

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

October 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

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.

ABSTRACT APPROVAL

As committee members of the candidate's graduate committee, I have read the dissertation abstract of Andrew J. Younge and found it in a form satisfactory to the graduate committee.

Date Geoffrey C. Fox, Ph.D, Chair

Date Judy Qiu, Ph.D

Date Thomas Sterling, Ph.D

Date D. Martin Swany, Ph.D

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	7
1.3 Research Challenges	12
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	Abstract	96
5.2	Introduction	97
5.3	Related Work & Background	98
5.3.1	GPU API Remoting	99
5.3.2	PCI Passthrough	99
5.3.3	GPU Passthrough, a Special Case of PCI Passthrough	100
5.4	Experimental Methodology	101
5.4.1	Host and Hypervisor Configuration	101
5.4.2	Guest Configuration	103
5.4.3	Microbenchmarks	103
5.4.4	Application Benchmarks	103
5.5	Performance Results	105
5.5.1	SHOC Benchmark Performance	106
5.5.2	GPU-LIBSVM Performance	112
5.5.3	LAMMPS Performance	114
5.5.4	LULESH Performance	116
5.6	Lessons Learned	116
5.7	Directions for Future Work	118

6 Supporting High Performance Molecular Dynamics in Virtualized Clusters using IOMMU, SR-IOV, and GPUDirect	120
6.1 Abstract	120
6.2 Introduction	121
6.3 Background and Related Work	124
6.3.1 GPU Passthrough	125
6.3.2 SR-IOV and InfiniBand	126
6.3.3 GPUDirect	127
6.4 A Cloud for High Performance Computing	128
6.5 Benchmarks	129
6.6 Experimental Setup	131
6.6.1 Node configuration	131
6.6.2 Cluster Configuration	132
6.7 Results	133
6.7.1 LAMMPS	133
6.7.2 HOOMD	136
6.8 Discussion	136
6.9 Chapter Summary	138
7 Virtualization advancements to support HPC applications	139
7.1 PCI Passthrough	143
7.2 Memory Page Table Optimizations	151
7.3 Live Migration Mechanisms	157
7.3.1 RDMA-enabled VM Migration	159
7.3.2 Moving the Compute to the Data	162
7.4 Fast VM Cloning	164
7.5 Virtual Cluster Scheduling	167
7.6 Chapter Summary	170
8 Conclusion	171
8.1 Impact	173
Bibliography	175

List of Figures

1.1	Data analytics and computing ecosystem compared (from [25]), with virtualization included	6
1.2	High Performance Virtual Cluster Framework	9
1.3	Architectural diagram for High Performance Virtual Clusters	11
2.1	Virtual Machine Abstraction	19
2.2	Hypervisors and Containers	22
2.3	View of the Layers within a Cloud Infrastructure	26
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.	108
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.	109
5.3	SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.	110
5.4	SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.	111
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.	117
6.1	Node PCI Passthrough of GPUs and InfiniBand	132
6.2	LAMMPS LJ Performance	134
6.3	LAMMPS RHODO Performance	135
6.4	HOOMD LJ Performance with 256k Simulation	137
7.1	Comparison of GPU Passthrough in Xen to rCUDA across various interconnects	147
7.2	Efficient VM networking comparison [206]	148
7.3	SR-IOV Architecture with Virtual Functions, PCI SIG [235]	150
7.4	Native TLB miss walk compared with 2D virtualized TLB miss walk from [237]. On the left illustrates a native page table walk. On the right illustrates the lengthy 2D nested page table walk for a VM.	154
7.5	Transparent Huge Pages with KVM	156
7.6	Adding extra specs to a VM flavor in OpenStack	168

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 is
¹⁶ 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 At 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. Virtualization has existed for
32 many years, but its availability with Intel x86 commodity hardware in conjunction
33 with the rise of clouds has brought it 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 as a field, which has a different evolution. Gone are
36 the IBM Mainframes of the 1980s, which dominated the enterprise landscape. While
37 some mainframes still exist, today they are used only for batch related processing
38 tasks and not for scientific applications as they are inefficient at Floating Point Op-
39 erations. As such, Beowulf Clusters [6], Massively Parallel Processors (MPPs) and
40 Supercomputers of the 90s and 00s replaced the mainframes of before. A novelty of
41 these distributed memory systems is that instead of just one large machine, many
42 machines are connected together to achieve a common goal, thereby maximizing the
43 overall speed of computation. Clusters represent a more commodity-based super-

44 computer with off-the-shelf CPUs instead of the highly customized and expensive
45 processors and interconnects found in Supercomputers. Supercomputers and Clus-
46 ters are best suited for large-scale applications, due to their innate ability to divide
47 the computational efforts into smaller concurrent tasks. These tasks often run in par-
48 allel on many CPUs, whereby each task communicates results to other tasks in some
49 way as per the application design. These HPC applications can even include “Grand
50 Challenge” applications [7] and can represent a sizable amount of the scientific calcu-
51 lations done on large-scale Supercomputing resources today. However, there exists a
52 gap of many orders of magnitude between leading-class high performance computing
53 and what is available on the common laboratory workshop. This gap, described here
54 as mid-tier scientific computation, is a fast growing field that struggles to harness dis-
55 tributed systems efficiently while hoping to minimize extensive development efforts.
56 These mid-tier scientific endeavors need to leverage distributed systems to complete
57 the calculations at hand effectively, however, they may not require the extreme scale
58 provided by the latest machines at the peak of the Top500 list [8]. Some small research
59 teams may not have the resources available or the desire to handle the development
60 complexity of high end supercomputing systems, and can look towards other options
61 instead. This can include some scientific disciplines such as high energy physics [9],
62 materials science [10], bioinformatics [11], and climate research [12], to name a few.

63 As more domain science turns to the aid of computational resources for conducting
64 novel scientific endeavors, there is a continuing and growing need for national cyber-
65 infrastructure initiatives to support an increasingly diverse set of scientific workloads.
66 Historically, there have been a number of national and international cyberinfrastruc-
67 ture efforts to support high end computing. These range from traditional supercom-
68 puters deployed at Department of Energy computing facilities [13], to efforts led by
69 the National Science Foundation such as the XSEDE environment [14] or domain spe-

70 cific initiatives such as the NSF iPlant collaborative [15]. Substantial growth can be
71 seen in the number of computational resource requests [14, 16] from many of the larger
72 computational facilities. Concurrently, there has also been an increase in accelerators
73 and hybrid computing models capable of quickly providing additional resources [17]
74 beyond commodity clusters.

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

89 However, many big-data paradigms [20] in distributed systems have been intro-
90 duced that represent new computational models for distributed computing, such as
91 MapReduce [21] and the corresponding Apache Big Data stack [22, 23]. While the
92 term big data can mean a number of things, it generally refers to data sets large or
93 complex enough to require advanced, non-traditional data processing mechanisms in
94 order to extract feature knowledge. Some HTC and HPC applications could poten-
95 tially be considered big data tools, generally the nomenclature has grown out of new

96 industry and academic problem sets created by a data deluge [24]. This has lead to
97 a number of novel platform services for handling big data problems in parallel, and
98 supporting these different distributed computational paradigms requires a flexible in-
99 frastructure capable of providing computational resources for all possible models in
100 a fast and efficient manner.

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

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

122 success of the research in virtualization could be defined by the ability to support the
123 convergence between HPC and Big Data.

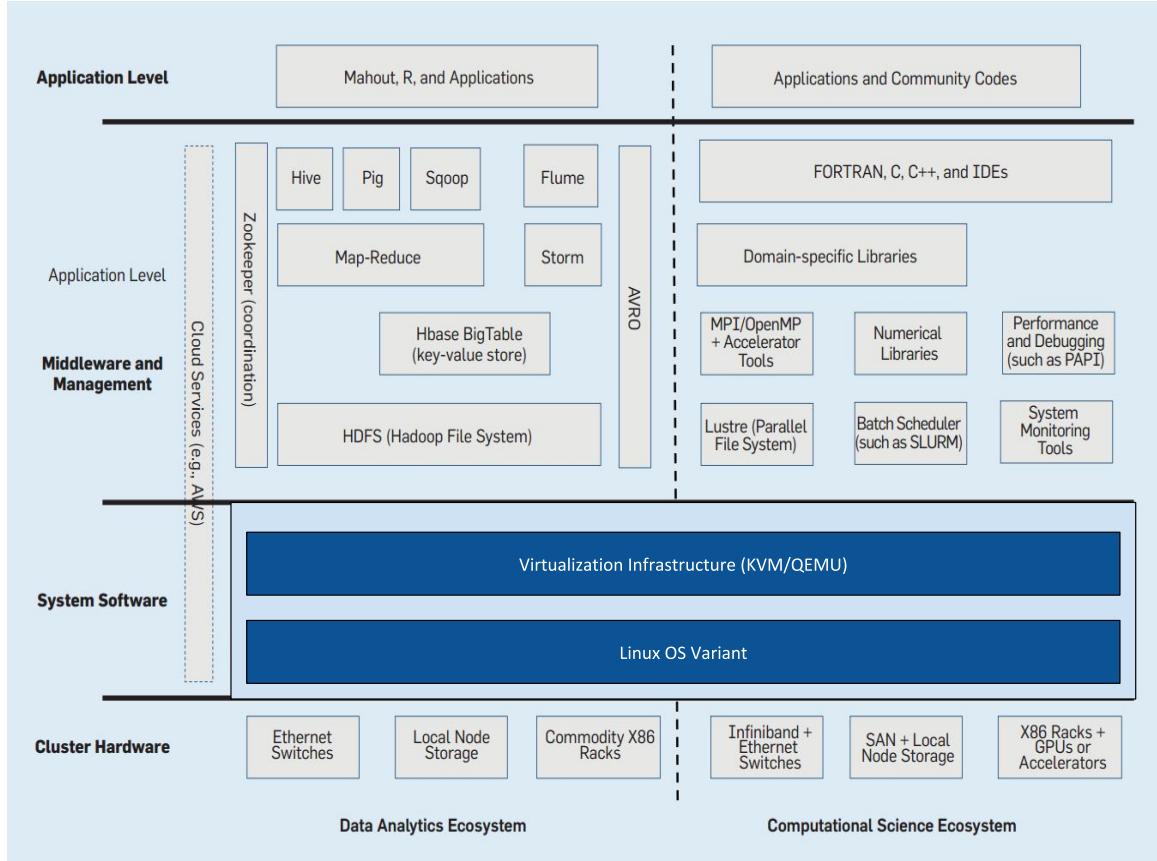


Figure 1.1 Data analytics and computing ecosystem compared (from [25]), with virtualization included

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

₁₃₂ at a critical level.

₁₃₃ This work proposes the use of virtual clusters [32] to provide distinct, separate
₁₃₄ environments on similar resources using virtual machines. These virtual clusters,
₁₃₅ similar in design to the Beowulf clusters and commodity HPC systems, provide a
₁₃₆ distributed memory environment, usually with a local resource management system
₁₃₇ [33]. However, past concerns with virtualization have limited the adoption of virtual
₁₃₈ clusters in many large-scale cyberinfrastructure deployments. This has in part been
₁₃₉ due to the overhead of virtualization, whereby many scientific computations have
₁₄₀ experienced a significant and notable degradation in performance. In an ecosystem
₁₄₁ familiar with HPC systems in which performance is paramount, this has been an
₁₄₂ obstructive hurdle for deploying many tightly coupled applications.

₁₄₃ **1.2 Research Statement**

₁₄₄ With the rise of cloud computing within the greater realm of distributed systems,
₁₄₅ there have been a number of scientific computing endeavors that map well to cloud in-
₁₄₆ frastructure. This first includes the simple and most common practice of on-demand
₁₄₇ computing whereby users can rent-a-workstation [34]. Perhaps these resources are
₁₄₈ more powerful than a given researcher's laptop and used to run their scientific appli-
₁₄₉ cations or support greater laboratory collaborative efforts, such as a shared database
₁₅₀ or Web services. We have also seen virtualized cloud infrastructure support high
₁₅₁ throughput computing very well. Often times pleasingly parallel applications, be it
₁₅₂ from high energy physics such as the LHC effort [9,35] or bioinformatics with BLAST
₁₅₃ alignment jobs [11], have proven to run with high efficiency in public and private cloud
₁₅₄ environments. Furthermore, the rise of public cloud infrastructure has also coincided
₁₅₅ with increase in big data computation and analytics. Many of these big data plat-

156 form services have evolved complimentary to cloud infrastructure, and as such have
157 a symbiotic relationship with virtualization technologies [36].

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

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

176 Given the current outlook on virtualization for supporting HPC applications, this
177 dissertation proposes a framework for High Performance Virtual Clusters that enable
178 advanced computational workloads, including tightly coupled distributed memory ap-
179 plications, to run with a high degree of efficiency in virtualized environments. This
180 framework, outlined in Figure 1.2, illustrates the topics to be addressed to provide
181 a supportive virtual cluster environment for high performance mid-tier scientific ap-

182 plications. Areas marked in darker green indicate topics this dissertation may touch
 183 upon, whereas light green areas in Figure 1.2 identify outstanding considerations to
 184 be investigated. Specifically, mid-tier distributed memory parallel computations is
 185 identified as a focal point for the computational challenges at hand as a way to sep-
 186 arate from some of the latest efforts towards Exascale computing [38–40]. While
 187 virtualization may in fact be able to play a role towards usable Exascale computing,
 188 such efforts fall outside the immediate scope of this dissertation.

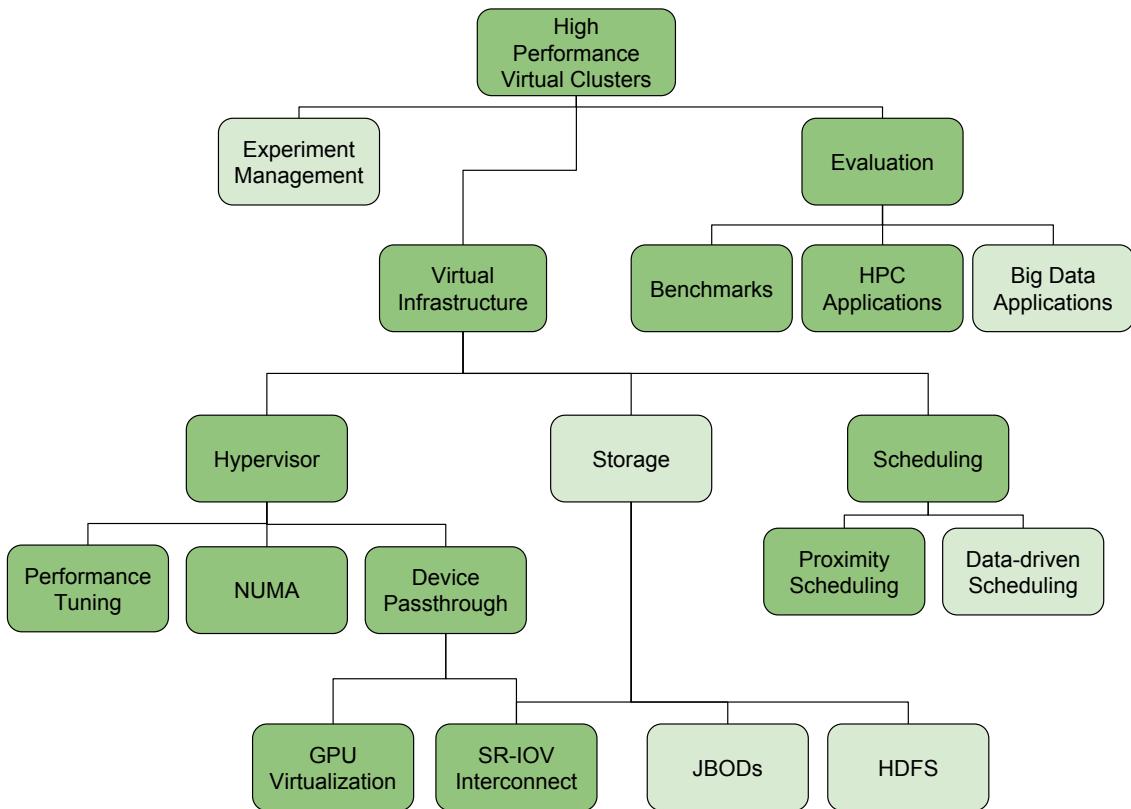


Figure 1.2 High Performance Virtual Cluster Framework

189 In order to provide high performance virtual clusters, there is a need to first look at
 190 a key area, the virtualized infrastructure itself. At the core, the hypervisor, or virtual
 191 machine monitor, has to be considered and the overhead and performance characteris-
 192 tics associated with it. This includes performance tuning considerations, non-uniform

193 memory access (NUMA) effects, and advanced hardware device passthrough. Specifi-
194 cally, device passthrough in the context of this manuscript refers to two major device
195 types; GPUs and InfiniBand interconnects (the later using SR-IOV). The virtual in-
196 frastructure also must consider scheduling as a major factor in performing efficient
197 placement of workloads on the underlying host infrastructure, and in particular a
198 Proximity scheduler is of interest [41]. Storage solutions in a virtualized environment
199 is an increasingly important aspect of this framework, as some HPC and big data
200 solutions are prioritizing I/O performance more than computation. Storage is also
201 likely to be heavily dependent on interconnect considerations as well, which could
202 be potentially provided by device passthrough. However, such I/O considerations lie
203 beyond this dissertation’s immediate scope.

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

215 Building from the historical virtual clusters in Grid computing, we see the new ar-
216 chitectural model for high performance virtual clusters illustrated in Figure 1.3 that
217 is implied by this dissertation. Here, we leverage commodity hardware, as well as
218 some advanced HPC hardware. While this notion could incorporate a wide array of

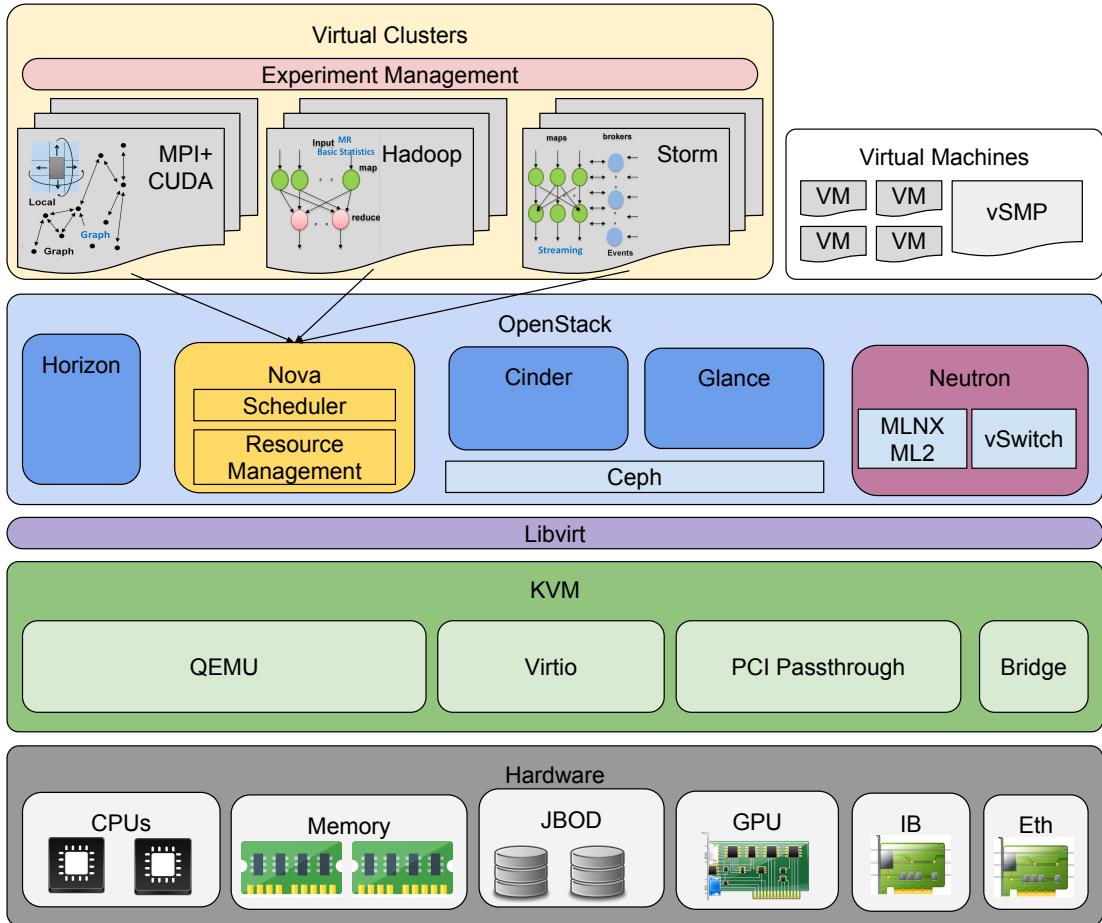


Figure 1.3 Architectural diagram for High Performance Virtual Clusters

219 differing technologies, we focus here on GPU-based accelerators and InfiniBand inter-
 220 connects in conjunction with x86 CPUs. Atop this, we leverage KVM and QEMU to
 221 provide an advanced hypervisor for creating and hosting VMs with direct hardware
 222 involvement from the lower level. Moving up, the Libvirt API is leveraged due to
 223 its hypervisor interoperability and popularity. Atop Libvirt is the OpenStack private
 224 cloud infrastructure. In Figure 1.3 we illustrate some (but not all) of OpenStack's
 225 services including the Horizon UI, Cinder and Glance storage mechanisms, and the
 226 Neutron (previously Quantum) components. The Nova component of OpenStack is

227 the point of focus for providing comprehensive VM management.¹ Atop OpenStack,
228 we can create a wide array of virtual clusters and machines to support diverse sci-
229 entific computing ecosystems necessary. This includes application models ranging
230 from tightly coupled MPI+CUDA HPC applications, to emerging big data analytics
231 toolkits such as Apache Storm [42].

