2009.
- 3307 [66] S. A. Baset, “Cloud slas: present and future,” *ACM SIGOPS Operating Systems
3308 Review*, vol. 46, no. 2, pp. 57–66, 2012.
- 3309 [67] “Amazon Elastic Compute Cloud.” [Online]. Available: <http://aws.amazon.com/ec2/>
- 3310
- 3311 [68] S. Krishnan and J. L. U. Gonzalez, “Google compute engine,” in *Building Your
3312 Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 53–81.
- 3313 [69] “Nimbus Project,” <http://www.nimbusproject.org>. Last access Mar. 2011.
- 3314 [70] K. Keahey, I. Foster, T. Freeman, and X. Zhang, “Virtual workspaces: Achiev-
3315 ing quality of service and quality of life in the grid,” *Scientific Programming
3316 Journal*, vol. 13, no. 4, pp. 265–276, 2005.
- 3317 [71] “Amazon web services s3 rest api,” <http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf>. Last Access Mar. 2011.
- 3318

- 3319 [72] “Jets3t project,” <http://bitbucket.org/jmurty/jets3t/wiki/Home>. Last Access
3320 Mar. 2011.
- 3321 [73] “Boto project,” <http://code.google.com/p/boto>. Last Access Mar. 2011.
- 3322 [74] “S3tools project,” <http://s3tools.org/s3cmd>. Last Access Mar. 2011.
- 3323 [75] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and
3324 D. Zagorodnov, “The Eucalyptus Open-source Cloud-computing System,” *Pro-*
3325 *ceedings of Cloud Computing and Its Applications*, 2008.
- 3326 [76] “Eucalyptus open-source cloud computing infrastructure,” an Overview, Euca-
3327 lyptus Systems, Inc. 2009.
- 3328 [77] “Eucalyptus,” <http://www.eucalyptus.com>, Last Access Mar. 2011.
- 3329 [78] I. H. Shaon, “Eucalyptus and its components,” Webpage.
3330 [Online]. Available: <https://mdshaonimran.wordpress.com/2011/11/26/eucalyptus-and-its-components>
- 3331
- 3332 [79] “Openstack. cloud software,” <http://openstack.org>, Last Access Mar. 2011.
- 3333 [80] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “Openstack: toward an open-source
3334 solution for cloud computing,” *International Journal of Computer Applications*,
3335 vol. 55, no. 3, 2012.
- 3336 [81] “Opennebula project,” <http://www.opennebula.org>. Last access Mar. 2011.
- 3337 [82] I. M. Llorente, R. Moreno-Vozmediano, and R. S. Montero, “Cloud computing
3338 for on-demand grid resource provisioning,” *Advances in Parallel Computing*,
3339 vol. 18, no. 5, pp. 177–191, 2009.

- 3340 [83] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Capacity leasing in cloud systems using the opennebula engine,” *Cloud Computing and its*
3341 *Applications*, 2008.
- 3342
- 3343 [84] “Libvirt API webpage,” [http:// libvirt.org](http://libvirt.org). Last access Mar. 2011.
- 3344 [85] “Elastichosts webpage,” <http://www.elastichosts.com>. Last Access Mar. 2011.
- 3345 [86] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Mat-
3346 sunaga, M. Tsugawa, J. Zhang, M. Zhao *et al.*, “From virtualized resources
3347 to virtual computing grids: the in-vigo system,” *Future Generation Computer
3348 Systems*, vol. 21, no. 6, pp. 896–909, 2005.
- 3349 [87] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle, “Dynamic virtual clus-
3350 ters in a grid site manager,” in *12th IEEE International Symposium on High
3351 Performance Distributed Computing, 2003. Proceedings*, 2003, pp. 90–100.
- 3352 [88] I. VMware, “VMware vCloud Air,” Webpage. [Online]. Available: <http://vcloud.vmware.com>
- 3353
- 3354 [89] CERN, “LHC Computing Grid Project,” Web Page, Dec. 2003. [Online].
3355 Available: <http://lcg.web.cern.ch/LCG/>
- 3356 [90] J. Luo, A. Song, Y. Zhu, X. Wang, T. Ma, Z. Wu, Y. Xu, and L. Ge, “Grid sup-
3357 porting platform for AMS data processing,” *Lecture notes in computer science*,
3358 vol. 3759, p. 276, 2005.
- 3359 [91] “CMS,” Web Page. [Online]. Available: <http://cms.cern.ch/>
- 3360 [92] J. Diaz, A. J. Younge, G. von Laszewski, F. Wang, and G. C. Fox, “Grappling
3361 Cloud Infrastructure Services with a Generic Image Repository,” in *Proceedings
3362 of Cloud Computing and Its Applications (CCA 2011)*, Argonne, IL, Mar 2011.