232 One of the higher-level aspects of providing high performance virtual clusters is the
233 high level orchestration of the virtual clusters themselves, which we term experiment
234 management. While this largely remains tangential to this immediate research, it is
235 nonetheless a key aspect for a successful solution. Some effort has been put forth for
236 virtual cluster experiment management [43], and many ongoing open sources solutions
237 also offer compelling options, such as OpenStack Heat [44]. An example of a project
238 delivering advanced orchestration mechanisms and a toolkit to aid in configurable
239 virtual clusters on heterogeneous IaaS deployments is the Cloudmesh effort [45].

240 1.3 Research Challenges

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

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

248 The notion that virtualization and Cloud infrastructure are not able to support
249 parallel distributed memory applications has been characterized many times. One
250 of the most prominent examples of this is the Department of Energy's Magellan
251 Project [46], whereby the Magellan Final Report [47] states the following finding as
252 a Key Finding:

253 **“Finding 2. Scientific applications with minimal communication
254 and I/O are best suited for clouds.**

255 We have used a range of application benchmarks and micro-benchmarks
256 to understand the performance of scientific applications. Performance of
257 tightly coupled applications running on virtualized clouds using commodi-
258 ty networks can be significantly lower than on clusters optimized for these
259 workloads. This can be true at even mid-range computing scales. For ex-
260 ample, we observed slowdowns of about 50x for PARATEC on the Amazon
261 EC2 instances compared to Magellan bare-metal (non-virtualized) at 64
262 cores and about 7x slower at 1024 cores on Amazon Cluster Compute in-
263 stances, the specialized high performance computing offering. As a result,
264 current cloud systems are best suited for high-throughput, loosely coupled
265 applications with modest data requirements.“

266 These findings underscore how classical usage of virtualization in cloud infrastruc-
267 ture has serious performance issues when running tightly coupled distributed memory
268 applications. Many of these performance concerns are sound, given the limitation of a
269 number of virtualization overheads commonplace at the time, including shadow page
270 tables, emulated Ethernet drivers, experimental hypervisors, and a complete lack of
271 sophisticated hardware commonplace in supercomputers and clusters. As a result, the
272 advantages of virtualization, including on-demand resource allocation, live migration

273 and advanced hybrid migration, and user-defined environments, have not been able
274 to show effectively their value in the context of the HPC community.

275 Other and related efforts within the scientific community too found limitations
276 with HPC applications in public cloud environments. This includes the study by
277 Jackson et. al [48] which illustrates how the Amazon Elastic Compute Cloud (EC2)
278 creates a 6x performance impact compared to a local cluster, due in large part to the
279 limiting Gigabit Ethernet on which benchmarks relied heavily within the EC2 system.
280 Other studies also found similar results; Ostermann [37] for instance, concludes that
281 Amazon EC2 “is insufficient for scientific computing at large, though it still appeals
282 to the scientists that need resources immediately and temporarily.” However, these
283 studies are now outdated and do not take into account the hardware and advance-
284 ments in virtualization detailed in this dissertation. Specifically, it is estimated that
285 with the KVM hypervisor in a performance-tuned environment, using accelerators
286 and most certainly a high-speed, low latency interconnect as detailed in Chapter 6,
287 the results would be drastically different.

288 One limitation in this research in high performance virtual clusters is the fact that
289 the degree to which applications can scale remains relatively unknown. While initial
290 results with SR-IOV InfiniBand are promising, scaling is naturally hard to predict. It
291 would be hypothetically possible that as the number of VM’s increases, interconnect
292 communication tail-latency could also increase, causing notable slowdowns during
293 distributed memory synchronization barriers. It is only when infrastructure able to
294 support high performance virtual clusters becomes available will scaling to thousands
295 of cores and beyond be investigated.

296 Another potential challenge, and perhaps also a strength, is the rising use of
297 containers within industry, such as we see in efforts like Docker [49]. Recently, we
298 have seen efforts at NERSC/LBNL adapt a container solution called Shifter with the

299 SLURM resource manager on CRAY XC based systems [50]. Shifter’s goal is to pro-
300 vide user defined images for NERSC’s bioinformatics users, and it adapts remarkably
301 well to the HPC environment. While further efforts are needed by Cray and NERSC
302 to fully provide a container-based solution on a large-scale Supercomputer for all ap-
303 plications, its efforts are in many ways related to virtualization. In Chapter 5, we
304 specifically compare virtualization efforts with LXC [51], a popular Linux container
305 solution, and find performance to be comparable and largely near-native.

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

317 **1.4 Outline**

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

320 Chapter 2 investigates the related research surrounding both cloud computing
321 and high performance computing. Within cloud computing, an introduction to cloud
322 infrastructure, virtualization, and containers will all be discussed. This also includes

323 details regarding virtual clusters as well as an overview of some national scale cloud
324 infrastructure efforts that exist. Furthermore, we investigate the state of high per-
325 formance computing and supercomputing, as well touch upon some of the current
326 Exascale efforts.

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

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

340 Chapter 5 continues where chapter 4 leaves off, by demonstrating that GPU
341 passthrough is possible on many other hypervisors, specifically also KVM and VMWare,
342 and compares with one of the main containerization solutions, LXC. Here, the GPUs
343 are evaluated using not only the SHOC GPU benchmark suite developed at Oak
344 Ridge National Laboratory, but also a diverse mix of real-world applications in order
345 to examine how and where overhead exists with GPUs in VMs for each virtualization
346 setup. Specifically, we find that with properly tuned hardware and NUMA-balanced
347 configurations, the KVM solution can perform at roughly near-native performance,
348 with on average 1.5% overhead compared to no virtualization. This illustrates that

349 with the correct hypervisor selection, careful tuning, and advanced hardware, scien-
350 tific computations can be supported using virtualized hardware.

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

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

367 Finally, this work concludes with an overall view of the current state of high
368 performance virtualization, as well as its potential to impact and support a wide
369 array of disciplines.

³⁷⁰ **Chapter 2**

³⁷¹ **Related Research**

³⁷² In order to depict the research presented in this article accurately, the topics within
³⁷³ Virtualization, Cloud computing, and High Performance Computing are reviewed in
³⁷⁴ detail.

³⁷⁵ **2.1 Virtualization**

³⁷⁶ Virtualization is a way to abstract the hardware and system resources from an op-
³⁷⁷ erating system. This is typically performed within a Cloud environment across a
³⁷⁸ large set of servers using a Hypervisor or Virtual Machine Monitor (VMM), which
³⁷⁹ lies in between the hardware and the Operating System (OS). From here, one or
³⁸⁰ more virtualized OSs can be started concurrently, as seen in Figure 2.1, leading to
³⁸¹ one of the key advantages of virtualization. This, along with the advent of multi-core
³⁸² processing capabilities, allows for a consolidation of resources within a data center.
³⁸³ It then becomes the cloud infrastructure's job, as discussed in the next section, to
³⁸⁴ exploit this capability to its maximum potential while still maintaining a given QoS.
³⁸⁵ It should be noted that virtualization is not specific to Cloud computing. IBM orig-

³⁸⁶ finally pioneered the concept in the 1960's with the M44/44X systems. It has only
³⁸⁷ recently been reintroduced for general use on x86 platforms.

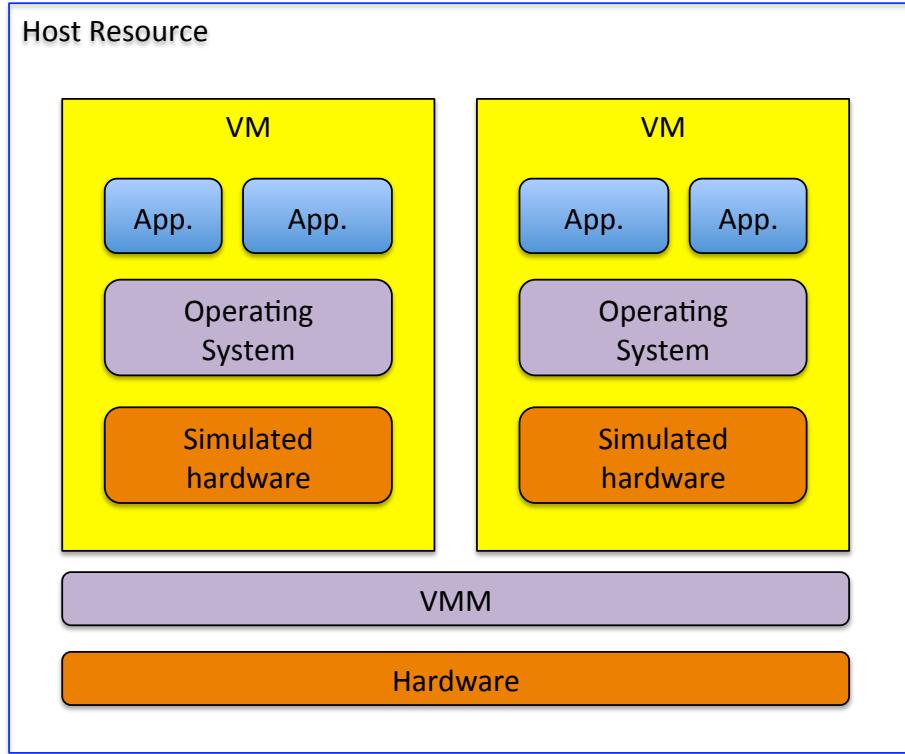


Figure 2.1 Virtual Machine Abstraction

³⁸⁸ However, virtualization, in the most general sense, is just another form of abstrac-
³⁸⁹ tion. As such, there are in fact many levels to virtualization that exist [52].

- ³⁹⁰ • **ISA -** Virtualization can start from the instruction set architecture (ISA) level,
³⁹¹ whereby an entire processor instruction set is emulated. This may be useful for
³⁹² running software or services developed for one instruction set (say MIPS), but
³⁹³ has to run on modern Intel x86 hardware. ISA level virtualization is usually
³⁹⁴ emulated through an interpreter which translates source instructions to target
³⁹⁵ instructions; however this can often be extremely inefficient. Dynamic binary
³⁹⁶ translation can help aid in efficiency by translating blocks of source instructions
³⁹⁷ to target instructions. This still can be limiting.

- 398 ● **Hardware -** One of the most important technologies in cloud computing is
399 hardware level virtualization [53, 54]. Hardware virtualization, in its most pure
400 form, refers to the process of creating virtual abstraction to hardware platforms,
401 operating systems, or software resources. This enables the creation of 1 or
402 more virtual machines (VMs) that are run concurrently on the same operating
403 environment, be it hardware or some higher software. Here, a virtual hardware
404 environment is generated for a VM, providing virtual processors, memory, and
405 I/O devices that allow for VM multiplexing, as depicted in Figure 2.1. This
406 layer also manages the physical hardware on behalf of a host OS, as well as for
407 guests. While the most popular implementations of hardware virtualization is
408 the Xen Virtual Machine Monitor [53]; this method has further separated into
409 type 1 and type 2 hypervisors, as detailed further in Section 2.1.1.

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

- 419 ● **Library (API) -** Moving up a layer, we find library or Application Level
420 Interface (API) level virtualization, where a programming interface is separated
421 and the communication between this API and the back-end library with the
422 computation is virtualized. This method removes the back end services from

423 within the operating environment, which can give the effect that resources are
424 local, when they are in fact remote. This is how GPUs are "virtualized" with
425 tools such as rCUDA [55] and vCUDA [56]. This method's performance often
426 can be very dependent on the underlying communications mechanisms, as well
427 as complete virtualization of the whole API (which may be difficult). API
428 virtualization also exists for entire OS libraries to translate between differing
429 OS, without actual containerization.

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

435 2.1.1 Hypervisors and Containers

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

441 Hardware virtualization can actually be dissected in more detail to two different
442 types of hypervisors. These hypervisor types are illustrated in Figure 2.2, and they
443 can be directly compared to containers. With a Type 1 virtualization system, the
444 hypervisor or VMM sits directly on the bare-metal hardware, below the OS. These na-
445 tive hypervisors provide direct control of the underlying hardware, and are controlled
446 and operated usually through the use of a privileged VM. One example of a type 1

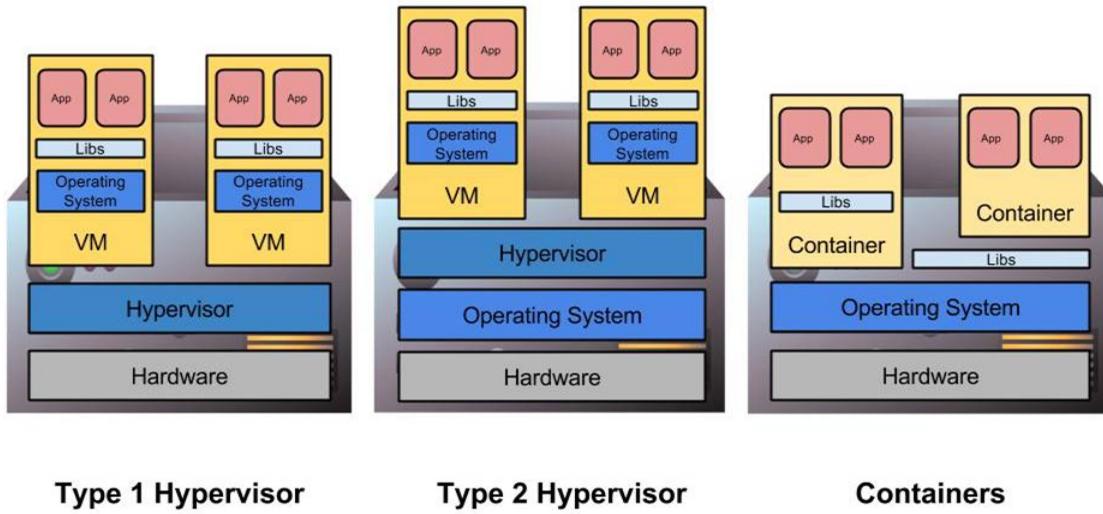


Figure 2.2 Hypervisors and Containers

447 hypervisor is the Xen Virtual Machine Monitor [53], which uses its VMM as well as a
 448 privileged Linux OS, called Dom0, to create and manage other user DomU instances.
 449 The VMM in this place provides the necessary hardware abstraction of CPU, mem-
 450 ory, and some I/O aspects, leaving the control aspects of the other DomUs to Dom0.
 451 With a Type 1 hypervisor, all virtualization functions are kept separate from control
 452 and OS functionality, effectively making a cleaner design. This design could lead to
 453 end application performance implications, as illustrated with the Palacios VMM [31].

454 Type 2 hypervisors utilize a different - and sometimes more convoluted - design.
 455 With a Type 2 hypervisor, there is a "host" OS that, like with native OSs, sits
 456 directly atop hardware. This OS is just like any normal native OS. However, the OS
 457 itself can abstract its own hardware, and provide and manage a VM, effectively as
 458 an OS process. In this case, the hypervisor providing the abstraction is effectively
 459 built within, atop, or as a module within the kernel. There are many different Type 2
 460 hypervisors, the most common of which is the Linux Kernel Virtual Machine (KVM)
 461 [57]. KVM is often used in conjunction with QEMU, an ISA level emulator, to
 462 provide some basic ISA level virtualization and emulation capabilities. KVM is simply

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

465 Type 1 and Type 2 hypervisors are very distinct from OS level virtualization, also
466 known as containerization. With containers, there is a single OS; however, instead
467 of direct hardware abstraction, a single kernel is used to simultaneously run multiple
468 user-space instances in a jailed-root environment. These environments may look and
469 feel like a separate machine, but in fact are not. Oftentimes the kernel itself pro-
470 vides resource management tools to help control resource utilization and allocations.
471 Linux container (LXC) is a great example of this, with their use of namespaces for
472 filesystem control and cgroups for resource management. With the recent advent of
473 Docker, which looks to control versioning and easy deployment of traceable contain-
474 ers, this aspect of OS level virtualization has grown in popularity; however, security
475 and usability concerns still exist, as dedicated isolation mechanisms do not exist and
476 the kernel space is the only point of security separation.

477 2.2 Cloud Computing

478 Cloud computing [58] is one of the most explosively expanding technologies in the
479 computing industry today. However, it is important to understand where it came
480 from in order to figure out where it will be heading in the future. While there is no
481 clear cut evolutionary path to Clouds, many believe the concepts originate from two
482 specific areas: Grid Computing and Web 2.0.

483 Grid computing [59, 60], in its practical form, represents the concept of connect-
484 ing two or more spatially and administratively diverse clusters or supercomputers
485 together in a federating manner. The term “the Grid” was coined in the mid 1990s
486 to represent a large distributed systems infrastructure for advanced scientific and en-

487 gineering computing problems. Grids aim to enable applications to harness the full
488 potential of resources through coordinated and controlled resource sharing by scalable
489 virtual organizations. While not all of these concepts carry over to the Cloud, the
490 control, federation, and dynamic sharing of resources is conceptually the same as in
491 the Grid. This is outlined by Foster et al [3], as Grids and clouds appear concep-
492 tually similar when compared abstractly. From a scientific perspective, the goals of
493 clouds and Grids are also similar. Both systems attempt to provide large amounts
494 of computing power by leveraging a multitude of sites running diverse applications
495 concurrently in symphony.

496 The other major component, Web 2.0, is also a relatively new concept in the
497 history of Computer Science. The term Web 2.0 was originally coined in 1999 in a
498 futuristic prediction by Dracy DiNucci [61]: “The Web we know now, which loads
499 into a browser window in essentially static screenfulls, is only an embryo of the Web
500 to come. The first glimmerings of Web 2.0 are beginning to appear, and we are just
501 starting to see how that embryo might develop. The Web will be understood not
502 as screenfulls of text and graphics but as a transport mechanism, the ether through
503 which interactivity happens. It will [...] appear on your computer screen, [...] on your
504 TV set [...] your car dashboard [...] your cell phone [...] hand-held game machines
505 [...] maybe even your microwave oven.” Her vision began to form, as illustrated in
506 2004 by the O’Riley Web 2.0 conference, and since then the term has been a pivotal
507 buzzword among the internet. While many definitions have been provided, Web 2.0
508 really represents the transition from static HTML to harnessing the Internet and the
509 Web as a platform in of itself.

510 Web 2.0 provides multiple levels of application services to users across the Inter-
511 net. In essence, the web becomes an application suite for users. Data is outsourced
512 to wherever it is wanted, and the users have total control over what they interact

513 with and spread accordingly. This requires extensive, dynamic, and scalable host-
514 ing resources for these applications, the demand for which provides the user-base for
515 much of the commercial Cloud computing industry today. Web 2.0 software requires
516 abstracted resources to be allocated and relinquished on the fly, depending on the
517 Web's traffic and service usage at each site. Furthermore, Web 2.0 brought into exist-
518 ence Web Services standards [62] and the Service Oriented Architecture (SOA) [63],
519 which outlined the interaction between users and cyber infrastructure. In summary,
520 Web 2.0 defined the interaction standards and user base, and Grid computing defined
521 the underlying infrastructure capabilities.