- 3363 [93] I. Foster, "The anatomy of the grid: Enabling scalable virtual organizations,"
3364 in *Euro-Par 2001 Parallel Processing: 7th International Euro-Par Conference*.
3365 Springer, 2001, pp. 1–5. [Online]. Available: www.globus.org/alliance/publications/papers/anatomy.pdf
- 3367 [94] K. Keahey and T. Freeman, "Contextualization: Providing one-click virtual
3368 clusters," in *eScience, 2008. eScience'08. IEEE Fourth International Conference on*. IEEE, 2008, pp. 301–308.
- 3370 [95] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos,
3371 W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni, R. P. Wagner, N. Wilkins-
3372 Diehr, and M. L. Norman, "Gateways to discovery: Cyberinfrastructure
3373 for the long tail of science," in *Proceedings of the 2014 Annual Conference on Extreme
3374 Science and Engineering Discovery Environment*, ser. XSEDE '14. New York, NY, USA: ACM, 2014, pp. 39:1–39:8. [Online]. Available:
3375 <http://doi.acm.org/10.1145/2616498.2616540>
- 3377 [96] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE
3378 Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
- 3379 [97] F. Berman, "From TeraGrid to knowledge grid," *Communications of the ACM*,
3380 vol. 44, no. 11, pp. 27–28, 2001.
- 3381 [98] C. Catlett, "The philosophy of TeraGrid: building an open, extensible, dis-
3382 tributed TeraScale facility," in *2nd IEEE/ACM International Symposium on Cluster
3383 Computing and the Grid, 2002*, 2002, pp. 8–8.
- 3384 [99] H. H. Goldstine and A. Goldstine, "The electronic numerical integrator and
3385 computer (eniac)," *Mathematical Tables and Other Aids to Computation*, vol. 2,
3386 no. 15, pp. 97–110, 1946.

- 3387 [100] J. E. Thornton, “Design of a computer - the control data 6600,” 1970.
- 3388 [101] R. M. Russell, “The cray-1 computer system,” *Communications of the ACM*,
3389 vol. 21, no. 1, pp. 63–72, 1978.
- 3390 [102] C. L. Seitz, “The cosmic cube,” *Communications of the ACM*, vol. 28, no. 1,
3391 pp. 22–33, 1985.
- 3392 [103] G. C. Fox, S. W. Otto, and A. J. Hey, “Matrix algorithms on a hypercube i:
3393 Matrix multiplication,” *Parallel computing*, vol. 4, no. 1, pp. 17–31, 1987.
- 3394 [104] W. D. Hillis, *The connection machine*. MIT press, 1989.
- 3395 [105] X. Zhang, C. Yang, F. Liu, Y. Liu, and Y. Lu, “Optimizing and scaling hpcg
3396 on tianhe-2: early experience,” in *International Conference on Algorithms and*
3397 *Architectures for Parallel Processing*. Springer, 2014, pp. 28–41.
- 3398 [106] A. Geist, *PVM: Parallel virtual machine: a users’ guide and tutorial for net-*
3399 *worked parallel computing*. MIT press, 1994.
- 3400 [107] B. Carpenter, V. Getov, G. Judd, A. Skjellum, and G. C. Fox, “Mpj: Mpi-like
3401 message passing for java,” 2000.
- 3402 [108] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres,
3403 V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, “Open mpi: Goals,
3404 concept, and design of a next generation mpi implementation,” in *European*
3405 *Parallel Virtual Machine/Message Passing Interface Users Group Meeting*.
3406 Springer, 2004, pp. 97–104.
- 3407 [109] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming*
3408 *with the message-passing interface*. MIT press, 1999, vol. 1.

- 3409 [110] M. J. Koop, T. Jones, and D. K. Panda, “Mvapich-aptus: Scalable high-
3410 performance multi-transport mpi over infiniband,” in *IEEE International Sym-
3411 posium on Parallel and Distributed Processing, 2008. IPDPS 2008.* IEEE,
3412 2008, pp. 1–12.
- 3413 [111] R. Rabenseifner, G. Hager, and G. Jost, “Hybrid mpi/openmp parallel program-
3414 ming on clusters of multi-core smp nodes,” in *2009 17th Euromicro international
3415 conference on parallel, distributed and network-based processing.* IEEE, 2009,
3416 pp. 427–436.
- 3417 [112] D. A. Jacobsen, J. C. Thibault, and I. Senocak, “An mpi-cuda implementation
3418 for massively parallel incompressible flow computations on multi-gpu clusters,”
3419 in *48th AIAA aerospace sciences meeting and exhibit*, vol. 16, 2010, p. 2.
- 3420 [113] A. S. Bland, J. Wells, O. E. Messer, O. Hernandez, and J. Rogers, “Titan:
3421 Early experience with the cray xk6 at oak ridge national laboratory,” *Cray
3422 User Group*, 2012.
- 3423 [114] J. J. Dongarra, P. Luszczek, and A. Petitet, “The linpack benchmark: past,
3424 present and future,” *Concurrency and Computation: practice and experience*,
3425 vol. 15, no. 9, pp. 803–820, 2003.
- 3426 [115] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, “Introducing the
3427 graph 500,” *Cray Users Group (CUG)*, 2010.
- 3428 [116] M. A. Heroux and J. Dongarra, “Toward a new metric for ranking high perfor-
3429 mance computing systems,” *Sandia Report, SAND2013-4744*, vol. 312, 2013.
- 3430 [117] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. E. Bernholdt, “Hobbes:
3431 Composition and virtualization as the foundations of an extreme-scale os/r,”