522 A Cloud computing implementation typically enables users to migrate their data
523 and computation to a remote location with some varying impact on system perfor-
524 mance [64]. This provides a number of benefits which could not otherwise be achieved:

- 525 ● *Scalable* - Clouds are designed to deliver as much computing power as any
526 user needs. While in practice the underlying infrastructure is not infinite, the
527 cloud resources are projected to ease the developer's dependence on any specific
528 hardware.
- 529 ● *Quality of Service (QoS)* - Unlike standard data centers and advanced com-
530 puting resources, a well-designed Cloud can project a much higher QoS than
531 traditionally possible. This is due to the lack of dependence on specific hard-
532 ware, so any physical machine failures can be mitigated without the prerequisite
533 user awareness.
- 534 ● *Specialized Environment* - Within a Cloud, the user can utilize customized tools
535 and services to meet their needs. This can be to utilize the latest library, toolkit,
536 or to support legacy code within new infrastructure.
- 537 ● *Cost Effective* - Users leverage only the resources required for their given task,

538 rather than putting forth a large capital investment in infrastructure. This re-
539 duces the risk for institutions potentially looking to build a scalable system,
540 thus providing greater flexibility, since the user is only paying for needed in-
541 frastructure while maintaining the option to increase services as needed in the
542 future.

- 543 • *Simplified Interface* - Whether using a specific application, a set of tools or
544 Web services, Clouds provide access to a potentially vast amount of computing
545 resources in an easy and user-centric way. We have investigated such an interface
546 within Grid systems through the use of the Cyberaide project [65, 66].

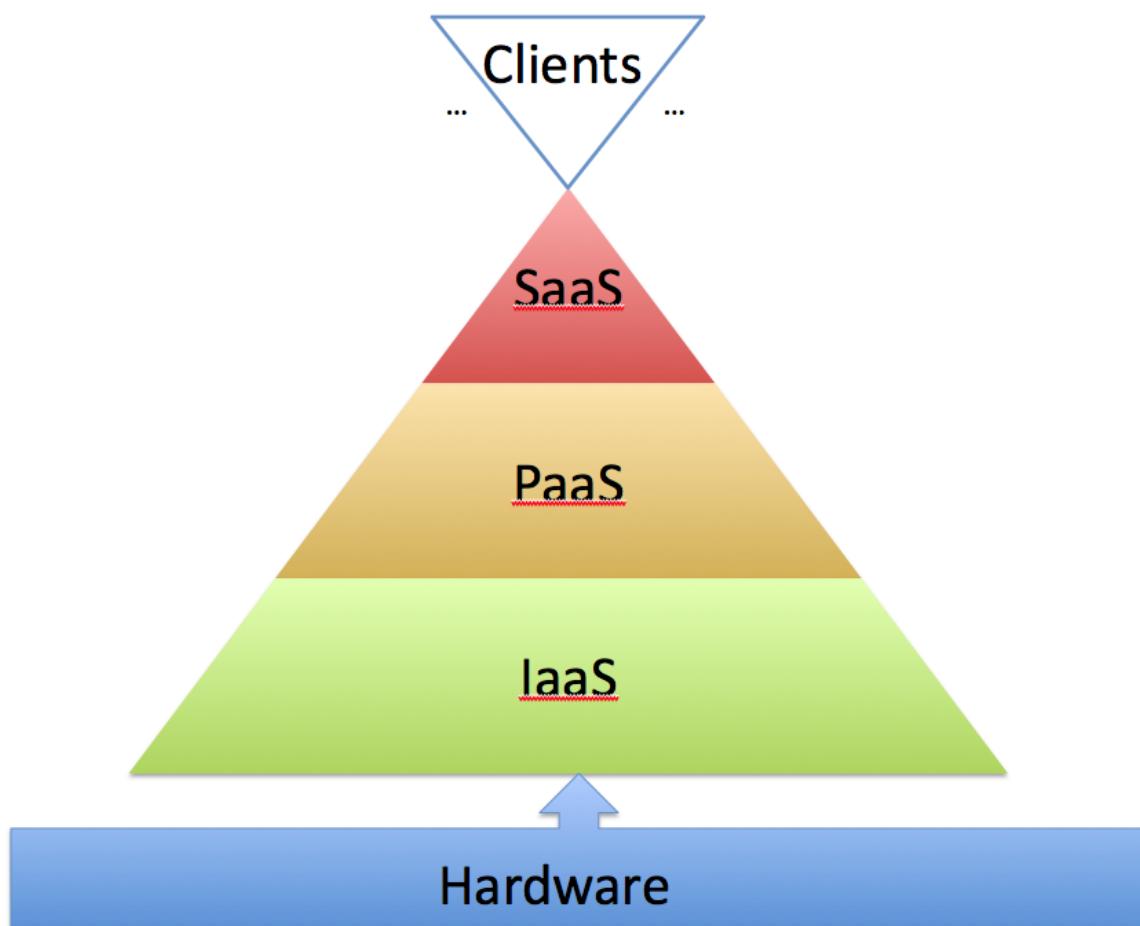


Figure 2.3 View of the Layers within a Cloud Infrastructure

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

- 552 ● *Clients* - A client interacts with a Cloud through a predefined, thin layer of
553 abstraction. This layer is responsible for communicating the user requests and
554 displaying data returned in a way that is simple and intuitive for the user.
555 Examples include a Web Browser or a thin client application.
- 556 ● *Software-as-a-Service (SaaS)* - A framework for providing applications or soft-
557 ware deployed on the Internet packaged as a unique service for users to consume.
558 By doing so, the burden of running a local application directly on the client's
559 machine is removed. Instead, all the application logic and data is managed
560 centrally and displayed to the user through a browser or thin client. Examples
561 include Google Docs, Facebook, or Pandora.
- 562 ● *Platform-as-a-Service (PaaS)* - A framework for providing a unique computing
563 platform or software stack on which applications and services can be developed.
564 The goal of PaaS is to alleviate many of the burdens of developing complex,
565 scalable software by proving a programming paradigm and tools that make ser-
566 vice development and integration a tractable task for many. Examples include
567 Microsoft Azure and Google App Engine.
- 568 ● *Infrastructure-as-a-Service (IaaS)* - A framework for providing entire computing
569 resources through a service. This typically represents virtualized Operating
570 Systems, thereby masking the underlying complexity and details of the physical
571 infrastructure. Users are allowed to rent or buy computing resources on demand

572 for their own use without needing to operate or manage physical infrastructure.
573 Examples include Amazon EC2, and OpenStack, and is the major cloud focal
574 point for this dissertation.

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

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

581 **2.2.1 Infrastructure-as-a-Service**

582 Today, there are a number of Clouds that offer solutions for Infrastructure-as-a-Service
583 (IaaS). There have been multiple comparison efforts between various IaaS service
584 [4, 67–69], which provide insight to the similarities and differences between the long
585 array of cloud infrastructure deployment solutions. However, at a high level, IaaS can
586 be split into 3 tiers based on their availability.

- 587 • **Public** - Public IaaS is where the services and virtualizaton of hardware re-
588 sources are provided over the internet. Usually this is in a centralized data center
589 whereby many users concurrently access these resources from across the globe,
590 often at a pre-negotiated price point and service level agreement (SLA) [70].
591 Public clouds, which sell hosting services en-masse at competitive costs, are
592 best suited to utilize the economies of scale. The Amazon Elastic Compute
593 Cloud (EC2) is a primary example of a public cloud.

- 594 • **Private** - Private IaaS is where the cloud infrastructure is limited to within
595 a distinct group, set of users, business, or virtual organization. Usually such

596 private cloud infrastructure is also on a private, dedicated network that can
597 either be separate or connected to other services. Data within private clouds
598 are often more secure than those on a public cloud, and as such can be the choice
599 for many users who have sensitive data, or who large-scale users who find better
600 cost, performance, or QoS than what is provided within public clouds.

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

606 Amazon EC2

607 The Amazon Elastic Compute Cloud (EC2) [71], is probably the most popular of
608 cloud infrastructure offerings to date, and is used extensively in the IT industry.
609 EC2 is the central component of Amazon Web Services platform. EC2 allows users
610 to effectively rent virtual machines (called instances), hosted within Amazon's data
611 centers, at a certain price point. Through their advanced UI or a RESTful API, users
612 can start, stop, pause, migrate, and destroy instances exactly as needed to match the
613 required computational tasks at hand.

614 Amazon EC2 predominantly relies on the Xen hypervisor to provide VMs on
615 demand to users, with an equivalent compute unit equal to a 1.7Ghz Intel Xeon
616 processor. However, recent advancements, instance types, and upgrades to EC2 have
617 increased this compute unit's power. Instance reservations have 3 types: On-demand,
618 Reserved, and Spot. With On-demand instances, users pay by the hour for however
619 long the desired instance is running. Users can instead rent reserved instances, where
620 they pay a one-time (discounted) cost based on a pre-determined allocation time.

621 There is also Spot pricing, where VMs are provisioned only when a given spot price is
622 met, which is determined simply based on supply and demand within the EC2 system
623 itself.

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

634 These advanced features, coupled with the first-to-market viability and continual
635 updates have made EC2 the largest cloud infrastructure today. While vendor lock-in
636 is a concern (EC2 is not available for download or replication), other alternatives
637 exist such as Google's Compute Engine [72], the prevalence and support with EC will
638 likely mean its status quo as the public cloud of choice will continue for the foreseeable
639 future.

640 **Nimbus**

641 Nimbus [73, 74] is a set of open source tools that provide a private IaaS cloud com-
642 puting solution. Nimbus is based on the concept of virtual workspaces previously
643 introduced for Globus [74]. A virtual workspace is an abstraction of an execution
644 environment that can be made dynamically available to authorized clients by using
645 well-defined protocols. In this way, it can create customized environments by de-

646 ploying virtual machines (VMs) among remote resources. To such an end, Nimbus
647 provides a web interface called Nimbus Web. Its aim is to provide administrative and
648 user functions in a friendly interface.

649 Within Nimbus, a storage cloud implementation called Cumulus [73] has been
650 tightly integrated with the other central services, although it can also be used stan-
651 dalone. Cumulus is compatible with the Amazon Web Services S3 REST API [75],
652 but extends its capabilities by including features such as quota management. The
653 Nimbus cloud client uses the Jets3t library [76] to interact with Cumulus. However,
654 since it is compatible with S3 REST API, other interfaces like boto [77] or s2cmd [78]
655 can also be used to interact with Nimbus.

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

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

664 **Eucalyptus**

665 Eucalyptus is a product from Eucalyptus Systems [79–81] that developed out of a
666 research project at the University of California, Santa Barbara. Eucalyptus was
667 initially aimed at bringing the cloud computing paradigm of computing to academic
668 super computers and clusters. Eucalyptus provides an Amazon Web Services (AWS)
669 complaint EC2 based web service interface for interacting with the Cloud service.

670 The architecture depicted in Figure 2.4 is based on a two level hierarchy of the

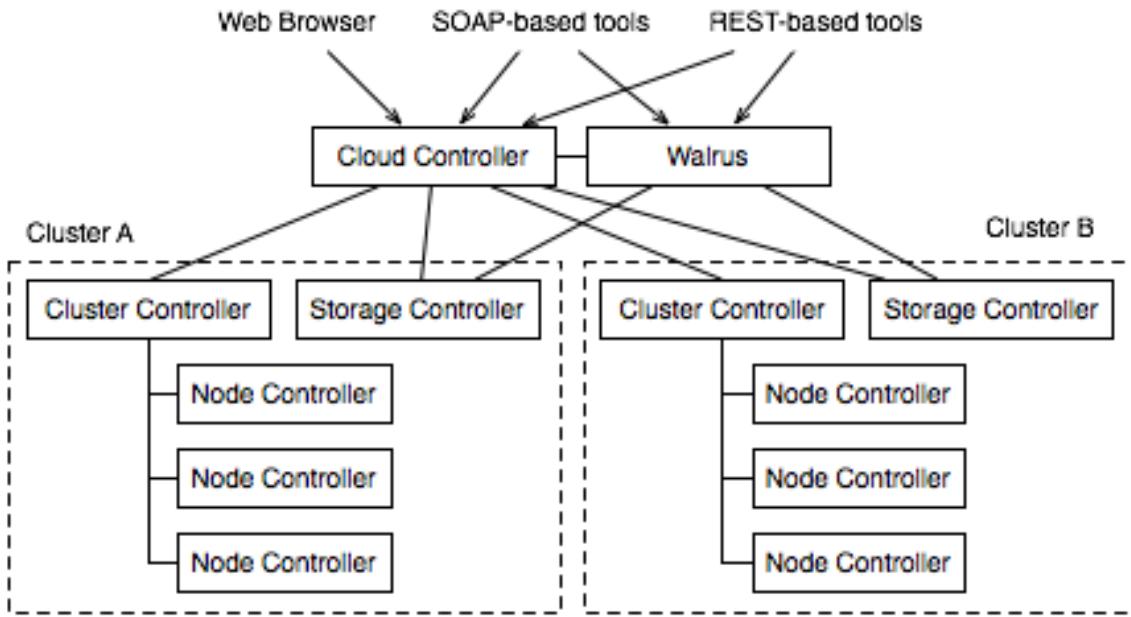


Figure 2.4 Eucalyptus Architecture

671 Cloud controller and the Cluster controller [82]. The Cluster Controller usually man-
 672 ages the nodes within a single cluster and multiple Cluster Controllers can be used
 673 to connect to a single Cloud Controller. The Cloud Controller is responsible for the
 674 resource management, scheduling and accounting aspects of the Cloud.

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

684 While Eucalyptus can be controlled using the EC2 AMI tools, it also provides its

685 own specific tool set: euca2ools. Euca2ools provides support for creating and man-
686 aging keypairs, querying the cloud system, managing VMs, starting and terminating
687 instances, network configuration, and block storage usage. The Eucalyptus system
688 also provides a secure web front-end to allow new users to create and manage account
689 information, view available VMs, and download their security credentials.

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

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

701 **OpenStack**

702 OpenStack [83, 84], another private cloud infrastructure service, was introduced by
703 Rackspace and NASA in July 2010. The project aims to build an open source com-
704 munity spanning technologists, developers, researchers, and industry that allows for
705 the sharing of resources and technologies in order to create a massively scalable and
706 secure cloud infrastructure. In tradition with other open source projects, the entire
707 software is open source.

708 Historically, OpenStack focuses on the development of two aspects of cloud com-
709 putting to address compute and storage aspects with their OpenStack Compute and

710 OpenStack Storage solutions. According to the documentation “OpenStack Com-
711 pute is the internal fabric of the cloud creating and managing large groups of virtual
712 private servers” and “OpenStack Object Storage is software for creating redundant,
713 scalable object storage using clusters of commodity servers to store Terabytes or even
714 petabytes of data.” However, OpenStack as a platform has evolved much more than
715 its original efforts, and has created a wide array of new sub-projects.

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

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

735 With OpenStack Nova’s increased prevalence, the number of auxiliary OpenStack

736 projects has also increased to support Nova. While there are many other recent Open-
737 Stack projects, these listed OpenStack efforts, along with Nova and Swift, represent
738 the common core of a current OpenStack deployment.

739 • **Cinder** for persistent block-level storage mechanisms to support VM instances
740 and elastic block storage.

741 • **Neutron** provides advanced networking and SDN solutions, IP addressing, and
742 VLAN configuration.

743 • **Glance** delivers comprehensive image management, including image discovery,
744 registration, and delivery mechanisms.

745 • **Keystone** identity and authentication service for all OpenStack services.

746 • **Horizon**, a dashboard web-based UI framework, complimentary to the RESTful
747 client API.

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

755 **OpenNebula**

756 OpenNebula [85, 86] is an open-source toolkit which allows administrators to trans-
757 form existing infrastructure into an Infrastructure as a Service (IaaS) cloud with

758 cloud-like interfaces. Figure 2.5 shows the OpenNebula architecture and their main
759 components.

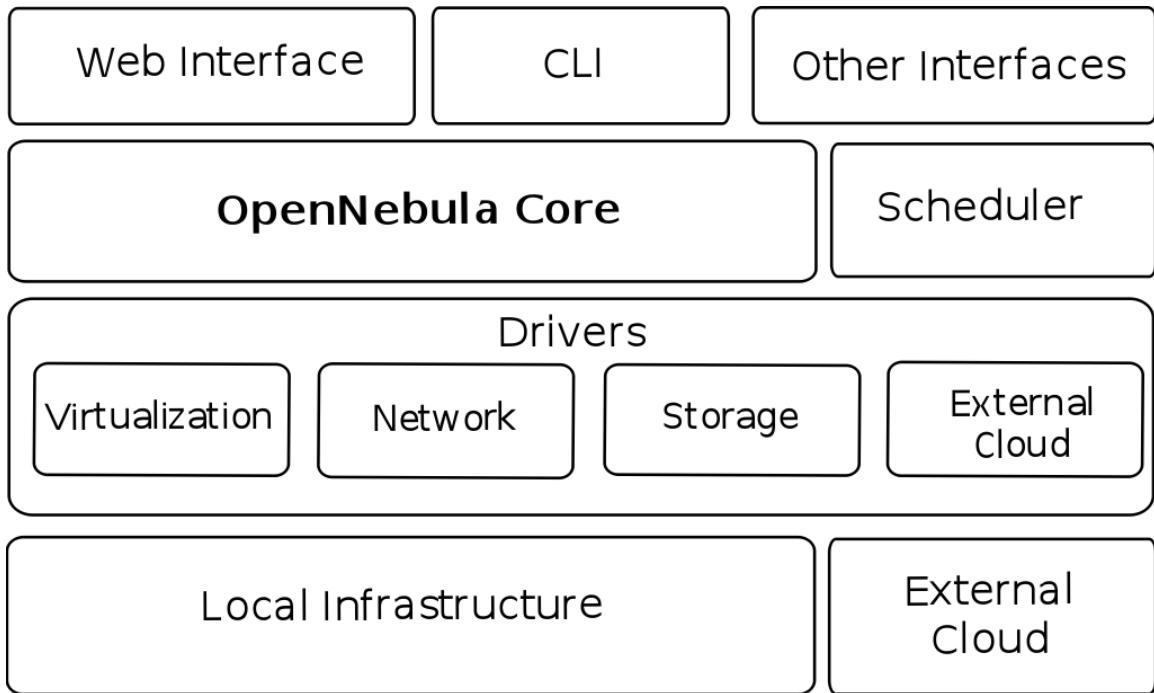


Figure 2.5 OpenNebula Architecture

760 The architecture of OpenNebula has been designed to be flexible and modular to
761 allow its integration with different storage and network infrastructure configurations,
762 as well as hypervisor technologies. Here, the core is a centralized component that
763 manages the virtual machine's (VM) full life cycle, including setting up networks dy-
764 namically for groups of VMs and managing their storage requirements, such as VM
765 disk image deployment or on-the-fly software environment creation. Another impor-
766 tant component is the capacity manager, which governs the functionality provided
767 by the core for scheduling. The default capacity scheduler is a requirement/rank
768 matchmaker. However, it is also possible to develop more complex scheduling poli-
769 cies through a lease model and advance reservations like Haizea [87]. The last main
770 components are the access drivers. They provide an abstraction of the underlying

771 infrastructure to expose the basic functionality of the monitoring, storage and virtu-
772 alization services available in the cluster. Therefore, OpenNebula is not tied to any
773 specific environment and can provide a uniform management layer regardless of the
774 virtualization platform.

775 Additionally, OpenNebula offers management interfaces to integrate the core's
776 functionality within other data center management tools, such as accounting or mon-
777 itoring frameworks. To this end, OpenNebula implements the libvirt API [88], an
778 open interface for VM management, as well as a command line interface (CLI). A
779 subset of this functionality is exposed to external users through a cloud interface.

780 Due to its architecture, OpenNebula is able to adapt to organizations with chang-
781 ing resource needs, including the addition or failure of physical resources [69]. Some
782 essential features to support changing environments are the live migration and the
783 snapshotting of VMs [85]. Furthermore, when the local resources are insufficient,
784 OpenNebula can support a hybrid cloud model by using cloud drivers to interface
785 with external clouds. This lets organizations supplement the local infrastructure
786 with computing capacity from a public cloud to meet peak demands, or implement
787 high availability strategies. OpenNebula includes an EC2 driver, which can submit
788 requests to Amazon EC2 and Eucalyptus [79], as well as an ElasticHosts driver [89].

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

794 **Others**

795 Other cloud specific projects exist, such as In-VIGO [90], Cluster-on-Demand [91],
796 and VMWare's own proprietary vCloud Air [92]. Each effort provides its own in-
797 terpretation of private cloud services within a data center, often with the ability to
798 interplay with public cloud offerings such as Amazon's EC2. Docker [49] also looks
799 to provide IaaS capabilities with specialized and easily configurable containers, based
800 on LXC and libcontainer solutions. While it is still to be determined how Docker
801 and containers will change the private IaaS landscape, they do provide similar func-
802 tionality for Linux users without some of the complexities of traditional virtualized
803 IaaS.

804 **2.2.2 Virtual Clusters**

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

808 Let us consider a typical usage for a Cloud data center that is used in part to
809 provide computational power for the Large Hadron Collider at CERN [93], a global
810 collaboration from more than 2000 scientists of 182 institutes in 38 nations. Such a
811 system would have a small number of experiments to run. Each experiment would
812 require a very large number of jobs to complete the computation needed for the
813 analysis. Examples of such experiments are the ATLAS [94] and CMS [95] projects,
814 that (combined) require Petaflops of computing power on a daily basis. Each job of an
815 experiment is unique, but the application runs are often the same. Therefore, virtual
816 machines are deployed to execute incoming jobs. There is a file server which provides
817 virtual machine templates. All typical jobs are preconfigured in virtual machine

818 templates. When a job arrives at the head node of the cluster, a correspondent
 819 virtual machine is dynamically started on a certain compute node within the cluster
 820 to execute the job. While the LHC project and CERN's cloud effort is a formidable
 821 one, it only covers pleasingly parallel HTC workloads, and often times HPC and big
 822 data workloads can be equally complex yet drastically different.

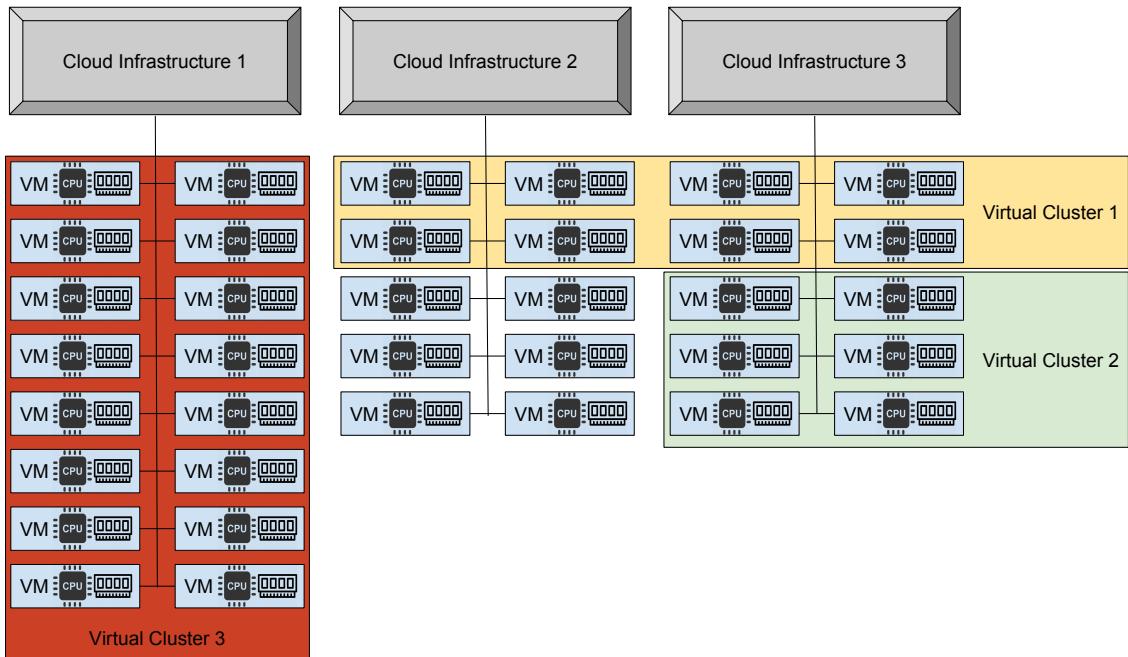


Figure 2.6 Virtual Clusters on Cloud Infrastructure

823 Cluster computing has become one of the core tools in distributed systems for
 824 use in parallel computation. Cluster computing revolves around the desire to get
 825 more computing power and better reliability by utilizing many computers together
 826 across a network to achieve larger computational tasks. Clusters have manifested
 827 themselves in many different ways, ranging from Beowulf clusters [6], which run using
 828 commodity PCs to some of the TOP500 [8] supercomputing systems today. Virtual
 829 clusters represent the growing need of users to organize computational resources in an
 830 environment specific to their tasks at hand effectively, instead of sharing a common

831 architecture across many users. With the advent of modern virtualization, virtual
832 clusters are deployed across a set of VMs in order to gain relative isolation and
833 flexibility between disjoint virtual clusters. Virtual clusters, or a set of multiple cluster
834 computing deployments on a single, larger physical cluster infrastructure, often have
835 the following properties and attributes [52]:

- 836 ● Resources allocation based on a VM unit
- 837 ● Clusters built of many VMs together, or by provisioning physical nodes
- 838 ● Leverage local infrastructure management tools to provide a middleware solu-
839 tion for virtual clusters
 - 840 – Implementations could be a cloud IaaS such as OpenStack
 - 841 – Some instances use a queueing system such as PBS
- 842 ● User experience based on virtual cluster management, not single VM manage-
843 ment
- 844 ● Consolidates functionality on a smaller resource platform using multiple VMs
- 845 ● Can provide fault tolerance through VM migration and management
- 846 ● Can utilize dynamic scaling through the addition or deletion of VMs from the
847 virtual cluster
- 848 ● Connection to back-end storage solution to provide virtual persistent storage

849 Given the properties, virtual clusters can take on many forms; however, a very
850 simplified set of virtual clusters across cloud infrastructure is provided as a represen-
851 tation in Figure 2.6. This could lead to the simple provisioning of multiple disjoint

852 OSs on a single physical resource. Virtual clusters generally have the ability to pro-
853 vide and manage their own user environment and tuned internal middleware. Virtual
854 clusters may enable the separation of multiple tasks into separate VMs, which still
855 in fact run on the same or similar underlying physical resources, effectively providing
856 task isolation. Virtual clusters can be deployed to be persistent, stored, shared, or
857 re-provisioned on demand. The size of a virtual cluster could potentially expand and
858 contrast relative to the necessary resource requirements, taking advantage of elastic-
859 ity found with virtualization. Furthermore, VM migration may enable fault tolerance
860 in the event of physical machine errors if properly managed.

861 With virtual clusters, the capability to deploy custom environments quickly be-
862 comes critical. As such, efforts have been put forth to configure and create VM
863 images on-demand. This includes custom efforts with configuration engines such as
864 CfEngine, Chef, Ansible, and others. Within FutureGrid, an image management sys-
865 tem was defined to provide preconfigured VM images for cloud infrastructure using
866 the BCFG2 engine [96].

867 Initially, virtual clusters were proposed for the use of Grid communities [32].
868 Specifically, Foster et al look to provide commodity clusters to various Virtual Or-
869 ganizations [97], whereby grid services can instantiate and deploy VMs. This design
870 and implementation was further refined through the use of metadata and contextu-
871 alization using appliances [98]. Some of these ideas have even come to take shape in
872 larger scale supercomputing deployments, such as with SDSC Comet's virtual cluster
873 availability [99].

874 Virtual clusters require orchestration services to be able to organize, deploy, man-
875 age, and re-play the desired user environment, and there have been a number of
876 efforts to bring this orchestration to utility. One efforts within FutureGrid is with
877 the experiment management design [43], which attempts to define how resources are

878 connected to and monitored, as well as how VMs are stored in a repository and pro-
879 visioned across multiple heterogeneous resources. This effort moved forward with
880 Cloudmesh [45], which provides a simple client interface to access multiple cloud
881 resources with a command-line shell interface.

882 Another virtual cluster orchestration, named OpenStack Heat [44], has developed
883 within the OpenStack community. Heat provides a method by which an individ-
884 ual or group can deploy "stacks", which could essentially be virtual clusters, using
885 OpenStack infrastructure. Specifically, Heat provides a human readable and machine-
886 accessible template for specifying environments and requirements, as well as a REST-
887 ful API. In submitting a Heat orchestration template to the API, heat will interpret
888 and build the designed custom environment within a given OpenStack cloud deploy-
889 ment. Kubernetes [100], a related effort, is a infrastructure orchestration framework
890 for managing containerized applications within Docker.

891 2.2.3 The FutureGrid Project

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

902 The FutureGrid features a unique WAN network structure that lent itself to a

multitude of experiments specifically designed for evaluating middleware technologies and experiment management services. This network can be dedicated to conduct experiments in isolation using a network impairment device for introducing a variety of predetermined network conditions. Figure 2.7 depicts the geographically distributed resources that are outlined in Table 2.1 in more detail. All network links within FutureGrid are dedicated 10GbE links with the exception of a shared 10GbE link to TACC over the TeraGrid [101, 102] network, enabling high-speed data management and transfer between each partner site within FutureGrid.



Figure 2.7 FutureGrid Participants, Network, and Resources

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

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

917 accommodate the needs of an experiment management system.

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

924 2.3 High Performance Computing

925 2.3.1 Brief History of Supercomputing

926 Supercomputing can date back to some of the forefront of computing itself, especially
927 if we consider the ENIAC [103], the first Turing Complete general purpose digital
928 computer, also as the first supercomputer. ENIAC was first deployed to calculate
929 artillery firing tables, but was later used during the Second World War for helping the
930 Manhattan project's thermonuclear calculations and later dedicated to the University
931 of Pennsylvania after the war.

932 The first properly termed supercomputer was the Control Data Corporation's 6600
933 mainframe [104], first delivered to CERN in 1965. The CDC 6600 was notably faster
934 than the IBM counterparts, and the first deployments were able to perform on the
935 order of 1 MFLOP. Interestingly, the CPU design that came from Seymour Cray's
936 CDC 6600 took advantage of a simplified yet fast CPU design with silicon-based
937 transistors, which founded the basis of the RISC processor architecture.

938 Cray's efforts eventually lead him to start his own company, and in 1975 released
939 the Cray 1 system [105]. The Cray 1 took the powerful aspects of vector processing
940 and memory pipelining from the STAR architecture (developed later by CDC) and in-
941 troduced scalar performance through splitting vectors and instruction chaining. This
942 resulted in an overall performance of around 250 MFLOPS at peak, but realistically
943 closer to 100 MFLOPS for general applications. The Cray 1 system also helped push
944 forward the integrated circuit design, which was finally performant enough to be used.
945 Interestingly enough, the Cray 1 also required an entirely new Freon-based coolant
946 system.

947 The Cray 1 system gave way to the Cray X-MP and Y-MP in the mid 1980s.
948 These machines were shared-memory vector processors, with two processors in the

949 X-MP and up to 8 processors for the later Y-MP systems. These first shared-memory
950 systems were aided by increased memory speeds. The X-MP machine was capable
951 of 200 MFLOPS sustained and 400 MFLOPS peak performance, whereas the Y-MP
952 variants were capable of over 2 GFLOPS.

953 Concurrently, during the 1980s, the advent of distributed memory architectures for
954 supercomputing were also starting to emerge. Specifically work on Caltech's Cosmic
955 Cube, also known as a Hypercube, by Seitz and Fox [19, 106], started the movement of
956 concurrent or parallel computing. The Cosmic Cube leveraged new VLSI techniques
957 and assembled Intel 8086/87 processors together with a novel hypercube interconnect
958 which required no switching, creating one of the first truly parallel computers. SIMD
959 programming and computation was done through a novel message passing architec-
960 ture, instead of shared variables. This design was first commercialized with Intel's
961 iPSC, and later contributed directly to designs in the Intel Paragon, ASCI Red, and
962 Cray T3D/E systems.

963 While concurrent and parallel processor supercomputers continued into the 90s
964 with aforementioned hypercube designs and the IBM Thinking Machines [107], a new
965 commodity-based strategy emerged with Becker and Sterling's Beowulf clusters [6].
966 Effectively, a Beowulf cluster is simply a cluster of commodity x86 machines linked
967 together with a simplified LAN network. Beowulf clusters often (but not always)
968 run Linux OS and leverage Ethernet solutions, and are programmed using a message
969 passing construct such as MPI [18]. This allows for the building of massively parallel
970 systems with relatively low cost and investment. Many specialized clusters today
971 still utilize commodity x86 hardware and Linux OSs similar to the original Beowulf
972 systems.

973 Concurrency and parallel computation on supercomputing resources has only
974 flourished since. This has been even more pronounced as CPU clock frequencies

975 stabilized and multi-core architectures took hold, driving the need for concurrency
976 not only at an increased rate for supercomputing, but also even for commodity sys-
977 tems. While commodity single-CPU, multi-core systems often look towards exploiting
978 shared memory parallelism, distributed memory architectures have become a way of
979 life for high performance computing.

980 2.3.2 Distributed Memory Computation

981 Abstractly, distributed memory architectures consist of multiple instances of a pro-
982 cessor, memory, and an interconnect, which allows each instance to perform inde-
983 pendent computations and communicate over the interconnect. These interconnects
984 could be built using point-to-point, or through more complex and advanced switching
985 hardware, building a larger topology. While a simple example could involve several
986 commodity PCs connected through an Ethernet switch, this model scales to the latest
987 supercomputing resources of today with millions of cores [108].

988 Programming such distributed memory systems is a nontrivial task that the
989 greater HPC community has been wrangling for years. In the early days, this took
990 the form of either the Message Passing Interface (MPI) [18] or PVM [109]; however,
991 MPI has been far more successful and dominates the market for distributed memory
992 parallel computation. MPI is a standardized message passing system, which defines
993 the semantics and syntax for writing parallel programs in C, C++, or Fortran. MPI
994 has even been implemented in other languages such as Java [110]. As MPI is a stan-
995 dardization, there are many implementations that exist, including OpenMPI [111],
996 MPICH [112], and MVAPICH [113], to name a few. Many MPI-enabled applications
997 have been shown to be, with proper and careful design, the most efficient way to run
998 a parallel application across a large subset of tightly coupled distributed resources,
999 and can often represent the status-quo for HPC applications today.

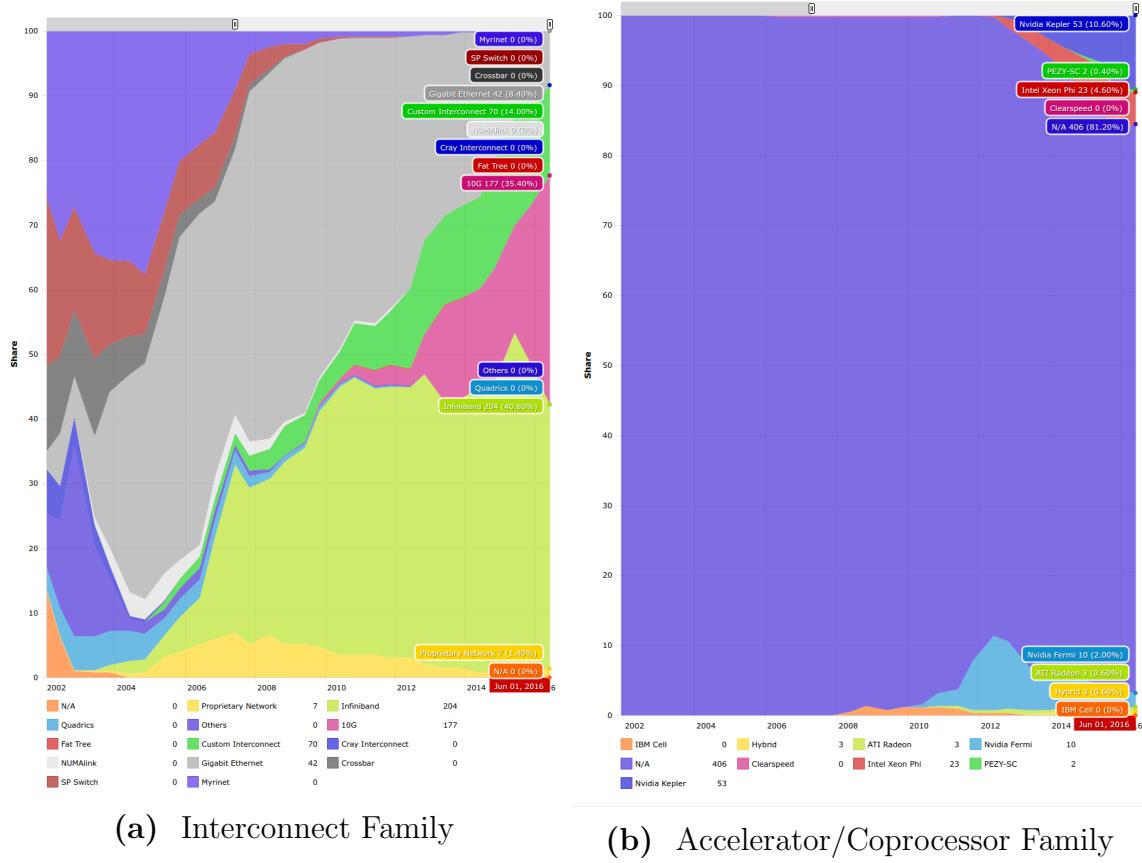


Figure 2.8 Top 500 Development over time from 2002 to 2016 [8]

More recently, the MPI programming model has been modified or joined with other models. This change or deviation largely revolves around the hardware that has changed within HPC resources themselves to meet the need for more computational power. This includes hybrid MPI + OpenMP models which become useful as multi-core and many-core technologies becomes increasingly available [114], or the MPI+CUDA model which interleaves message passing with GPU utilization [115]. As some of the latest supercomputing resources have been deployed specifically with Nvidia GPUs for GPGPU programming, such as the ORNL’s Titan system [13], the need for distributed memory computation with GPUs has increased.

To get an idea of some of the hardware advances over the past few years, it is useful to examine the Top 500 [8], a comprehensive list of the top 500 supercom-

1011 puters known and their key characteristics. Looking at the past decade in HPC, we
1012 can see some trends emerge within hardware. Specifically, looking at Interconnect
1013 and Coprocessor architectures in Figure 2.8, we see a notable jump in the number
1014 of deployed system that are using both InfiniBand and GPUs in the past decade.
1015 In particular, InfiniBand usage has increased to roughly 40% of the total number of
1016 deployed systems and remained somewhat stable as an interconnect family. Concur-
1017 rently, the use of accelerators has increased from only 1 in 500 systems a decade ago,
1018 to almost a 20% of the top 500 systems that are using coprocessors. Within that fac-
1019 tor, the majority of such accelerator-equipped systems have been using GPUs, with
1020 a concurrent increase in the Intel Xeon Phi coprocessor as well. While these do not
1021 represent all of supercomputing nor exclusively the most high end of systems, they do
1022 represent advanced hardware that has been increasingly common in the last decade,
1023 yet relatively underutilized in comparison within cloud infrastructure. As such, much
1024 of the effort in this dissertation focuses on, but is not limited to, these two technology
1025 families.

1026 2.3.3 Exascale

1027 As the forefront of supercomputing moves beyond the latest petaflop machines of the
1028 past few years [13], the HPC community is setting their sights on the next significant
1029 milestone: Exascale. Exascale computing refers broadly to performing roughly one
1030 exaFLOPS, or 10^{18} floating point operations per second. However, exascale itself is
1031 far more than just a theoretical FLOPS goal; instead, it is a set of new comput-
1032 ing advancements and challenges that requires reaching computational power at such
1033 magnitude. While FLOPs are often used as the ubiquitous yard stick for supercom-
1034 puting with the LINPACK benchmark [116], other efforts have taken hold to classify
1035 systems under a different set of parameters [117, 118], with the loose understanding

1036 that these may incorporate a richer application set destined for exascale systems. This
1037 could include, for instance, integer calculations at a similar scale to satisfy defence
1038 and intelligence perspectives, or graph processing with billions of vertices.

1039 With exascale, there are a number of barriers that exist with current technolo-
1040 gies that must be overcome to reach exascale. The exascale Computing Study [39]
1041 specifically lists 4 major focal areas:

1042 1. Energy and Power Challenge

1043 • Describes the physical difficulties in providing the amount of power needed
1044 to drive a sufficiently large exascale system. The US DOE estimates the
1045 maximum power envelope for a deployed first exascale system to be within
1046 20-40MW. Extrapolating current technology power utilization shows an
1047 order of magnitude more energy utilization than the specificity power en-
1048 envelop. As such, new architectures and conversation techniques will need
1049 to be investigated.

1050 2. Memory and Storage Challenge

1051 • This challenge illustrates the problem that has grown in relation to the
1052 memory wall, defined by the exponential difference between processor and
1053 memory performance, as well as the storage capacity limits to support
1054 calculations at the level of performance necessary. This challenge incorpo-
1055 rates not only main memory limitations, but also tertiary storage issues as
1056 well.

1057 3. Concurrency and Storage Challenge

1058 • This challenge is born from the recent limit in CPU clock rates as a way
1059 to gain performance. Instead, performance must be gained through paral-

1060 lelism. The depth of this challenge is especially profound when we consider
1061 parallelism on the order of millions, if not billions, of threads.

1062 4. Resiliency Challenge

1063 • The resiliency challenge defines the necessity of computation to recover
1064 and continue in the event of a fault or fluctuation. As parallelism and
1065 the number of individualized components substantially increases in a path
1066 towards exascale, the mean time to failure of any given component also
1067 increases.