- 3432 in *Proceedings of the 3rd International Workshop on Runtime and Operating*
3433 *Systems for Supercomputers.* ACM, 2013, p. 2.
- 3434 [118] S. Perarnau, R. Gupta, and P. Beckman, “Argo: An exascale operating system
3435 and runtime,” in *Poster Proceedings from IEEE/ACM Supercomputing 2015,*
3436 2015.
- 3437 [119] T. Sterling, D. Kogler, M. Anderson, and M. Brodowicz, “SLOWER: A perfor-
3438 mance model for Exascale computing,” *Supercomputing Frontiers and Innova-*
3439 *tions*, vol. 1, pp. 42–57, Sep 2014.
- 3440 [120] E. Ciurana, *Developing with Google App Engine.* Springer, 2009.
- 3441 [121] D. Chappell, “Introducing windows azure,” Microsoft, Inc, Tech. Rep., 2009.
- 3442 [122] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, “Science
3443 clouds: Early experiences in cloud computing for scientific applications,” *Cloud*
3444 *Computing and Applications*, vol. 2008, 2008.
- 3445 [123] R. Creasy, “The origin of the VM/370 time-sharing system,” *IBM Journal of*
3446 *Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.
- 3447 [124] “Amazon elastic compute cloud,” [Online], <http://aws.amazon.com/ec2/>.
- 3448 [125] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, “Virtual
3449 Workspaces in the Grid,” *Lecture Notes in Computer Science*, vol. 3648,
3450 pp. 421–431, 2005. [Online]. Available: http://workspace.globus.org/papers/VW_EuroPar05.pdf
- 3452 [126] J. Fontan, T. Vazquez, L. Gonzalez, R. S. Montero, and I. M. Llorente, “Open-
3453 NEbula: The Open Source Virtual Machine Manager for Cluster Computing,”

- 3454 in *Open Source Grid and Cluster Software Conference*, San Francisco, CA, USA,
3455 May 2008.
- 3456 [127] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Mat-
3457 sunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, “From virtualized
3458 resources to virtual computing Grids: the In-VIGO system,” *Future Generation
3459 Comp. Syst.*, vol. 21, no. 6, pp. 896–909, 2005.
- 3460 [128] Rackspace, “Openstack,” WebPage, Jan 2011. [Online]. Available: <http://www.openstack.org/>
- 3461
- 3462 [129] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, “Performance evaluation
3463 of virtualization technologies for server consolidation,” HP Laboratories, Tech.
3464 Rep., 2007.
- 3465 [130] J. Watson, “Virtualbox: bits and bytes masquerading as machines,” *Linux
3466 Journal*, vol. 2008, no. 166, p. 1, 2008.
- 3467 [131] D. Leinenbach and T. Santen, “Verifying the Microsoft Hyper-V Hypervisor
3468 with VCC,” *FM 2009: Formal Methods*, pp. 806–809, 2009.
- 3469 [132] I. Parallels, “An introduction to os virtualization and paral-
3470 lels virtuozzo containers,” Parallels, Inc, Tech. Rep., 2010. [On-
3471 line]. Available: http://www.parallels.com/r/pdf/wp/pvc/Parallels_Virtuozzo_Containers_WP_an_introduction_to_os_EN.pdf
- 3472
- 3473 [133] D. Bartholomew, “Qemu: a multihost, multitarget emulator,” *Linux Journal*,
3474 vol. 2006, no. 145, p. 3, 2006.
- 3475 [134] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamil-
3476 ton, M. McCabe, and J. Owens, “Quantifying the performance isolation prop-

- 3477 erties of virtualization systems,” in *Proceedings of the 2007 workshop on Ex-*
- 3478 perimental computer science, ser. ExpCS ’07. New York, NY, USA: ACM,
- 3479 2007.
- 3480 [135] Oracle, “Performance evaluation of oracle vm server virtualization software,”
- 3481 Oracle, Whitepaper, 2008. [Online]. Available: <http://www.oracle.com/us/technologies/virtualization/oraclevm/026997.pdf>
- 3483 [136] K. Adams and O. Agesen, “A comparison of software and hardware techniques
- 3484 for x86 virtualization,” in *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*. ACM,
- 3485 2006, pp. 2–13, vMware.
- 3487 [137] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, “An analysis
- 3488 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.
- 3491 [138] S. Rixner, “Network virtualization: Breaking the performance barrier,” *Queue*,
- 3492 vol. 6, no. 1, p. 36, 2008.
- 3493 [139] S. Nanda and T. Chiueh, “A survey of virtualization technologies,” Tech. Rep.,
- 3494 2005.
- 3495 [140] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, “Performance im-
- 3496 plications of virtualizing multicore cluster machines,” in *Proceedings of the 2nd workshop on System-level virtualization for high performance computing*. ACM,
- 3497 2008, pp. 1–8.
- 3499 [141] R. Harper and K. Rister, “Kvm limits arbitrary or architectural?” IBM Linux
- 3500 Technology Center, Jun 2009.