1068 Current exascale efforts are as wide as they are varying, not only with concepts,
1069 architectures, and runtime systems, but also with deployment plans and expectations
1070 between future deployments. Of particular interest in current exascale research is
1071 in Operating System and runtime (OS/R) developments to support new extreme-
1072 scale applications in an efficient manner. Two examples of novel OS approaches are
1073 the Hobbes project [119] and the ARGO Exascale Operating System [120]. These OS
1074 efforts, along with novel programming models for exascale such as ParalleX [121] look
1075 to fundamentally change the relationship between HPC hardware architectures and
1076 the libraries and applications to be leveraged on such future exascale deployments.

1077 It is possible that virtualization itself may have an impact in OS and runtime ser-
1078 vices in exascale [119]. While some of the work herein may tangentially be of utility
1079 to such efforts, the immediate goal of this dissertation is not to investigate the appli-
1080 cability of virtualization for exascale systems, but rather to enable the diversification
1081 of HPC towards cloud infrastructure.

1082 **Chapter 3**

1083 **Analysis of Virtualization**

1084 **Technologies for High Performance**

1085 **Computing Environments**

1086 **3.1 Abstract**

1087 As Cloud computing emerges as a dominant paradigm in distributed systems, it is
1088 important to fully understand the underlying technologies that make Clouds possible.
1089 One technology, and perhaps the most important, is virtualization. Recently virtual-
1090 ization, through the use of hypervisors, has become widely used and well understood
1091 by many. However, there are a large spread of different hypervisors, each with their
1092 own advantages and disadvantages. This chapter provides an in-depth analysis of
1093 some of today's commonly accepted virtualization technologies from feature com-
1094 parison to performance analysis, focusing on the applicability to High Performance
1095 Computing environments using FutureGrid resources. The results indicate virtualiza-
1096 tion sometimes introduces slight performance impacts depending on the hypervisor

1097 type, however the benefits of such technologies are profound and not all virtualization
1098 technologies are equal.

1099 **3.2 Introduction**

1100 Cloud computing [58] is one of the most explosively expanding technologies in the
1101 computing industry today. A Cloud computing implementation typically enables
1102 users to migrate their data and computation to a remote location with some varying
1103 impact on system performance [64]. This provides a number of benefits which could
1104 not otherwise be achieved.

1105 Such benefits include:

- 1106 • *Scalability* - Clouds are designed to deliver as much computing power as any
1107 user needs. While in practice the underlying infrastructure is not infinite, the
1108 cloud resources are projected to ease the developer's dependence on any specific
1109 hardware.
- 1110 • *Quality of Service (QoS)* - Unlike standard data centers and advanced com-
1111 puting resources, a well-designed Cloud can project a much higher QoS than
1112 traditionally possible. This is due to the lack of dependence on specific hard-
1113 ware, so any physical machine failures can be mitigated without the prerequisite
1114 user awareness.
- 1115 • *Customization* - Within a Cloud, the user can utilize customized tools and
1116 services to meet their needs. This can be to utilize the latest library, toolkit, or
1117 to support legacy code within new infrastructure.
- 1118 • *Cost Effectiveness* - Users finds only the hardware required for each project.
1119 This reduces the risk for institutions potentially want build a scalable system,

1120 thus providing greater flexibility, since the user is only paying for needed in-
1121 frastructure while maintaining the option to increase services as needed in the
1122 future.

- 1123 • *Simplified Access Interfaces* - Whether using a specific application, a set of
1124 tools or Web services, Clouds provide access to a potentially vast amount of
1125 computing resources in an easy and user-centric way.

1126 While Cloud computing has been driven from the start predominantly by the in-
1127 dustry through Amazon [71], Google [122] and Microsoft [123], a shift is also occurring
1128 within the academic setting as well. Due to the many benefits, Cloud computing is
1129 becoming immersed in the area of High Performance Computing (HPC), specifically
1130 with the deployment of scientific clouds [124] and virtualized clusters [32].

1131 There are a number of underlying technologies, services, and infrastructure-level
1132 configurations that make Cloud computing possible. One of the most important
1133 technologies is virtualization. Virtualization, in its simplest form, is a mechanism to
1134 abstract the hardware and system resources from a given Operating System. This is
1135 typically performed within a Cloud environment across a large set of servers using a
1136 Hypervisor or Virtual Machine Monitor (VMM), which lies in between the hardware
1137 and the OS. From the hypervisor, one or more virtualized OSs can be started concur-
1138 rently, leading to one of the key advantages of Cloud computing. This, along with the
1139 advent of multi-core processors, allows for a consolidation of resources within any data
1140 center. From the hypervisor level, Cloud computing middleware is deployed atop the
1141 virtualization technologies to exploit this capability to its maximum potential while
1142 still maintaining a given QoS and utility to users.

1143 The rest of this chapter is as follows: First, we look at what virtualization is,
1144 and what current technologies currently exist within the mainstream market. Next

1145 we discuss previous work related to virtualization and take an in-depth look at the
1146 features provided by each hypervisor. We follow this by outlining an experimental
1147 setup to evaluate a set of today's hypervisors on a novel Cloud test-bed architecture.
1148 Then, we look at performance benchmarks which help explain the utility of each
1149 hypervisor and the feasibility within an HPC environment. We conclude with our
1150 final thoughts and recommendations for using virtualization in Clouds for HPC.

1151 **3.3 Related Research**

1152 While the use of virtualization technologies has increased dramatically in the past few
1153 years, virtualization is not specific to the recent advent of Cloud computing. IBM
1154 originally pioneered the concept of virtualization in the 1960's with the M44/44X
1155 systems [125]. It has only recently been reintroduced for general use on x86 plat-
1156 forms. Today there are a number of public Clouds that offer IaaS through the use
1157 of virtualization technologies. The Amazon Elastic Compute Cloud (EC2) [126] is
1158 probably the most popular Cloud and is used extensively in the IT industry to this
1159 day. Nimbus [127] and Eucalyptus [79] are popular private IaaS platforms in both
1160 the scientific and industrial communities. Nimbus, originating from the concept of
1161 deploying virtual workspaces on top of existing Grid infrastructure using Globus, has
1162 pioneered scientific Clouds since its inception. Eucalyptus has historically focused
1163 on providing an exact EC2 environment as a private cloud to enable users to build
1164 an EC2-like cloud using their own internal resources. Other scientific Cloud specific
1165 projects exist such as OpenNebula [128], In-VIGO [129], and Cluster-on-Demand [91],
1166 all of which leverage one or more hypervisors to provide computing infrastructure on
1167 demand. In recent history, OpenStack [130] has also come to light from a joint col-
1168 laboration between NASA and Rackspace which also provide compute and storage

1169 resources in the form of a Cloud.

1170 While there are currently a number of virtualization technologies available today,
1171 the virtualization technique of choice for most open platforms over the past 5 years has
1172 typically been the Xen hypervisor [53]. However more recently VMWare ESX [131]
1173 ¹, Oracle VirtualBox [132] and the Kernel-based Virtual Machine (KVM) [57] are
1174 becoming more commonplace. As these look to be the most popular and feature-
1175 rich of all virtualization technologies, we look to evaluate all four to the fullest extent
1176 possible. There are however, numerous other virtualization technologies also available,
1177 including Microsoft's Hyper-V [133], Parallels Virtuozzo [134], QEMU [135], OpenVZ
1178 [136], Oracle VM [137], and many others. However, these virtualization technologies
1179 have yet to seen widespread deployment within the HPC community, at least in their
1180 current form, so they have been placed outside the scope of this work.

1181 In recent history there have actually been a number of comparisons related to
1182 virtualization technologies and Clouds. The first performance analysis of various hy-
1183 pervisors started with, unsurprisingly, the hypervisor vendors themselves. VMWare
1184 has happy to put out its own take on performance in [138], as well as the original
1185 Xen article [53] which compares Xen, XenoLinux, and VMWare across a number of
1186 SPEC and normalized benchmarks, resulting in a conflict between both works. From
1187 here, a number of more unbiased reports originated, concentrating on server consol-
1188 idation and web application performance [131, 139, 140] with fruitful yet sometimes
1189 incompatible results. A feature base survey on virtualization technologies [141] also
1190 illustrates the wide variety of hypervisors that currently exist. Furthermore, there
1191 has been some investigation into the performance within HPC, specifically with In-
1192 finiBand performance of Xen [142] and rather recently with a detailed look at the
1193 feasibility of the Amazon Elastic Compute cloud for HPC applications [48], however

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

1194 both works concentrate only on a single deployment rather than a true comparison
1195 of technologies.

1196 As these underlying hypervisor and virtualization implementations have evolved
1197 rapidly in recent years along with virtualization support directly on standard x86
1198 hardware, it is necessary to carefully and accurately evaluate the performance impli-
1199 cations of each system. Hence, we conducted an investigation of several virtualization
1200 technologies, namely Xen, KVM, VirtualBox, and in part VMWare. Each hypervisor
1201 is compared alongside one another with base-metal as a control and (with the exception
1202 of VMWare) run through a number of High Performance benchmarking tools.

1203 3.4 Feature Comparison

1204 With the wide array of potential choices of virtualization technologies available, its
1205 often difficult for potential users to identify which platform is best suited for their
1206 needs. In order to simplify this task, we provide a detailed comparison chart between
1207 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

1208 The first point of investigation is the virtualization method of each VM. Each

1209 hypervisor supports full virtualization, which is now common practice within most
1210 x86 virtualization deployments today. Xen, originating as a para-virtualized VMM,
1211 still supports both types, however full virtualization is often preferred as it does
1212 not require the manipulation of the guest kernel in any way. From the Host and
1213 Guest CPU lists, we see that x86 and, more specifically, x86-64/amd64 guests are all
1214 universally supported. Xen and KVM both support Itanium-64 architectures for full
1215 virtualization (due to both hypervisors dependency on QEMU), and KVM also claims
1216 support for some recent PowerPC architectures. However, we concern ourselves only
1217 with x86-64 features and performance, as other architectures are out of the scope of
1218 this manuscript. Of the x86-64 platforms, KVM is the only hypervisor to require
1219 either Intel VT-X or AMD-V instruction sets in order to operate. VirtualBox and
1220 VMWare have internal mechanisms to provide full virtualization even without the
1221 virtualization instruction sets, and Xen can default back to para-virtualized guests.

1222 Next, we consider the host environments for each system. As Linux is the pri-
1223 mary OS type of choice within HPC deployments, its key that all hypervisors sup-
1224 port Linux as a guest OS, and also as a host OS. As VMWare ESX is meant to be
1225 a virtualization-only platform, it is built upon a specially configured Linux/UNIX
1226 proprietary OS specific to its needs. All other hypervisors support Linux as a host
1227 OS, with VirtualBox also supporting Windows, as it was traditionally targeted for
1228 desktop-based virtualization. However, as each hypervisor uses VT-X or AMD-V in-
1229 structions, each can support any modern OS targeted for x86 platforms, including all
1230 variants of Linux, Windows, and UNIX.

1231 While most hypervisors have desirable host and guest OS support, hardware sup-
1232 port within a guest environment varies drastically. Within the HPC environment,
1233 virtual CPU (vCPU) and maximum VM memory are critical aspects to choosing the
1234 right virtualization technology. In this case, Xen is the first choice as it supports up

1235 to 128 vCPUs and can address 4TB of main memory in 64-bit modes, more than
1236 any other. VirtualBox, on the other hand, supports only 32 vCPUs and 16GB of
1237 addressable RAM per guest OS, which may lead to problems when looking to deploy
1238 it on large multicore systems. KVM also faces an issue with the number of vCPU
1239 supported limited to 16, recent reports indicate it is only a soft limit [143], so deploy-
1240 ing KVM in an SMP environment may not be a significant hurdle. Furthermore, all
1241 hypervisors provide some 3D acceleration support (at least for OpenGL) and support
1242 live migration across homogeneous nodes, each with varying levels of success.

1243 Another vital juxtaposition of these virtualization technologies is the license agree-
1244 ments for its applicability within HPC deployments. Xen, KVM, and VirtualBox are
1245 provided for free under the GNU Public License (GPL) version 2, so they are open
1246 to use and modification by anyone within the community, a key feature for many
1247 potential users. While VirtualBox is under GPL, it has recently also offered with
1248 additional features under a more proprietary license dictated by Oracle since its ac-
1249 quirement from Sun last year. VMWare, on the other hand, is completely proprietary
1250 with an extremely limited licensing scheme that even prevents the authors from will-
1251 fully publishing any performance benchmark data without specific and prior approval.
1252 As such, we have neglected VMWare form the remainder of this chapter. Whether
1253 going with a proprietary or open source hypervisor, support can be acquired (usually
1254 for an additional cost) with ease from each option.

1255 3.4.1 Usability

1256 While side by side feature comparison may provide crucial information about a poten-
1257 tial user's choice of hypervisor, that may also be interested in its ease of installation
1258 and use. We will take a look at each hypervisor from two user perspectives, a systems
1259 administrator and normal VM user.

1260 One of the first things on any system administrator's mind on choosing a hypervi-
1261 sor is the installation. For all of these hypervisors, installation is relatively painless.
1262 For the FutureGrid support group, KVM and VirualBox are the easiest of the all
1263 tested hypervisors to install, as there are a number of supported packages available
1264 and installation only requires the addition of one or more kernel modules and the sup-
1265 port software. Xen, while still supported in binary form by many Linux distributions,
1266 is actually much more complicated. This is because Xen requires a full modification
1267 to the kernel itself, not just a module. Loading a new kernel into the boot process
1268 which may complicate patching and updating later in the system's maintenance cycle.
1269 VMWare ESX, on the other hand, is entirely separate from most other installations.
1270 As previously noted, ESX is actually a hypervisor and custom UNIX host OS com-
1271 bined, so installation of ESX is likewise to installing any other OS from scratch. This
1272 may be either desirable or adverse, depending on the system administrator's usage of
1273 the systems and VMWare's ability to provide a secure and patched environment.

1274 While system administrators may be concerned with installation and maintenance,
1275 VM users and Cloud developers are more concerned with daily usage. The first thing
1276 to note about all of such virtualiation technologies is they are supported (to some
1277 extent) by the libvirt API [144]. Libvirt is commonly used by many of today's IaaS
1278 Cloud offerings, including Nimbus, Eucalyptus, OpenNebula and OpenStack. As
1279 such, the choice of hypervisor for Cloud developer's is less of an issue, so long as
1280 the hypervisor supports the features they desire. For individual command line usage
1281 of each tool, it varies quite a bit more. Xen does provide their own set of tools for
1282 controlling and monitoring guests, and seem to work relatively well but do incur a
1283 slight learning curve. KVM also provides its own CLI interface, and while it is often
1284 considered less cumbersome it provides less advanced features directly to users, such as
1285 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) [145] 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 [43]. 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

1323 transition from one environment to the other. These interests and research objectives
1324 make the FutureGrid project the perfect match for this work. Furthermore, we expect
1325 that the results gleaned from this chapter will have a direct impact on the FutureGrid
1326 deployment itself.

1327 3.5.2 Experimental Environment

1328 Currently, one of FutureGrid's latest resources is the *India* system, a 256 CPU IBM
1329 iDataPlex machine consisting of 1024 cores, 2048 GB of ram, and 335 TB of storage
1330 within the Indiana University Data Center. In specific, each compute node of India
1331 has two Intel Xeon 5570 quad core CPUs running at 2.93Ghz, 24GBs of Ram, and a
1332 QDR InfiniBand connection. A total of four nodes were allocated directly from India
1333 for these experiments. All were loaded with a fresh installation of Red Hat Enterprise
1334 Linux server 5.5 x86_64 with the 2.6.18-194.8.1.el5 kernel patched. Three of the four
1335 nodes were installed with different hypervisors; Xen version 3.1, KVM (build 83), and
1336 VirtualBox 3.2.10, and the forth node was left as-is to act as a control for bare-metal
1337 native performance.

1338 Each guest virtual machine was also built using Red Hat EL server 5.5 running
1339 an unmodified kernel using full virtualization techniques. All tests were conducted
1340 giving the guest VM 8 cores and 16GB of ram to properly span a compute node.
1341 Each benchmark was run a total of 20 times, with the results averaged to produce
1342 consistent results, unless indicated otherwise.

1343 3.5.3 Benchmarking Setup

1344 As this chapter aims to objectively evaluate each virtualization technology from a
1345 side-by-side comparison as well as from a performance standpoint, the selection of

1346 benchmarking applications is critical.

1347 The performance comparison of each virtual machine is based on two well known
1348 industry standard performance benchmark suites; HPCC and SPEC. These two
1349 benchmark environments are recognized for their standardized reproducible results in
1350 the HPC communit, and the National Science Foundation (NSF), Department of En-
1351 ergy (DOE), and DARPA are all sponsors of the HPCC benchmarks. The following
1352 benchmarks provide a means to stress and compare processor, memory, inter-process
1353 communication, network, and overall performance and throughput of a system. These
1354 benchmarks were selected due to their importance to the HPC community sinse they
1355 are often directly correlated with overall application performance [146].

1356 **HPCC Benchmarks**

1357 The HPCC Benchmarks [147, 148] are an industry standard for performing bench-
1358 marks for HPC systems. The benchmarks are aimed at testing the system on multiple
1359 levels to test their performance. It consists of 7 different tests:

- 1360 ● *HPL* - The Linpack TPP benchmark measures the floating point rate of exe-
1361 cution for solving a linear system of equations. This benchmark is perhaps the
1362 most important benchmark within HPC today, as it is the basis of evaluation
1363 for the Top 500 list [8].
- 1364 ● *DGEMM* - Measures the floating point rate of execution of double precision real
1365 matrix-matrix multiplication.
- 1366 ● *STREAM* - A simple synthetic benchmark program that measures sustainable
1367 memory bandwidth (in GB/s) and the corresponding computation rate for sim-
1368 ple vector kernel.

- 1369 ● *PTRANS* - Parallel matrix transpose exercises the communications where pairs
1370 of processors communicate with each other simultaneously. It is a useful test of
1371 the total communications capacity of the network.
- 1372 ● *RandomAccess* - Measures the rate of integer random updates of memory (GUPS).
- 1373 ● *FFT* - Measures the floating point rate of execution of double precision complex
1374 one-dimensional Discrete Fourier Transform (DFT).
- 1375 ● *Communication bandwidth and latency* - A set of tests to measure latency and
1376 bandwidth of a number of simultaneous communication patterns; based on b_eff
1377 (effective bandwidth benchmark).

1378 This benchmark suite uses each test to stress test the performance on multiple
1379 aspects of the system. It also provides reproducible results which can be verified by
1380 other vendors. This benchmark is used to create the Top 500 list [8] which is the list
1381 of the current top supercomputers in the world. The results that are obtained from
1382 these benchmarks provide an unbiased performance analysis of the hypervisors. Our
1383 results provide insight on inter-node PingPong bandwidth, PingPong latency, and
1384 FFT calculation performance.

1385 **SPEC Benchmarks**

1386 The Standard Performance Evaluation Corporation (SPEC) [149, 150] is the other
1387 major standard for evaluation of benchmarking systems. SPEC has several different
1388 testing components that can be utilized to benchmark a system. For our benchmark-
1389 ing comparison we will use the SPEC OMP2001 because it appears to represent a
1390 vast array of new and emerging parallel applications while simultaneously providing
1391 a comparison to other SPEC benchmarks. SPEC OMP continues the SPEC tradi-

1392 tion of giving HPC users the most objective and representative benchmark suite for
1393 measuring the performance of SMP (shared memory multi-processor) systems.

- 1394 • The benchmarks are adapted from SPEC CPU2000 and contributions to its
1395 search program.
- 1396 • The focus is to deliver systems performance to real scientific and engineering
1397 applications.
- 1398 • The size and runtime reflect the needs of engineers and researchers to model
1399 large complex tasks.
- 1400 • Two levels of workload characterize the performance of medium and large sized
1401 systems.
- 1402 • Tools based on the SPEC CPU2000 toolset make these the easiest ever HPC
1403 tests to run.
- 1404 • These benchmarks place heavy demands on systems and memory.

1405 **3.6 Performance Comparison**

1406 The goal of this chapter is to effectively compare and contrast the various virtual-
1407 ization technologies, specifically for supporting HPC-based Clouds. The first set of
1408 results represent the performance of HPCC benchmarks. Each benchmark was run
1409 a total of 20 times, and the mean values taken with error bars represented using the
1410 standard deviation over the 20 runs. The benchmarking suite was built using the Intel
1411 11.1 compiler, uses the Intel MPI and MKL runtime libraries, all set with defaults
1412 and no optimizations whatsoever.

1413 We open first with High Performance Linpack (HPL), the de-facto standard for
1414 comparing resources. In Figure 3.3, we can see the comparison of Xen, KVM, and
1415 Virtual Box compared to native bare-metal performance. First, we see that native
1416 is capable of around 73.5 Gflops which, with no optimizations, achieves 75% of the
1417 theoretical peak performance. Xen, KVM and VirtualBox perform at 49.1, 51.8 and
1418 51.3 Gflops, respectively when averaged over 20 runs. However Xen, unlike KVM
1419 and VirtualBox, has a high degree of variance between runs. This is an interesting
1420 phenomenon for two reasons. First, this may impact performance metrics for other
1421 HPC applications and cause errors and delays between even pleasingly-parallel appli-
1422 cations and add to reducer function delays. Second, this wide variance breaks a key
1423 component of Cloud computing providing a specific and predefined quality of service.
1424 If performance can sway as widely as what occurred for Linpack, then this may have
1425 a negative impact on users.

1426 Next, we turn to another key benchmark within the HPC community, Fast Fourier
1427 Transforms (FFT). Unlike the synthetic Linpack benchmark, FFT is a specific, pur-
1428 poseful benchmark which provides results which are often regarded as more relative
1429 to a user's real-world application than HPL. From Figure 3.4, we can see rather dis-
1430 tinct results from what was previously provided by HPL. Looking at Star and Single
1431 FFT, its clear performance across all hypervisors is roughly equal to bare-metal per-
1432 formance, a good indication that HPC applications may be well suited for use on
1433 VMs. The results for MPI FFT also show similar results, with the exception of Xen,
1434 which has a decreased performance and high variance as seen in the HPL benchmark.
1435 Our current hypothesis is that there is an adverse affect of using Intel's MPI runtime
1436 on Xen, however the investigation is still ongoing.

1437 Another useful benchmark illustrative of real-world performance between bare-
1438 metal performance and various hypervisors are the ping-pong benchmarks. These

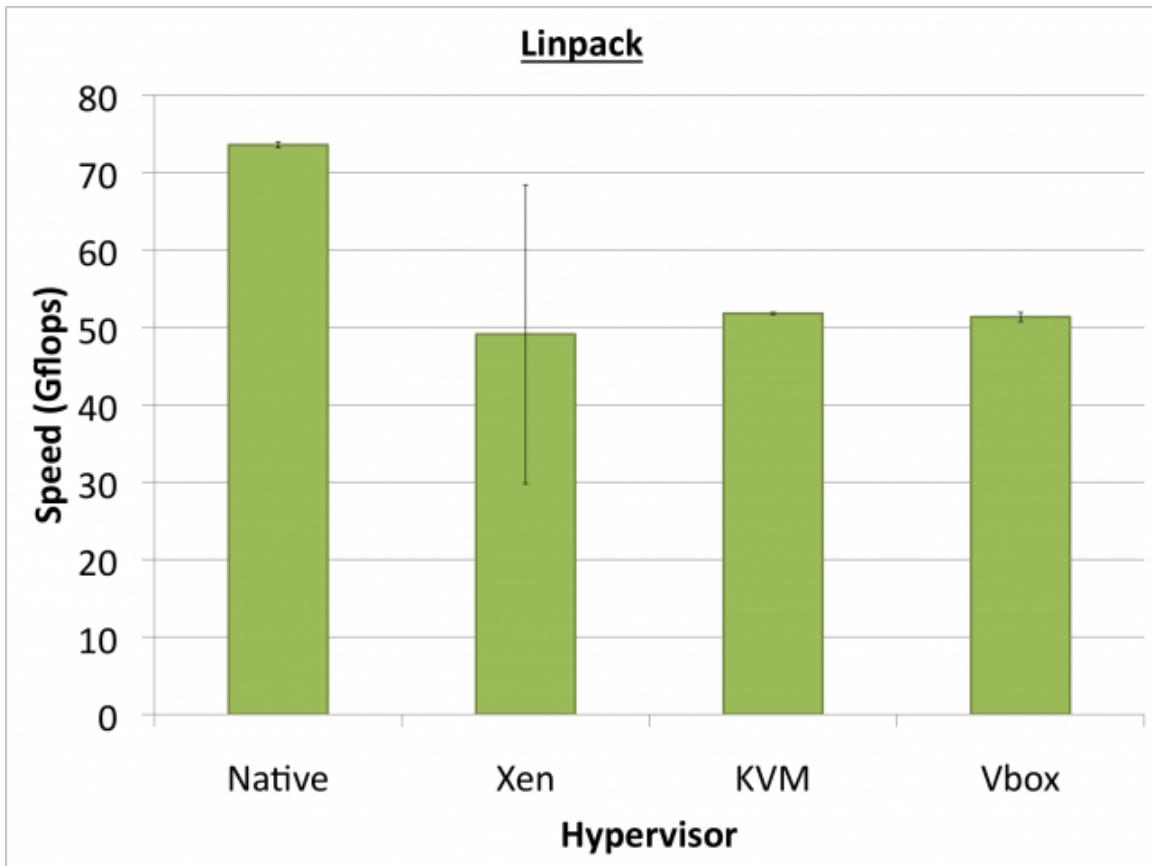


Figure 3.3 Linpack performance

1439 benchmarks measure the bandwidth and latency of passing packets between multiple
 1440 CPUs. With this experiment, all ping-pong latencies are kept within a given node,
 1441 rather than over the network. This is done to provide further insight into the CPU and
 1442 memory overhead within each hypervisor. From Figure 3.5 the intranode bandwidth
 1443 performance is uncovered, with some interesting distinctions between each hypervi-
 1444 sor. First, Xen performs, on average, close to native speeds, which is promising for
 1445 the hypervisor. KVM, on the other hand, shows consistent overhead proportional
 1446 to native performance across minimum, average, and maximum bandwidth. Virtu-
 1447 alBox, on the other hand, performs well, in fact too well to the point that raises
 1448 alarm. While the minimum and average bandwidths are within native performance,
 1449 the maximum bandwidth reported by VirtualBox is significantly greater than native

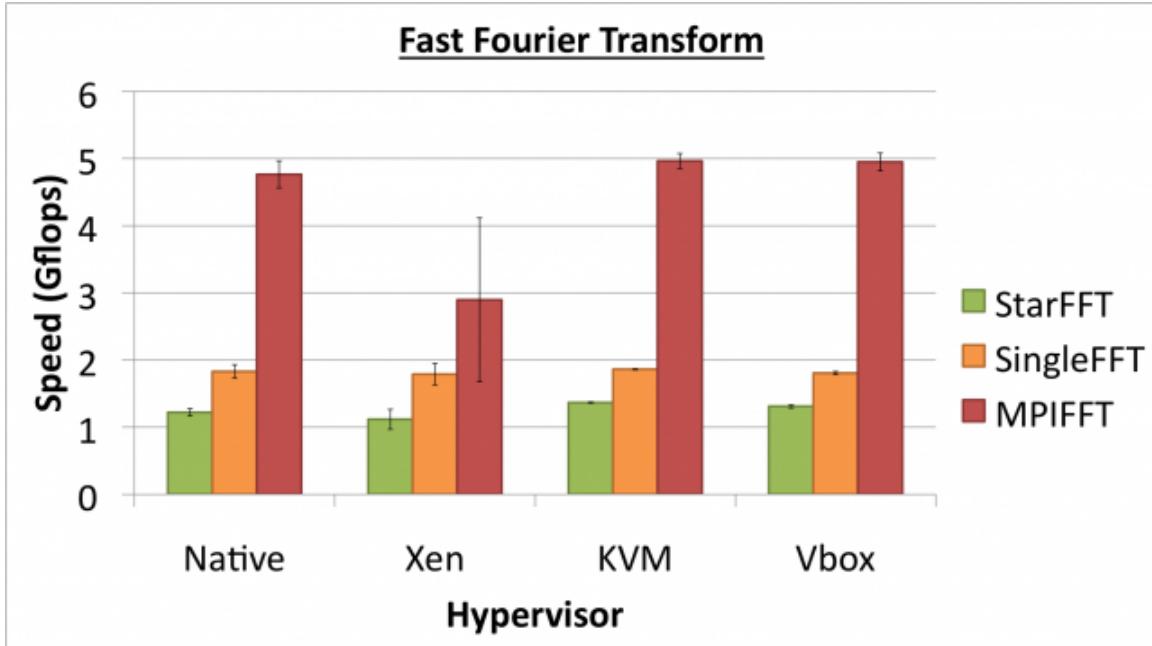


Figure 3.4 Fast Fourier Transform performance

1450 measurements, with a large variance. After careful examination, it appears this is
 1451 due to how VirtualBox assigns its virtual CPUs. Instead of locking a virtual CPU to
 1452 a real CPU, a switch may occur which could benefit on the off-chance the two CPU's
 1453 in communication between a ping-pong test could in fact be the same physical CPU.
 1454 The result would mean the ping-pong packet would remain in cache and result in a
 1455 higher perceived bandwidth than normal. While this effect may be beneficial for this
 1456 benchmark, it may only be an illusion towards the real performance gleaned from the
 1457 VirtualBox hypervisor.

1458 The Bandwidth may in fact be important within the ping-pong benchmark, but
 1459 the latency between each ping-pong is equally useful in understanding the perfor-
 1460 mance impact of each virtualization technology. From Figure 3.6, we see KVM and
 1461 VirtualBox have near-native performance; another promising result towards the util-
 1462 ity of hypervisors within HPC systems. Xen, on the other hand, has extremely high
 1463 latencies, especially at for maximum latencies, which in turn create a high variance

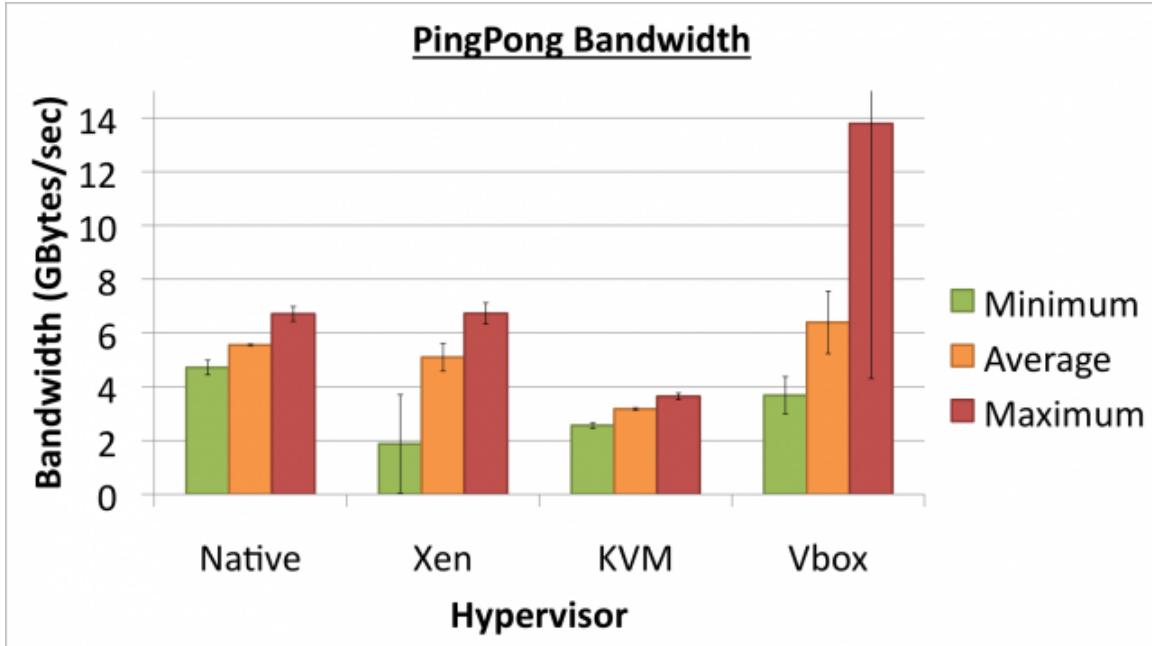


Figure 3.5 Ping Pong bandwidth performance

¹⁴⁶⁴ within the average latency within the VM’s performance.

¹⁴⁶⁵ 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. ¹⁴⁶⁶ 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. ¹⁴⁶⁷ KVM performance comes close with a score of 34384, which is so similar to the native performance that most users will never notice the difference. Xen and VirtualBox ¹⁴⁶⁸ both perform notably slower with scores of 31824 and 31695, respectively, however ¹⁴⁶⁹ this is only an 8% performance drop compared to native speeds. Further results can ¹⁴⁷⁰ be found on the SPEC website [151]. ¹⁴⁷¹

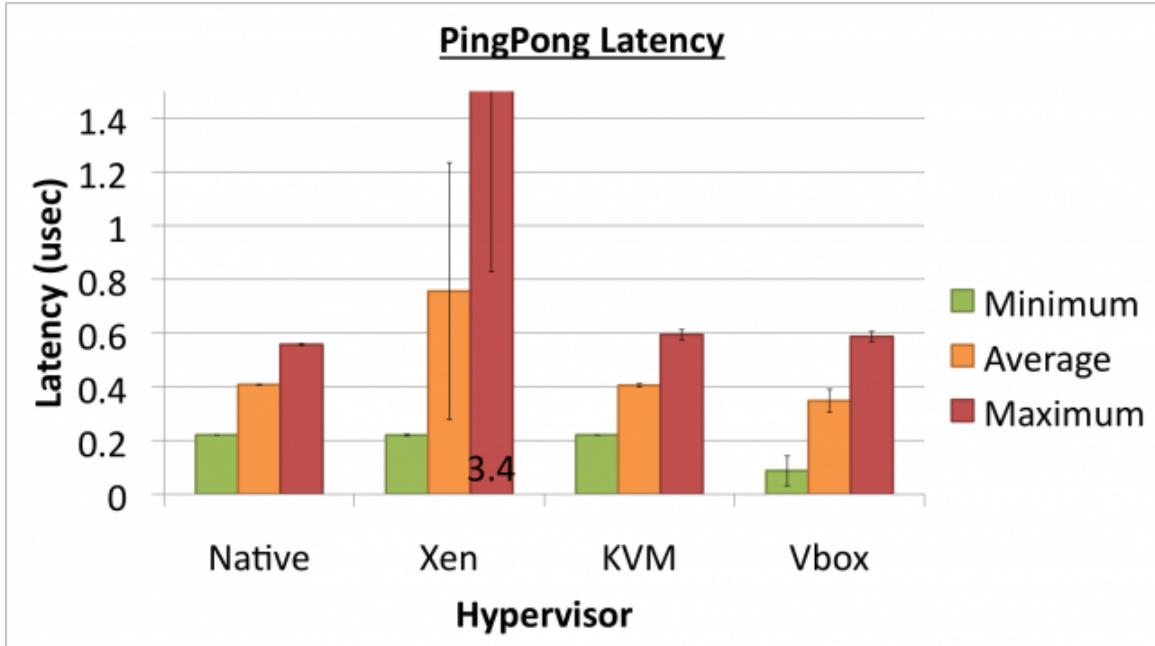


Figure 3.6 Ping Pong latency performance (lower is better)

1476 3.7 Discussion

1477 The primary goal of this chapter is to evaluate the viability of virtualization within
 1478 HPC. After our analysis, the answer seems to be a resounding "yes." However, we
 1479 also hope to select the best virtualization technology for such an HPC environment.

1480 In order to do this, we combine the feature comparison along with the performance
 1481 results, and evaluate the potential impact within the FutureGrid testbed.

1482 From a feature standpoint, most of today's virtualization technologies fit the bill
 1483 for at least small scale deployment, including VMWare. In short, each support Linux
 1484 x86_64 platforms, use VT-X technology for full virtualization, and support live mi-
 1485 gration. Due to VMWare's limited and costly licensing, it is immediately out of
 1486 contention for most HPC deployments. From a CPU and memory standpoint, Xen
 1487 seems to provide the best expandability, supporting up to 128 cpus and 4TB of ad-
 1488 dressable RAM. So long as KVM's vCPU limit can be extended, it too shows promise

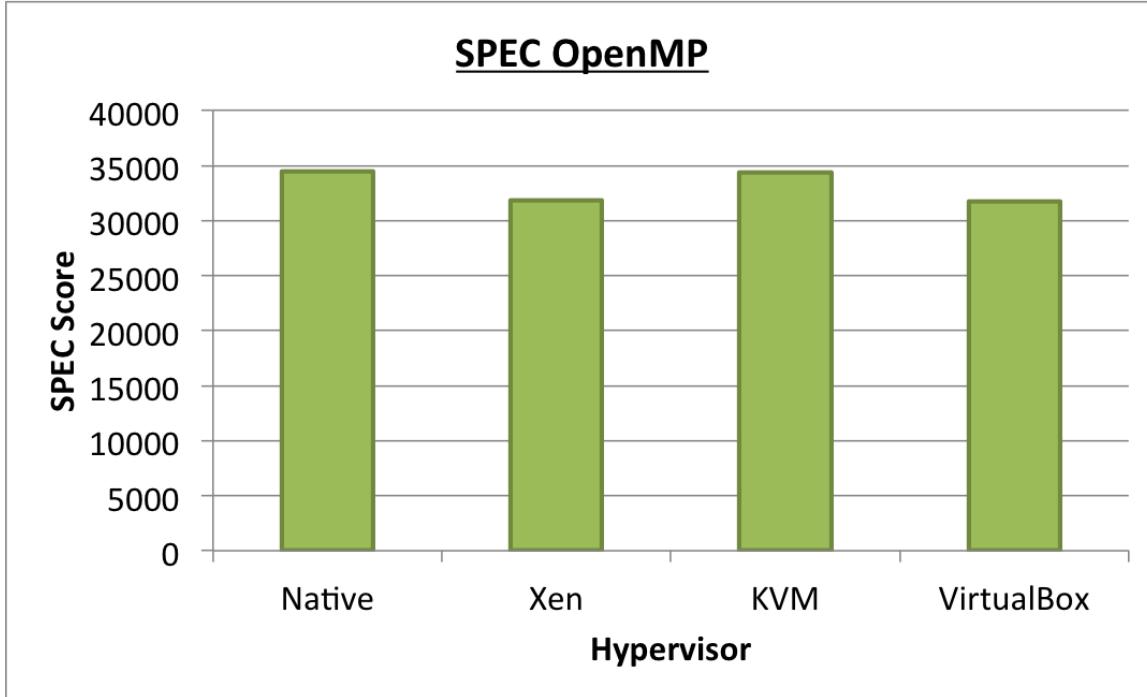


Figure 3.7 Spec OpenMP performance

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)

1502 through the SPEC OMP benchmarking suite also shows promising results for the use
 1503 of hypervisors in general, with KVM taking a clear lead by almost matching native
 1504 speeds.

1505 While Xen is typically regarded as the most widely used hypervisor, especially
 1506 within academic clouds and grids, it's performance has shown lack considerably when
 1507 compared to either KVM or VirtualBox. In particular, Xen's wide and unexplained
 1508 fluctuations in performance throughout the series of benchmarks suggests that Xen
 1509 may not be the best choice for building a lasting quality of service infrastructure upon.

1510 From Figure 3.8, KVM rates the best across all performance benchmarks, making it
 1511 the optimal choice for *general* deployment in an HPC environment. Furthermore,
 1512 this work's illustration of the variance in performance among each benchmark and
 1513 the applicability of each benchmark towards new applications may make possible the
 1514 ability to preemptively classify applications for accurate prediction towards the ideal
 1515 virtualized Cloud environment. We hope to further investigate this concept through
 1516 the use of the FutureGrid experiment management framework at a later date.

1517 In summary, it is the authors' projection that KVM is the best overall choice for

1518 use within HPC Cloud environments. KVM's feature-rich experience and near-native
1519 performance makes it a natural fit for deployment in an environment where usability
1520 and performance are paramount. Within the FutureGrid project specifically, we hope
1521 to deploy the KVM hypervisor across our Cloud platforms in the near future, as it
1522 offers clear benefits over the current Xen deployment. Furthermore, we expect these
1523 findings to be of great importance to other public and private Cloud deployments, as
1524 system utilization, Quality of Service, operating cost, and computational efficiency
1525 could all be improved through the careful evaluation of underlying virtualization
1526 technologies.

1527 **Chapter 4**

1528 **Evaluating GPU Passthrough in
1529 Xen for High Performance Cloud
1530 Computing**

1531 **4.1 Abstract**

1532 With the advent of virtualization and Infrastructure-as-a-Service (IaaS), the broader
1533 scientific computing community is considering the use of clouds for their technical
1534 computing needs. This is due to the relative scalability, ease of use, advanced user
1535 environment customization abilities clouds provide, as well as many novel comput-
1536 ing paradigms available for data-intensive applications. However, there is concern
1537 about a performance gap that exists between the performance of IaaS when com-
1538 pared to typical high performance computing (HPC) resources, which could limit the
1539 applicability of IaaS for many potential scientific users.