- 3501 [142] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, and A. Brinkmann, “Non-
3502 intrusive virtualization management using libvirt,” in *Design, Automation Test
3503 in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 574 –579.
- 3504 [143] “FutureGrid,” Web Page, 2009. [Online]. Available: <http://www.futuregrid.org>
- 3505 [144] J. J. Dujmovic and I. Dujmovic, “Evolution and evaluation of spec bench-
3506 marks,” *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 2–9, 1998.
- 3507 [145] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner,
3508 and D. Takahashi, “The HPC Challenge (HPCC) benchmark suite,” in *SC06
3509 Conference Tutorial*. Citeseer, 2006.
- 3510 [146] J. Dongarra and P. Luszczek, “Reducing the time to tune parallel dense linear
3511 algebra routines with partial execution and performance modelling,” University
3512 of Tennessee Computer Science Technical Report, Tech. Rep., 2010.
- 3513 [147] K. Dixit, “The SPEC benchmarks,” *Parallel Computing*, vol. 17, no. 10-11, pp.
3514 1195–1209, 1991.
- 3515 [148] SPEC, “Standard performance evaluation corporation,” Webpage, Jan 2011.
3516 [Online]. Available: <http://www.spec.org/>
- 3517 [149] R. Henschel and A. J. Younge, “First quarter 2011 spec omp results,” Webpage,
3518 Mar 2011. [Online]. Available: <http://www.spec.org/omp/results/res2011q1/>
- 3519 [150] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. Prentice Hall, 2002,
3520 vol. 2.
- 3521 [151] Amazon, “Elastic Compute Cloud.” [Online]. Available: <http://aws.amazon.com/ec2/>

- 3523 [152] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large
3524 clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- 3525 [153] "Windows azure platform," [Online], <http://www.microsoft.com/azure/>.
- 3526 [154] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, G. G.
3527 Pike, W. Smith, J. Vockler, R. J. Figueiredo, J. Fortes *et al.*, "Design of the
3528 futuregrid experiment management framework," in *Gateway Computing Envi-
3529 ronments Workshop (GCE), 2010.* IEEE, 2010, pp. 1–10.
- 3530 [155] OpenStack, "Openstack compute administration manual," 2013.
- 3531 [156] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and
3532 D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in
3533 *Proceedings of Cloud Computing and Its Applications*, October 2008. [Online].
3534 Available: <http://eucalyptus.cs.ucsb.edu/wiki/Presentations>
- 3535 [157] C. Nvidia, "Programming guide," 2008.
- 3536 [158] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard
3537 for heterogeneous computing systems," *Computing in science & engineering*,
3538 vol. 12, no. 3, p. 66, 2010.
- 3539 [159] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron,
3540 "A performance study of general-purpose applications on graphics processors
3541 using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68,
3542 no. 10, pp. 1370 – 1380, 2008, k-means implementation on CUDA
3543 with 72x speedup. Compares to 4-threaded CPU version with 30x
3544 speedup. [Online]. Available: <http://www.sciencedirect.com/science/article/B6WKJ-4SVV8GS-2/2/f7a1dccceb63cbbfd25774c6628d8412>

- 3546 [160] Z. Liu and W. Ma, "Exploiting computing power on graphics processing unit,"
3547 vol. 2, Dec. 2008, pp. 1062–1065.
- 3548 [161] S. Hong and H. Kim, "An integrated GPU power and performance model,"
3549 in *ACM SIGARCH Computer Architecture News*, vol. 38. ACM, 2010, pp.
3550 280–289.
- 3551 [162] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Den-
3552 neau, P. Franzon, W. Harrod, K. Hill *et al.*, "Exascale computing study: Tech-
3553 nology challenges in achieving exascale systems," 2008.
- 3554 [163] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Mod-
3555 ium, K. Singh, J. Suh, and J. P. Walters, "Heterogeneous cloud computing," in
3556 *Proceedings of the Workshop on Parallel Programming on Accelerator Clusters*
3557 (*PPAC*). *Cluster Computing (CLUSTER), 2011 IEEE International Confer-*
3558 *ence on*. IEEE, 2011, pp. 378–385.
- 3559 [164] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-*
3560 *purpose GPU programming*. Addison-Wesley Professional, 2010.
- 3561 [165] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E.
3562 Stone, J. C. Phillips, and W.-m. Hwu, "GPU clusters for high-performance
3563 computing," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE*
3564 *International Conference on*. IEEE, 2009, pp. 1–8.
- 3565 [166] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C.
3566 Sancho, "Entering the petaflop era: the architecture and performance of Road-
3567 runner," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*.
3568 IEEE Press, 2008, p. 1.

- 3569 [167] S. Craven and P. Athanas, "Examining the viability of FPGA supercomputing,"
3570 *EURASIP Journal on Embedded systems*, vol. 2007, no. 1, pp. 13–13, 2007.
- 3571 [168] G. Shainer, T. Liu, J. Layton, and J. Mora, "Scheduling strategies for HPC as a
3572 service (HPCaaS)," in *Cluster Computing and Workshops, 2009. CLUSTER'09.*
3573 *IEEE International Conference on.* IEEE, 2009, pp. 1–6.
- 3574 [169] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C.
3575 Fox, "Analysis of Virtualization Technologies for High Performance Computing
3576 Environments," in *Proceedings of the 4th International Conference on Cloud*
3577 *Computing (CLOUD 2011).* Washington, DC: IEEE, July 2011.
- 3578 [170] W. Wade, "How NVIDIA and Citrix are driving the future of virtualized visual
3579 computing," [Online], <http://blogs.nvidia.com/blog/2013/05/22/synergy/>.
- 3580 [171] S. Long, "Virtual machine graphics acceleration deployment guide," VMWare,
3581 Tech. Rep., 2013.
- 3582 [172] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and
3583 J. T. Klosowski, "Chromium: a stream-processing framework for interactive
3584 rendering on clusters," in *ACM Transactions on Graphics (TOG)*, vol. 21.
3585 ACM, 2002, pp. 693–702.
- 3586 [173] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, "A GPGPU
3587 transparent virtualization component for high performance computing clouds,"
3588 in *Euro-Par 2010 - Parallel Processing*, ser. Lecture Notes in Computer
3589 Science, P. D'Ambra, M. Guarracino, and D. Talia, Eds. Springer
3590 Berlin Heidelberg, 2010, vol. 6271, pp. 379–391. [Online]. Available:
3591 http://dx.doi.org/10.1007/978-3-642-15277-1_37