1540 Most recently, general-purpose graphics processing units (GPGPUs or GPUs) have
1541 become commonplace within high performance computing. We look to bridge the

1542 gap between supercomputing and clouds by providing GPU-enabled virtual machines
1543 (VMs) and investigating their feasibility for advanced scientific computation. Specif-
1544 ically, the Xen hypervisor is utilized to leverage specialized hardware-assisted I/O
1545 virtualization and PCI passthrough in order to provide advanced HPC-centric Nvidia
1546 GPUs directly in guest VMs. This methodology is evaluated by measuring the per-
1547 formance of two Nvidia Tesla GPUs within Xen VMs and comparing to bare-metal
1548 hardware. Results show PCI passthrough of GPUs within virtual machines is a vi-
1549 able use case for many scientific computing workflows, and could help support high
1550 performance cloud infrastructure in the near future.

1551 4.2 Introduction

1552 Cloud computing [4] has established itself as a prominent paradigm within the realm
1553 of Distributed Systems [152] in a very short period of time. Clouds are an internet-
1554 based solution that provide computational and data models for utilizing resources,
1555 which can be accessed directly by users on demand in a uniquely scalable way. Cloud
1556 computing functions by providing a layer of abstraction on top of base hardware
1557 to enable a new set of features that are otherwise intangible or intractable. These
1558 benefits and features include Scalability, a guaranteed Quality of Service (QoS), cost
1559 effectiveness, and direct user customization via a simplified user interface [64].

1560 While the origin of cloud computing is based in industry through solutions such
1561 as Amazon's EC2 [153], Google's MapReduce [154], and Microsoft's Azure [155], the
1562 paradigm has since become integrated in all areas of science and technology. Most
1563 notably, there is an increasing effort within the High Performance Computing (HPC)
1564 community to leverage the utility of clouds for advanced scientific computing to solve
1565 a number of challenges still standing in the field. This can be clearly seen in large-

1566 scale efforts such as the FutureGrid project [156], the Magellan project [47], and
1567 through various other Infrastructure-as-a-Service projects including OpenStack [157],
1568 Nimbus [74], and Eucalyptus [158].

1569 Within HPC, there has also been a notable movement toward dedicated accelerator
1570 cards such as general purpose graphical processing units (GPGPUs, or GPUs) to
1571 enhance scientific computation problems by upwards of two orders of magnitude.
1572 This is accomplished through dedicated programming environments, compilers, and
1573 libraries such as CUDA [159] from Nvidia as well as the OpenCL effort [160]. When
1574 combining GPUs in an otherwise typical HPC environment or supercomputer, major
1575 gains in performance and computational ability have been reported in numerous fields
1576 [161, 162], ranging from Astrophysics to Bioinformatics. Furthermore, these gains
1577 in computational power have also reportedly come at an increased performance-per-
1578 watt [163], a metric that is increasingly important to the HPC community as we move
1579 closer to exascale computing [164] where power consumption is quickly becoming the
1580 primary constraint.

1581 With the advent of both clouds and GPUs within the field of scientific computing,
1582 there is an immediate and ever-growing need to provide heterogeneous resources, most
1583 immediately GPUs, within a cloud environment in the same scalable, on-demand, and
1584 user-centric way that many cloud users are already accustomed to [165]. While this
1585 task alone is nontrivial, it is further complicated by the high demand for performance
1586 within HPC. As such, it is performance that is paramount to the success of deploying
1587 GPUs within cloud environments, and thus is the central focus of this work.

1588 The rest of this chapter is organized as follows. First, in Section 2, we discuss
1589 the related research and the options currently available for providing GPUs within a
1590 virtualized cloud environment. In Section 3, we discuss the methodology for providing
1591 GPUs directly within virtual machines. In Section 4 we outline the evaluation of the

1592 given methodology using two different Nvidia Tesla GPUs and compare to the best-
1593 case native application in Section 5. Then, we discuss the implications of these results
1594 in Section 6 and consider the applicability of each method within a production cloud
1595 system. Finally, we conclude with our findings and suggest directions for future work.

1596 4.3 Virtual GPU Directions

1597 Recently, GPU programming has been a primary focus for numerous scientific com-
1598 puting applications. Significant progress has been accomplished in many different
1599 workloads, both in science and engineering, based on parallel abilities of GPUs for
1600 floating point operations and very high on-GPU memory bandwidth. This hardware,
1601 coupled with CUDA and OpenCL programming frameworks, has led to an explosion
1602 of new GPU-specific applications. In some cases, GPUs outperform even the fastest
1603 multicore counterparts by an order of magnitude [166]. In addition, further research
1604 could leverage the per-node performance of GPU accelerators with the high speed,
1605 low latency interconnects commonly utilized in supercomputers and clusters to create
1606 a hybrid GPU + MPI class of applications. The number of distributed GPU appli-
1607 cations is increasing substantially in supercomputing, usually scaling many GPUs
1608 simultaneously [167].

1609 Since the establishment of cloud computing in industry, research groups have
1610 been evaluating its applicability to science [3]. Historically, HPC and Grids have
1611 been on similar but distinct paths within distributed systems, and have concentrated
1612 on performance, scalability, and solving complex, tightly coupled problems within
1613 science. This has led to the development of supercomputers with many thousands
1614 of cores, high speed, low latency interconnects, and sometimes also coprocessors and
1615 FPGAs [168, 169]. Only recently have these systems been evaluated from a cloud

1616 perspective [47]. An overarching goal exists to provide HPC Infrastructure as its own
1617 service (HPCaaS) [170], aiming to classify and limit the overhead of virtualization, and
1618 reducing the bottlenecks classically found in CPU, memory, and I/O operations within
1619 hypervisors [48, 171]. Furthermore, the transition from HPC to cloud computing
1620 becomes more complicated when we consider adding GPUs to the equation.

1621 GPU availability within a cloud is a new concept that has sparked a large amount
1622 of interest within the community. The first successfully deployment of GPUs within
1623 a cloud environment was the Amazon EC2 GPU offering. A collaboration between
1624 Nvidia and Citrix also exists to provide cloud-based gaming solutions to users using
1625 the new Kepler GPU architecture [172]. However, this is currently not targeted
1626 towards HPC applications.

1627 The task of providing a GPU accelerator for use in a virtualized cloud environment
1628 is one that presents a myriad of challenges. This is due to the complicated nature of
1629 virtualizing drivers, libraries, and the heterogeneous offerings of GPUs from multiple
1630 vendors. Currently, two possible techniques exist to fill the gap in providing GPUs
1631 in a cloud infrastructure: back-end I/O virtualization, which this chapter focuses on,
1632 and Front-end remote API invocation.

1633 4.3.1 Front-end Remote API invocation

1634 One method for using GPUs within a virtualized cloud environment is through front-
1635 end library abstractions, the most common of which is remote API invocation. Also
1636 known as API remoting or API interception, it represents a technique where API
1637 calls are intercepted and forwarded to a remote host where the actual computation
1638 occurs. The results are then returned to the front-end process that spawned the
1639 invocation, potentially within a virtual machine. The goal of this method is to provide
1640 an emulated device library where the actual computation is offloaded to another

1641 resource on a local network.

1642 Front-end remote APIs for GPUs have been implemented by a number of differ-
1643 ent technologies for different uses. To solve the problem of graphics processing in
1644 VMs, VMWare [173] has developed a device-emulation approach that emulates the
1645 Direct3D and OpenGL calls to leverage the host OS graphics processing capabili-
1646 ties to provide a 3D environment within a VM. API interception through the use of
1647 wrapper binaries has also been implemented by technologies such as Chromium [174],
1648 and Blink. However these graphics processing front-end solutions are not suitable for
1649 general purpose scientific computing, as they do not expose interfaces that CUDA or
1650 OpenCL can use.

1651 Currently, efforts are being made to provide a front-end remote API invocation
1652 solutions for the CUDA programming architecture. vCUDA [56] was the first of such
1653 technologies to enable transparent access of GPUs within VMs by API call inter-
1654 ception and redirection of the CUDA API. vCUDA substitutes the CUDA runtime
1655 library and supports a transmission mode using XMLRPC, as well as a sharing mode
1656 that is built on VMRPC, a dedicated remote procedure call architecture for VMM
1657 platforms. This share model can leads to better performance, especially as the volume
1658 of data increases, although there may be limitations in VMM interoperability.

1659 Like vCUDA, gVirtuS uses API interception to enable transparent CUDA, OpenCL,
1660 and OpenGL support for Xen, KVM, and VMWare virtual machines [175]. gVirtuS
1661 uses a front-end/back-end model to provide a VMM-independent abstraction layer
1662 to GPUs. Data transport from gVirtuS' front-end to the back-end is accomplished
1663 through a combination of shared memory, sockets, or other hypervisor-specific APIs.
1664 gVirtuS' primary disadvantage is in its decreased performance in host-to-device and
1665 device-to-host data movement due to overhead of data copies to and from its shared
1666 memory buffers. Recent work has also enabled the dynamic sharing of GPUs by

1667 leveraging the gVirtus back-end system with relatively good results [176], however
1668 process-level GPU resource sharing is outside the scope of this manuscript.

1669 rCUDA [55], a recent popular remote CUDA framework, also provides remote API
1670 invocation to enable VMs to access remote GPU hardware by using a sockets based
1671 implementation for high-speed near-native performance of CUDA based applications.
1672 rCUDA recently added support for using InfiniBand’s high speed, low latency network
1673 to increase performance for CUDA applications with large data volume requirements.
1674 rCUDA version 4.1 also supports the CUDA runtime API version 5.0, which supports
1675 peer device memory access and unified addressing. One drawback of this method
1676 is that rCUDA cannot implement the undocumented and hidden functions within
1677 the runtime framework, and therefore does not support all CUDA C extensions.
1678 While rCUDA provides some support tools, native execution of CUDA programs
1679 is not possible and programs need to be recompiled or rewritten to use rCUDA.
1680 Furthermore, like gVirtuS and many other solutions, performance between host-to-
1681 device data movement is only as fast as the underlying interconnect, and in the best
1682 case with native RDMA InfiniBand, is roughly half as fast as native PCI Express
1683 usage when using the standard QDR InfiniBand.

1684 4.3.2 Back-end PCI passthrough

1685 Another approach to using a GPU in a virtualized environment is to provide a VM
1686 with direct access to the GPU itself, instead of relying on a remote API. This chapter
1687 focuses on such an approach. Devices on a host’s PCI-express bus are virtualized
1688 using directed I/O virtualization technologies recently implemented by chip manu-
1689 facturers, and then direct access is relinquished upon request to a guest VM. This
1690 can be accomplished using the VT-d and IOMMU instruction sets from Intel and
1691 AMD, respectively. This mechanism, typically called PCI passthrough, uses a mem-

1692 ory management unit (MMU) to handle direct memory access (DMA) coordination
1693 and interrupt remapping directly to the guest VM, thus bypassing the host entirely.
1694 With host involvement being nearly non-existent, near-native performance of the PCI
1695 device within the guest VM can be achieved, which is an important characteristic for
1696 using a GPU within a cloud infrastructure.

1697 PCI passthrough itself has recently become a standard technique for many other
1698 I/O systems such as storage or network controllers. However, GPUs (even from
1699 the same vendor) have additional legacy VGA compatibility issues and non-standard
1700 low-level interface DMA interactions that make direct PCI passthrough nontrivial.
1701 VMWare has started use of a vDGA system for hardware GPU utilization, however it
1702 remains in tech preview and only documentation for Windows VMs is present [173]. In
1703 our experimentation, we have found that the Xen hypervisor provides a good platform
1704 for performing PCI passthrough of GPU devices to VMs due to its open nature,
1705 extensive support, and high degree of reconfigurability. Work with Xen in [177] gives
1706 hints at good performance for PCI passthrough in Xen, however further evaluation
1707 with independent benchmarks is needed when looking at scientific computing with
1708 GPUs.

1709 Today's GPUs can provide a variety of frameworks for application programmers to
1710 use. Two common solutions are CUDA and OpenCL. CUDA, or the Compute Unified
1711 Device Architecture, is a framework for creating and running parallel applications on
1712 Nvidia GPUs. OpenCL provides a more generic and open framework for parallel
1713 computation on CPUs and GPUs, and is available for a number of Nvidia, AMD, and
1714 Intel GPUs and CPUs. While OpenCL provides a more robust and portable solution,
1715 many HPC applications utilize the CUDA framework. As such, we focus only on
1716 Nvidia based CUDA-capable GPUs as they offer the best support for a wide array of
1717 programs, although this work is not strictly limited to Nvidia GPUs.

1718 4.4 Implementation

1719 In this chapter we use a specific host environment to enable PCI passthrough. First,
1720 we start with the Xen 4.2.2 hypervisor on Centos 6.4 and a custom 3.4.50-8 Linux
1721 kernel with Dom0 Xen support. Within the Xen hypervisor, GPU devices are seized
1722 upon boot and assigned to the xen-pciback kernel module. This process blocks the
1723 host devices from loading the Nvidia or nouveau drivers, keeping the GPUs uninitialized
1724 and therefore able to be assigned to DomU VMs.

1725 Xen, like other hypervisors, provides a standard method of passing through PCI
1726 devices to guest VMs upon creation. When assigning a GPU to a new VM, Xen loads
1727 a specific VGA BIOS to properly initialize the device enabling DMA and interrupts
1728 to be assigned to the guest VM. Xen also relinquishes control of the GPU via the
1729 xen-pciback module. From there, the Linux Nvidia drivers are loaded and the device
1730 is able to be used as expected within the guest. Upon VM termination, the xen-
1731 pciback module re-seizes the GPU and the devices can be re-assigned to new VMs in
1732 the future.

1733 This mechanism of PCI passthrough for GPUs can be implemented using multiple
1734 devices per host, as illustrated in Figure 6.1. Here, we see how the device's connection
1735 to the VM totally bypasses the Dom0 host as well as the Xen VMM, and is managed
1736 by VT-d or IOMMU to access the PCI-Express bus which the GPUs utilize. This is in
1737 contrast to other common virtual device uses, where hardware is emulated by the host
1738 and shared across all guests. This is the common usage for Ethernet controllers and
1739 input devices to enable users to interact with VMs as they would with native hosts,
1740 unlike the bridged model shown in the figure. The potential downside of this method
1741 is there can be a 1:1 or 1:N mapping of VMs to GPUs only. A M:1 mapping where
1742 multiple VMs use a GPU is not possible. However, almost all scientific applications

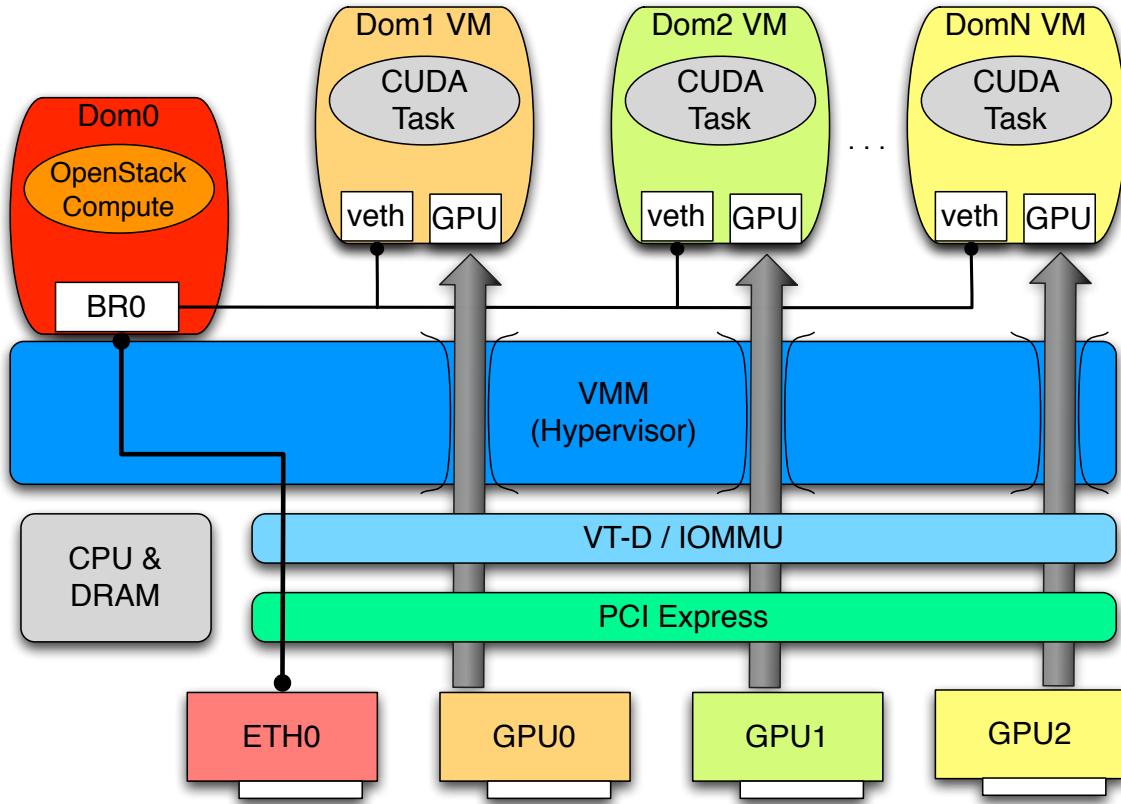


Figure 4.1 GPU PCI passthrough within the Xen Hypervisor

1743 environments using GPUs generally do not share GPUs between processes or other
 1744 nodes, as doing so would cause unpredictable and serious performance degradation.
 1745 As such, this GPU isolation within a VM can be considered an advantage in many
 1746 contexts.

1747 4.4.1 Feature Comparison

1748 Using the GPU PCI passthrough technique described previously has a number of
 1749 advantages compared to front-end API implementations. First, it allows for an op-
 1750 erating environment that more closely relates to native bare-metal usage of GPUs.
 1751 Essentially, a VM provides a nearly identical infrastructure to clusters and supercom-
 1752 puters with integrated GPUs. This lowers the learning curve for many researchers,

1753 and even enables researchers to potentially use other tools within a VM that might
1754 not be supported within a supercomputer or cluster. Unlike with remote API im-
1755 plementations, users don't need to recompile or modify their code, as the GPUs are
1756 essentially local to the data. Further comparing to remote API implementations,
1757 using PCI passthrough within a VM allows users to leverage any GPU framework
1758 available, OpenCL or CUDA, and any higher level programming frameworks such as
1759 within Matlab or Python.

1760 Through the use of advanced scheduling techniques within cloud infrastructure, we
1761 can also take advantage of PCI passthrough implementation for efficiency purposes.
1762 Users could request VMs with GPUs which get scheduled for creation on machines
1763 that provide such resources, but can also request normal VMs as well. The scheduler
1764 can correctly map VM requirement requests to heterogeneous hardware. This enables
1765 large scale resources to support a wide array of scientific computing applications
1766 without the added cost of putting GPUs in all compute nodes.

1767 4.5 Experimental Setup

1768 In this chapter back-end GPU PCI passthrough to virtual machines using the Xen
1769 hypervisor is detailed, however proper evaluation of the performance of such method
1770 needs to be properly considered. As such, we ran an array of benchmarks that
1771 evaluate the performance of this method compared to the same hardware running
1772 native bare-metal GPU code without any virtualization. We focus our tests on single-
1773 node performance to best understand low level overhead.

1774 To evaluate the effectiveness of GPU-enabled VMs within Xen, two different ma-
1775 chines were used to represent two generations of Nvidia GPUs. The first system at
1776 Indiana University consists of dual-socket Intel Xeon X5660 6-core CPUs at 2.8Ghz

1777 with 192GB DDR3 RAM, 8TB RAID5 array, and two Nvidia Tesla "Fermi" C2075
1778 GPUs. The system at USC/ISI uses a dual-socket Intel Xeon E5-2670 8-core CPUs
1779 at 2.6Ghz with 48GB DDR3 RAM, 3 600GB SAS disk drives, but with the latest
1780 Nvidia Tesla "Kepler" K20m GPU supplied by Nvidia. These machines represent
1781 the present Fermi series GPUs along with the recently release Kepler series GPUs,
1782 providing a well-rounded experimental environment. Native systems were installed
1783 with standard Centos 6.4 with a stock 2.6.32-279 Linux kernel. Xen host systems
1784 were still loaded with Centos 6.4 but with a custom 3.4.53-8 Linux kernel and Xen
1785 4.2.22. Guest VMs were also Centos 6.4 with N-1 processors and 24GB of memory
1786 and 1 GPU passed through in HVM full virtualization mode. Using both IU and
1787 USC/ISI machine configurations in native and VM modes represent the 4 test cases
1788 for our work.