- 3592 [174] K. Diab, M. Rafique, and M. Hefeeda, “Dynamic sharing of GPUs in cloud
3593 systems,” in *Parallel and Distributed Processing Symposium Workshops PhD*
3594 *Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 947–954.
- 3595 [175] C.-T. Yang, H.-Y. Wang, and Y.-T. Liu, “Using pci pass-through for gpu vir-
3596 tualization with cuda,” in *Network and Parallel Computing*. Springer, 2012,
3597 pp. 445–452.
- 3598 [176] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford,
3599 V. Tippurajju, and J. S. Vetter, “The scalable heterogeneous computing (SHOC)
3600 benchmark suite,” in *Proceedings of the 3rd Workshop on General-Purpose*
3601 *Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.
- 3602 [177] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen, “Direct numerical
3603 simulation of turbulent combustion: fundamental insights towards predictive
3604 models,” in *Journal of Physics: Conference Series*, vol. 16. IOP Publishing,
3605 2005, p. 65.
- 3606 [178] F. Silla, “rCUDA: share and aggregate GPUs in your cluster,” Nov. 2012, mel-
3607 lanox Booth Presaentation.
- 3608 [179] H. Jo, J. Jeong, M. Lee, and D. H. Choi, “Exploiting GPUs in virtual machine
3609 for biocloud,” *BioMed research international*, vol. 2013, 2013.
- 3610 [180] J. Duato, A. Pena, F. Silla, R. Mayo, and E. Quintana-Orti, “rCUDA: Re-
3611 ducing the number of gpu-based accelerators in high performance clusters,”
3612 in *High Performance Computing and Simulation (HPCS), 2010 International*
3613 *Conference on*, June 2010, pp. 224–231.
- 3614 [181] B. L. Jacob and T. N. Mudge, “A look at several memory management units,
3615 TLB-refill mechanisms, and page table organizations,” in *Proceedings of the*

- 3616 *Eighth International Conference on Architectural Support for Programming*
3617 *Languages and Operating Systems.* ACM, 1998, pp. 295–306.
- 3618 [182] B.-A. Yassour, M. Ben-Yehuda, and O. Wasserman, “Direct device assignment
3619 for untrusted fully-virtualized virtual machines,” IBM Research, Tech. Rep.,
3620 2008.
- 3621 [183] M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, and
3622 L. Van Doorn, “The price of safety: Evaluating IOMMU performance,” in
3623 *Ottawa Linux Symposium*, 2007.
- 3624 [184] J. Liu, “Evaluating standard-based self-virtualizing devices: A performance
3625 study on 10 GbE NICs with SR-IOV support,” in *Parallel Distributed Processing*
3626 (*IPDPS*), 2010 IEEE International Symposium on, April 2010, pp. 1–12.
- 3627 [185] V. Jujjuri, E. Van Hensbergen, A. Ligouri, and B. Pulavarty, “VirtFS-a virtu-
3628 alization aware file system passthrough,” in *Ottawa Linux Symposium*, 2010.
- 3629 [186] C. Yang, H. Wang, W. Ou, Y. Liu, and C. Hsu, “On implementation of GPU
3630 virtualization using PCI pass-through,” in *4th IEEE International Conference*
3631 *on Cloud Computing Technology and Science Proceedings (CloudCom)*, 2012.
- 3632 [187] M. Dowty and J. Sugerman, “GPU virtualization on VMware’s hosted I/O
3633 architecture,” *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 73
3634 – 82, 2009.
- 3635 [188] “FutureGrid,” Web page. [Online]. Available: <http://portal.futuregrid.org>
- 3636 [189] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spaf-
3637 ford, V. Tipparaju, and J. S. Vetter, “The Scalable Heterogeneous Computing

- 3638 (SHOC) benchmark suite,” in *Proceedings of the 3rd Workshop on General-*
3639 *Purpose Computation on Graphics Processing Units*, ser. GPGPU ’10. ACM,
3640 2010, pp. 63–74.
- 3641 [190] “LAMMPS molecular dynamics simulator,” <http://lammps.sandia.gov/>, [On-
3642 line; accessed Jan. 2, 2014].
- 3643 [191] A. Athanasopoulos, A. Dimou, V. Mezaris, and I. Kompatsiaris, “GPU ac-
3644 leartion for support vector machines,” in *Proc 12th International Workshop*
3645 *on Image Analysis for Multimedia Interactive Services (WIAMIS 2001)*, April
3646 2011.
- 3647 [192] “LULESH shock hydrodynamics application,” <https://codesign.llnl.gov/lulesh.php>, [Online; accessed Jan. 2, 2014].
- 3649 [193] S. Plimpton, “Fast parallel algorithms for short-range molecular dynamics,”
3650 *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995.
- 3651 [194] W. M. Brown, “GPU acceleration in LAMMPS,” <http://lammps.sandia.gov/workshops/Aug11/Brown/brown11.pdf>, [Online; accessed Jan. 2, 2014].
- 3653 [195] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,”
3654 *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.
- 3655 [196] “Hydrodynamics Challenge Problem,” <https://codesign.llnl.gov/pdfs/spec-7.pdf>, [Online; accessed Jan. 2, 2014].
- 3657 [197] A. Arcangeli, “Transparent hugepage support,” <http://www.linux-kvm.org/wiki/images/9/9e/2010-forum-thp.pdf>, 2010, [Online; accessed Jan. 29, 2014].
- 3659 [198] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, “SR-IOV
3660 support for virtualization on infiniband clusters: Early experience,” in *Cluster*

- 3661 *Computing and the Grid, IEEE International Symposium on.* IEEE Computer
3662 Society, 2013, pp. 385–392.
- 3663 [199] 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
3664 discovery: Cyberinfrastructure for the long tail of science,” in *Proceedings of*
3665 *the 2014 Annual Conference on Extreme Science and Engineering Discovery*
3666 *Environment.* ACM, 2014, p. 39.
- 3667
- 3668 [200] P. Luszczek, E. Meek, S. Moore, D. Terpstra, V. M. Weaver, and J. Dongarra,
3669 “Evaluation of the hpc challenge benchmarks in virtualized environments,” in
3670 *Proceedings of the 2011 International Conference on Parallel Processing - Volume 2*, ser. Euro-Par’11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 436–
3671 445.
- 3672
- 3673 [201] N. Huber, M. von Quast, M. Hauck, and S. Kounev, “Evaluating and model-
3674 ing virtualization performance overhead for cloud environments.” in *CLOSER*,
3675 2011, pp. 563–573.
- 3676
- 3677 [202] J. P. Walters, A. J. Younge, D.-I. Kang, K.-T. Yao, M. Kang, S. P. Crago,
3678 and G. C. Fox, “GPU-Passthrough Performance: A Comparison of KVM, Xen,
3679 VMWare ESXi, and LXC for CUDA and OpenCL Applications,” in *Proceedings*
3680 *of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014).*
Anchorage, AK: IEEE, 2014.
- 3681
- 3682 [203] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, “Sr-iov
3683 support for virtualization on infiniband clusters: Early experience,” in *Cluster,*
3684 *Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International*
Symposium on. IEEE, 2013, pp. 385–392.

- 3685 [204] M. Musleh, V. Pai, J. P. Walters, A. J. Younge, and S. P. Crago, “Bridging
3686 the Virtualization Performance Gap for HPC using SR-IOV for InfiniBand,”
3687 in *Proceedings of the 7th IEEE International Conference on Cloud Computing*
3688 (*CLOUD 2014*). Anchorage, AK: IEEE, 2014.
- 3689 [205] “Aws high performance computing,” <http://aws.amazon.com/hpc/>, last Access
3690 Nov. 2014.
- 3691 [206] “Openstack flavors,” <http://docs.openstack.org/openstack-ops/content/flavors.html>, last Access Nov. 2014.
- 3693 [207] K. Pepple, *Deploying OpenStack*. O’Reilly Media, Inc., 2011.
- 3694 [208] S. Hazelhurst, “Scientific computing using virtual high-performance computing:
3695 a case study using the amazon elastic computing cloud,” in *Proceedings of the
3696 2008 annual research conference of the South African Institute of Computer
3697 Scientists and Information Technologists on IT research in developing countries:
3698 riding the wave of technology*. ACM, 2008, pp. 94–103.
- 3699 [209] E. Amazon, “Amazon elastic compute cloud (amazon ec2),” *Amazon Elastic
3700 Compute Cloud (Amazon EC2)*, 2010.
- 3701 [210] R. Jennings, *Cloud Computing with the Windows Azure Platform*. John Wiley
3702 & Sons, 2010.
- 3703 [211] “Google cloud platform,” <https://cloud.google.com/>, last Access Nov. 2014.
- 3704 [212] L. Ramakrishnan, R. S. Canon, K. Muriki, I. Sakrejda, and N. J. Wright,
3705 “Evaluating interconnect and virtualization performance forhigh performance
3706 computing,” *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 2, pp. 55–60,
3707 Oct. 2012. [Online]. Available: <http://doi.acm.org/10.1145/2381056.2381071>

- 3708 [213] M. Righini, "Enabling intel virtualization technology features and benefits,"
3709 Intel Corporation, Tech. Rep., 2010.
- 3710 [214] AMD, "AMD i/o virtualization technology (IOMMU) specification," AMD Cor-
3711 poration, Tech. Rep., 2009.
- 3712 [215] A. Limited, "Arm system memory management unit architecture specification,"
3713 ARM Limited, Tech. Rep., 2013.
- 3714 [216] A. J. Younge, J. P. Walters, S. Crago, and G. C. Fox, "Evaluating GPU
3715 Passthrough in Xen for High Performance Cloud Computing," in *High-*
3716 *Performance Grid and Cloud Computing Workshop at the 28th IEEE Inter-*
3717 *national Parallel and Distributed Processing Symposium*, IEEE. Pheonix, AZ:
3718 IEEE, 05/2014 2014.
- 3719 [217] L. Vu, H. Sivaraman, and R. Bidarkar, "Gpu virtualization for high
3720 performance general purpose computing on the esx hypervisor," in *Proceedings*
3721 *of the High Performance Computing Symposium*, ser. HPC '14. San Diego,
3722 CA, USA: Society for Computer Simulation International, 2014, pp. 2:1–2:8.
3723 [Online]. Available: <http://dl.acm.org/citation.cfm?id=2663510.2663512>
- 3724 [218] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance
3725 study on 10 gbe nics with sr-iov support," in *Parallel Distributed Processing*
3726 (*IPDPS*), 2010 IEEE International Symposium on, April 2010, pp. 1–12.
- 3727 [219] T. P. P. D. L. Ruivo, G. B. Altayo, G. Garzoglio, S. Timm, H. Kim, S.-Y.
3728 Noh, and I. Raicu, "Exploring infiniband hardware virtualization in opennebula
3729 towards efficient high-performance computing." in *CCGRID*, 2014, pp. 943–948.
- 3730 [220] "NVIDIA GPUDirect," <https://developer.nvidia.com/gpudirect>, [Online; ac-
3731 cessed Nov. 24, 2014].

- 3732 [221] G. Shainer, A. Ayoub, P. Lui, T. Liu, M. Kagan, C. R. Trott, G. Scantlen, and
3733 P. S. Crozier, “The development of mellanox/nvidia gpudirect over infinibanda
3734 new model for gpu to gpu communications,” *Computer Science-Research and*
3735 *Development*, vol. 26, no. 3-4, pp. 267–273, 2011.
- 3736 [222] “Getting Xen working for Intel(R) Xeon Phi(tm)
3737 Coprocessor,” <https://software.intel.com/en-us/articles/getting-xen-working-for-intelr-xeon-phitm-coprocessor>, [Online; accessed
3738 Nov. 24, 2014].
- 3740 [223] “Mellanox Neutron Plugin,” <https://wiki.openstack.org/wiki/Mellanox-Neutron>, [Online; accessed Nov. 24, 2014].
- 3742 [224] S. Plimpton, P. Crozier, and A. Thompson, “Lammps-large-scale
3743 atomic/molecular massively parallel simulator,” *Sandia National Laboratories*, 2007.
- 3745 [225] J. Anderson, A. Keys, C. Phillips, T. Dac Nguyen, and S. Glotzer, “Hoomd-
3746 blue, general-purpose many-body dynamics on the gpu,” in *APS Meeting Abstracts*, vol. 1, 2010, p. 18008.
- 3748 [226] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer,
3749 D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, “The land-
3750 scape of parallel computing research: A view from berkeley,” Technical Report
3751 UCB/EECS-2006-183, EECS Department, University of California, Berkeley,
3752 Tech. Rep., 2006.
- 3753 [227] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo,
3754 S. Smallen, W. Smith, and A. Grimshaw, “Futuregrida reconfigurable testbed

- 3755 for cloud, hpc and grid computing,” *Contemporary High Performance Computing: From Petascale toward Exascale, Computational Science*. Chapman and
3756 Hall/CRC, 2013.
- 3757
- 3758 [228] K. Keahey, J. Mambretti, D. K. Panda, P. Rad, W. Smith, and
3759 D. Stanzione, “Nsf chameleon cloud,” website, November 2014. [Online].
3760 Available: <http://www.chameleoncloud.org/>
- 3761 [229] M. Rosenblum and T. Garfinkel, “Virtual machine monitors: Current technol-
3762 ogy and future trends,” *Computer*, vol. 38, no. 5, pp. 39–47, 2005.
- 3763 [230] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and
3764 A. Warfield, “Live migration of virtual machines,” in *Proceedings of the 2nd*
3765 *conference on Symposium on Networked Systems Design & Implementation-*
3766 *Volume 2*. USENIX Association, 2005, pp. 273–286.
- 3767 [231] M. R. Hines and K. Gopalan, “Post-copy based live virtual machine migration
3768 using adaptive pre-paging and dynamic self-ballooning,” in *Proceedings of the*
3769 *2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution*
3770 *environments*. ACM, 2009, pp. 51–60.
- 3771 [232] P. Lu, A. Barbalace, and B. Ravindran, “Hsg-lm: Hybrid-copy speculative
3772 guest os live migration without hypervisor,” in *Proceedings of the*
3773 *6th International Systems and Storage Conference*, ser. SYSTOR ’13.
3774 New York, NY, USA: ACM, 2013, pp. 2:1–2:11. [Online]. Available:
3775 <http://doi.acm.org/10.1145/2485732.2485736>
- 3776 [233] J. Chu and V. Kashyap, “Transmission of IP over InfiniBand (IPoIB),” IETF,
3777 Tech. Rep., 2006.

- 3778 [234] W. Yu, N. S. Rao, P. Wyckoff, and J. S. Vetter, “Performance of rdma-capable
3779 storage protocols on wide-area network,” in *2008 3rd Petascale Data Storage
3780 Workshop*. IEEE, 2008, pp. 1–5.
- 3781 [235] W. Huang, Q. Gao, J. Liu, and D. K. Panda, “High performance virtual machine
3782 migration with rdma over modern interconnects,” in *2007 IEEE International
3783 Conference on Cluster Computing*. IEEE, 2007, pp. 11–20.
- 3784 [236] D. Gilbert, “Post copy live migration,” Webpage, 2015. [Online]. Available:
3785 <http://wiki.qemu.org/Features/PostCopyLiveMigration>
- 3786 [237] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer,
3787 K. D. Underwood, and R. C. Zak, “Intel omni-path architecture: Enabling
3788 scalable, high performance fabrics,” in *2015 IEEE 23rd Annual Symposium on
3789 High-Performance Interconnects*, Aug 2015, pp. 1–9.
- 3790 [238] M. Beck and M. Kagan, “Performance evaluation of the rdma over ethernet
3791 (roce) standard in enterprise data centers infrastructure,” in *Proceedings of
3792 the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*.
3793 International Teletraffic Congress, 2011, pp. 9–15.
- 3794 [239] E. Kissel and M. Swany, “Photon: Remote memory access middleware for high-
3795 performance runtime systems,” in *2016 IEEE International Parallel and Dis-
3796 tributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 1736–
3797 1743.
- 3798 [240] J. Lofstead, I. Jimenez, and C. Maltzahn, “Consistency and fault tolerance con-
3799 siderations for the next iteration of the doe fast forward storage and io project,”
3800 in *2014 43rd International Conference on Parallel Processing Workshops*, Sept
3801 2014, pp. 61–69.

- 3802 [241] C. G. Wright Jr, “Trinity burst buffer-architecture and design,” Los Alamos
3803 National Laboratory (LANL), Tech. Rep., 2015.
- 3804 [242] F. B. Schmuck and R. L. Haskin, “Gpfss: A shared-disk file system for large
3805 computing clusters.” in *FAST*, vol. 2, 2002, pp. 231–244.
- 3806 [243] M. J. Rashti and A. Afsahi, “10-gigabit iwarps ethernet: comparative perfor-
3807 mance analysis with infiniband and myrinet-10g,” in *2007 IEEE International*
3808 *Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.
- 3809 [244] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, “Image processing on a
3810 simd/spmd architecture: Opsila,” in *Pattern Recognition, 1988., 9th Interna-*
3811 *tional Conference on*, Nov 1988, pp. 430–433 vol.1.
- 3812 [245] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble,
3813 E. De Lara, M. Brudno, and M. Satyanarayanan, “Snowflock: rapid virtual
3814 machine cloning for cloud computing,” in *Proceedings of the 4th ACM European*
3815 *conference on Computer systems*. ACM, 2009, pp. 1–12.
- 3816 [246] H. A. Lagar-Cavilla, J. A. Whitney, R. Bryant, P. Patchin, M. Brudno,
3817 E. de Lara, S. M. Rumble, M. Satyanarayanan, and A. Scannell, “Snowflock:
3818 Virtual machine cloning as a first-class cloud primitive,” *ACM Transactions on*
3819 *Computer Systems (TOCS)*, vol. 29, no. 1, p. 2, 2011.
- 3820 [247] A. J. Younge, G. von Laszewski, L. Wang, and G. C. Fox, “Providing a Green
3821 Framework for Cloud Based Data Centers,” in *The Handbook of Energy-Aware*
3822 *Green Computing*, I. Ahmad and S. Ranka, Eds. Chapman and Hall/CRC
3823 Press, 2011, ch. 17, in press.
- 3824 [248] D. P. Berrange, “Openstack performance optimization,” in *KVM Forum 2014*,
3825 2014.

- 3826 [249] Y. Jiang and Y. He, “Openstack pci passthrough,” Webpage, Intel, Inc, 2016.
3827 [Online]. Available: https://wiki.openstack.org/wiki/Pci_passthrough
- 3828 [250] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Lapacz,
3829 D. M. Swany, S. Trocha, and J. Zurawski, “Perfsonar: A service oriented archi-
3830 tecture for multi-domain network monitoring,” in *International Conference on*
3831 *Service-Oriented Computing*. Springer, 2005, pp. 241–254.
- 3832 [251] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena,
3833 L. Arantes, and P. Sens, “Towards qos-oriented sla guarantees for online
3834 cloud services,” in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th*
3835 *IEEE/ACM International Symposium on*. IEEE, 2013, pp. 50–57.
- 3836 [252] K. Pedretti, “Kitten: A lightweight operating system for ultrасcale supercom-
3837 puters,” 2011.
- 3838 [253] J. Ahrens, D. Rogers, and B. Springmeyer, “Visualization and data analysis at
3839 the exascale,” *National Nuclear Security Administration (NNSA) Accelerated*
3840 *Strategic Computing (ASC) Exascale Environment Planning Process*, vol. 2,
3841 2010.
- 3842 [254] V. Vishwanath, M. Hereld, and M. E. Papka, “Toward simulation-time data
3843 analysis and i/o acceleration on leadership-class systems,” in *Large Data Anal-*
3844 *ysis and Visualization (LDAV), 2011 IEEE Symposium on*. IEEE, 2011, pp.
3845 9–14.