1789 In order to evaluate the performance, the SHOC Benchmark suite [178] was used
1790 to extensively evaluate performance across each test platform. The SHOC bench-
1791 marks were chosen because they provide a higher level of evaluation regarding GPU
1792 performance than the sample applications provided in the Nvidia SDK, and can also
1793 evaluate OpenCL performance in similar detail. The benchmarks were compiled us-
1794 ing the NVCC compiler in CUDA5 driver and library, along with OpenMPI 1.4.2 and
1795 GCC 4.4. Each benchmark was run a total of 20 times, with the results given as an
1796 average of all runs.

1797 4.6 Results

1798 Results of all benchmarks are compressed into three subsections: floating point oper-
1799 ations, device bandwidth and pci bus performance. Each represents a different level
1800 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%).

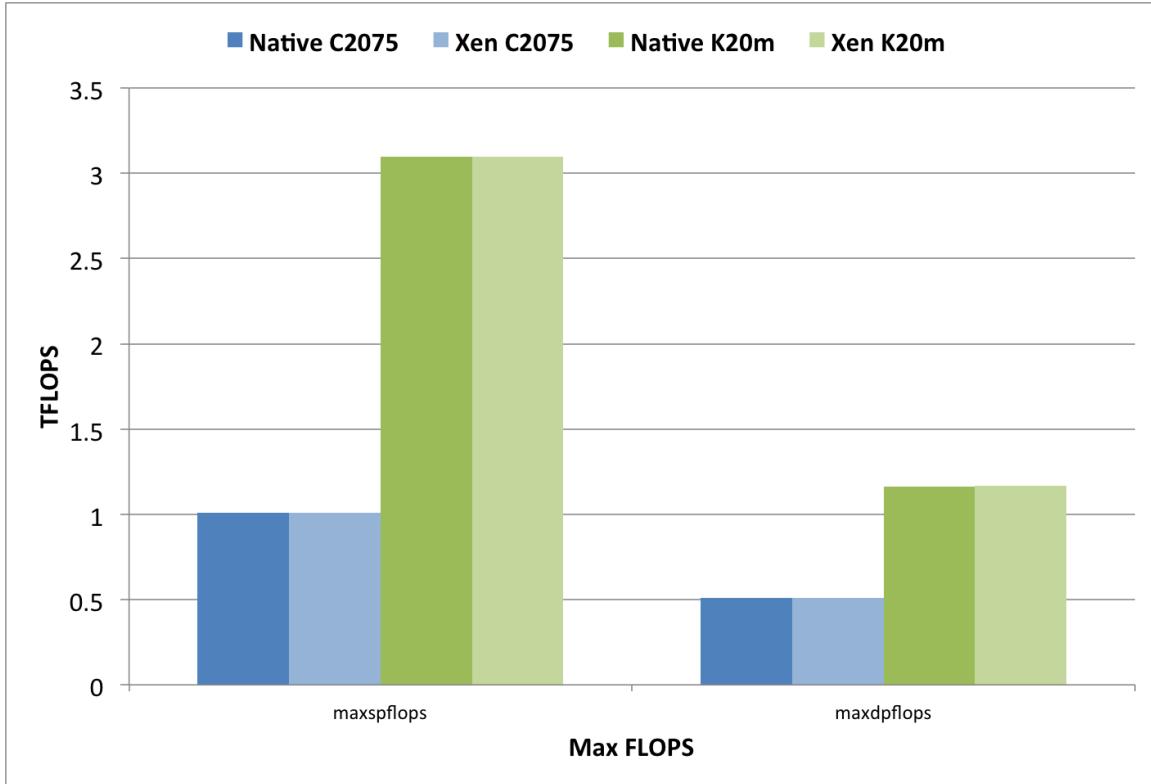


Figure 4.2 GPU Floating Point Operations per Second

1827 Other FLOP-based benchmarks are used to emulate higher level applications.
 1828 Stencil represents a 9-point stencil operation applied to a 2D data set, and the S3D
 1829 benchmark is a computationally-intensive kernel from the S3D turbulent combustion
 1830 simulation program [179]. In Figure 4.5, we see that both the Fermi C2075 and Kepler
 1831 K20m GPUs performing well compared to the native base case, showing the overhead
 1832 of virtualization is low. The C2075-enabled VMs experience slightly more overhead
 1833 when compared to native performance again for pcie runs, but overhead is at most
 1834 7% for the S3D benchmark.

1835 4.6.2 Device Speed

1836 While floating point operations allow for the proposed solution to relate to many tradi-
 1837 tional HPC applications, they are just one facet of GPU performance within scientific

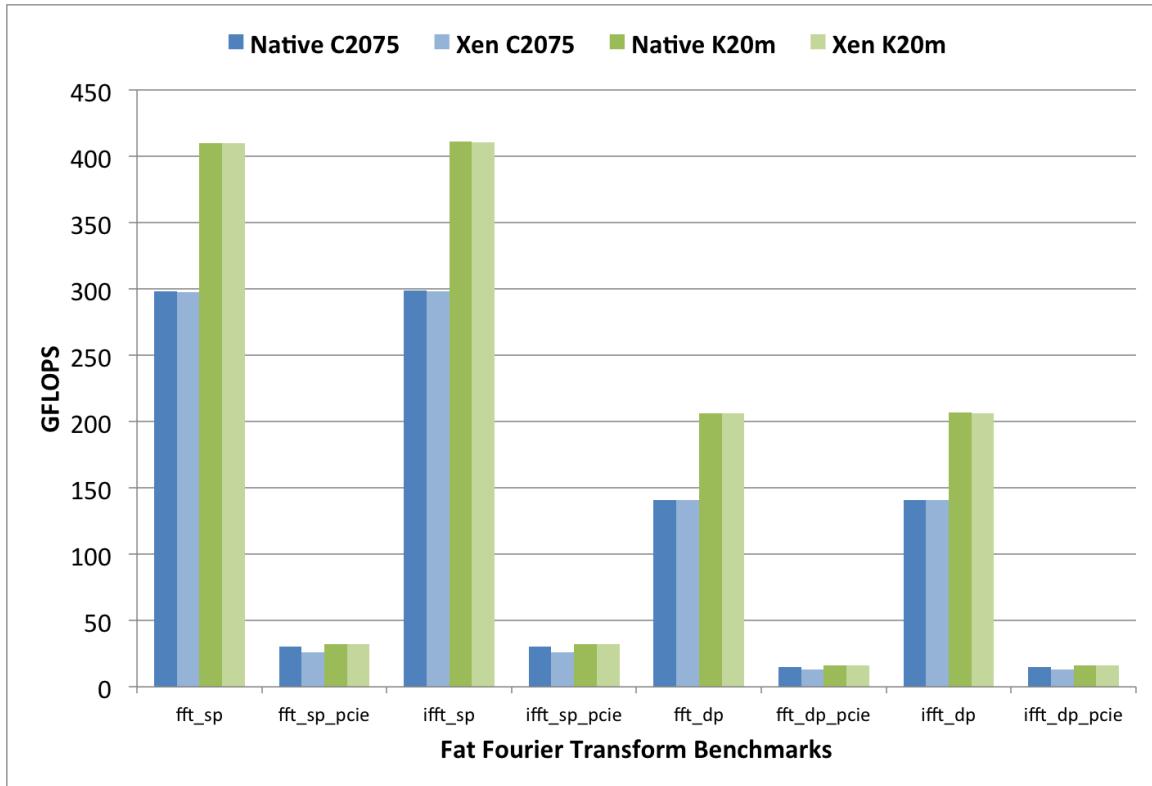


Figure 4.3 GPU Fast Fourier Transform

1838 computing. Device speed, measured in both raw bandwidth and additional bench-
 1839 marks, provides a different perspective towards evaluating GPU PCI passthrough in
 1840 Xen. Figure 4.6 illustrates device level memory access of various GPU device mem-
 1841 ory structures. With both Nvidia GPUs, virtualization has little to no impact on the
 1842 performance of inter-device memory bandwidth. As expected the Kepler K20m out-
 1843 performed the C2075 VMs and there was a higher variance between runs with both
 1844 native and VM cases. Molecular Dynamics and Reduction benchmarks in Figure 4.7
 1845 perform again at near-native performance without the pcie bus taken into account.
 1846 However the overhead observed increases to 10-15% when the PCI-Express bus is
 1847 considered when looking at the Fermi C2075 VMs.

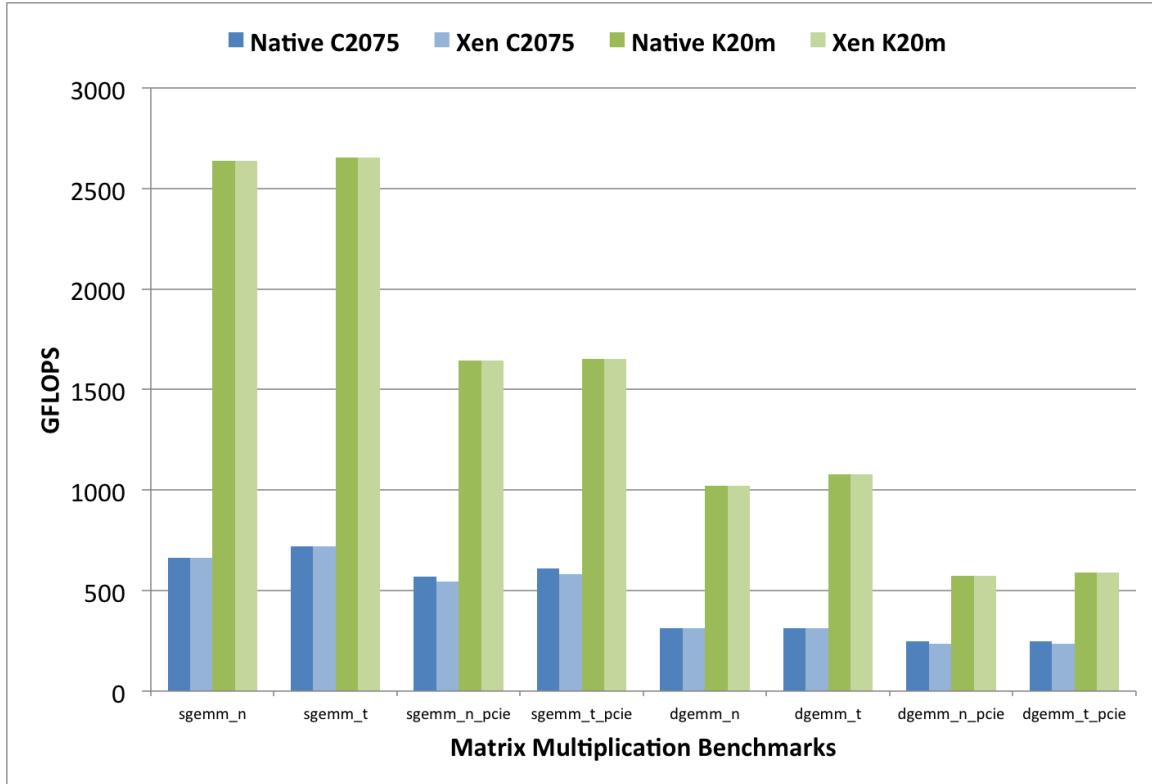


Figure 4.4 GPU Matrix Multiplication

1848 4.6.3 PCI Express Bus

1849 Upon evaluating PCI passthrough of GPUs, it is apparent that the PCI express bus
 1850 is subject to the greatest potential for overhead, as was observed in the Fermi C2075
 1851 benchmarks. The VT-d and IOMMU chip instruction sets interface directly with the
 1852 PCI bus to provide operational and security related mechanisms for each PCI device,
 1853 thereby ensuring proper function in a multi-guest environment but potentially intro-
 1854 ducing some overhead. As such, it is imperative to investigate any and all overhead
 1855 at the PCI Express bus.

1856 Figure 4.8 looks at maximum PCI bus speeds for each experimental implemen-
 1857 tation. First, we see a consistent overhead in the Fermi C2075 VMs, with a 14.6%
 1858 performance impact for download (to-device) and a 26.7% impact in readback (from-

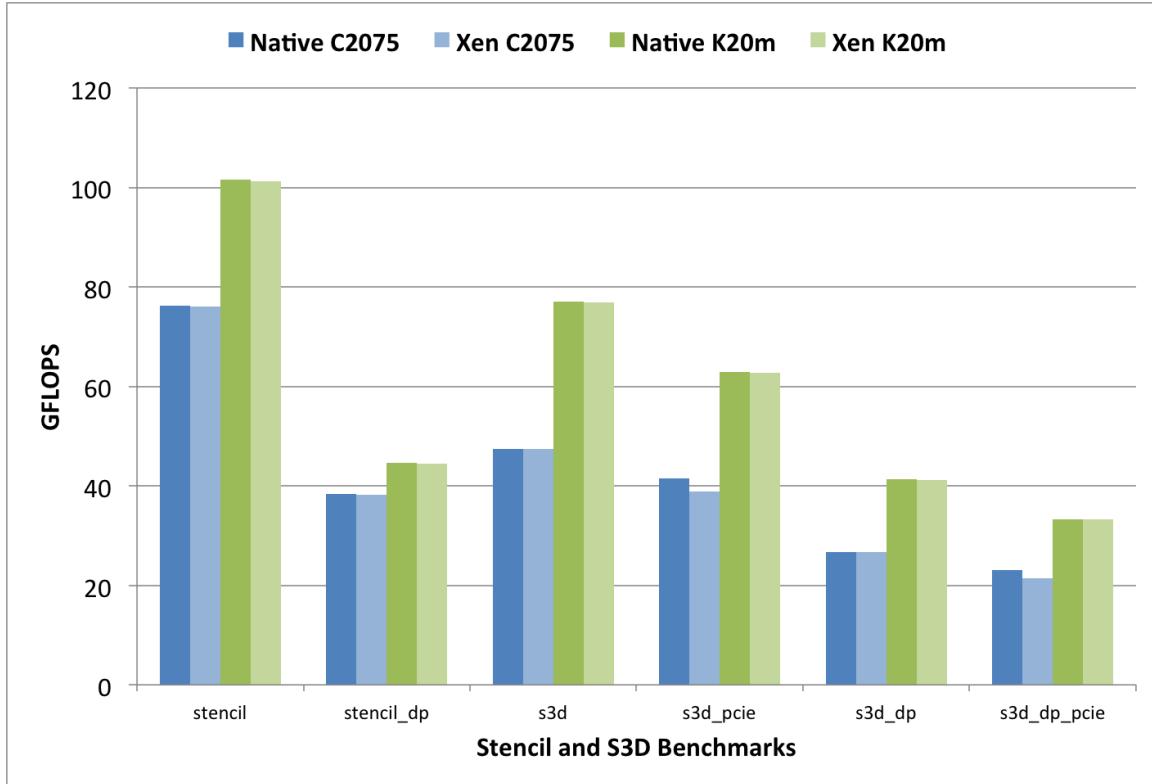


Figure 4.5 GPU Stencil and S3D

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-

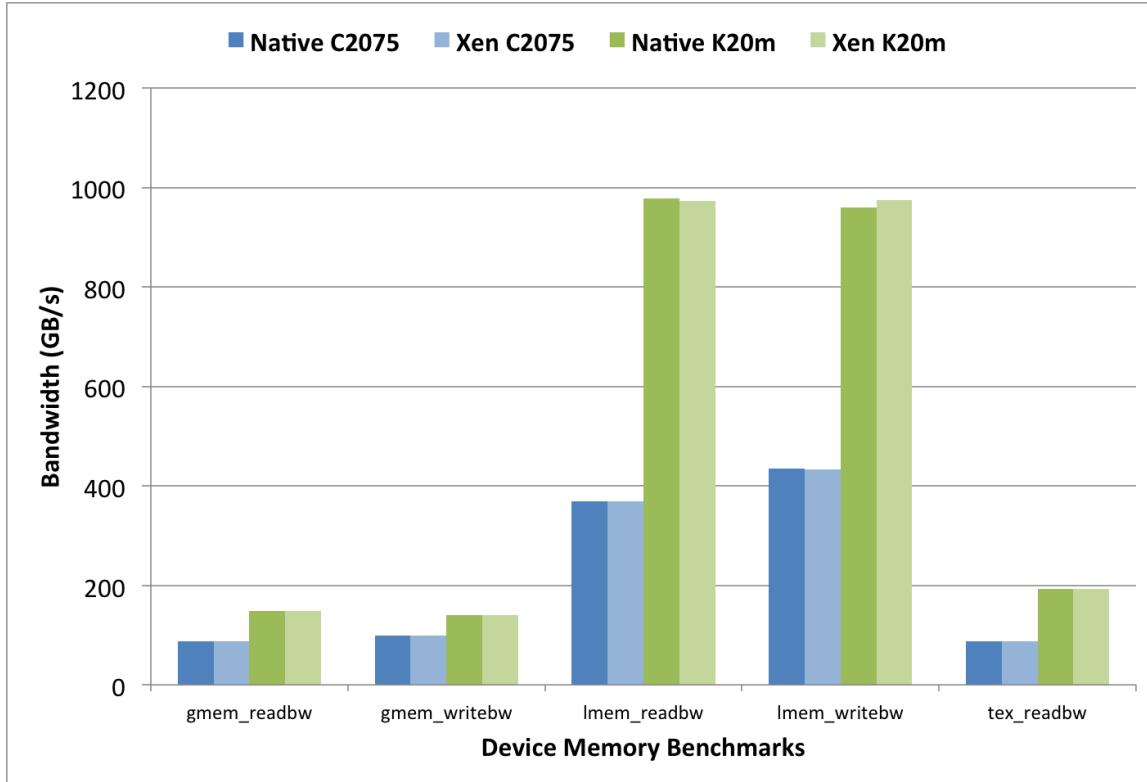


Figure 4.6 GPU Device Memory Bandwidth

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

4.7 Discussion

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

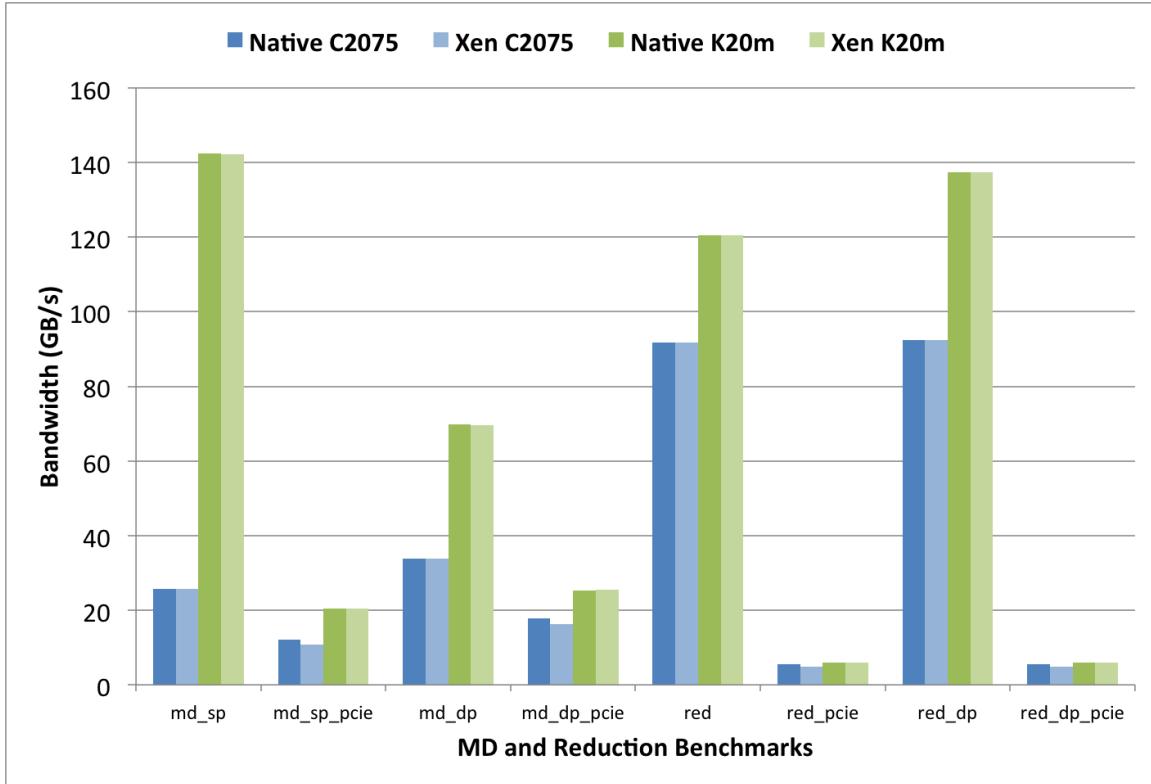


Figure 4.7 GPU Molecular Dynamics and Reduction

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-

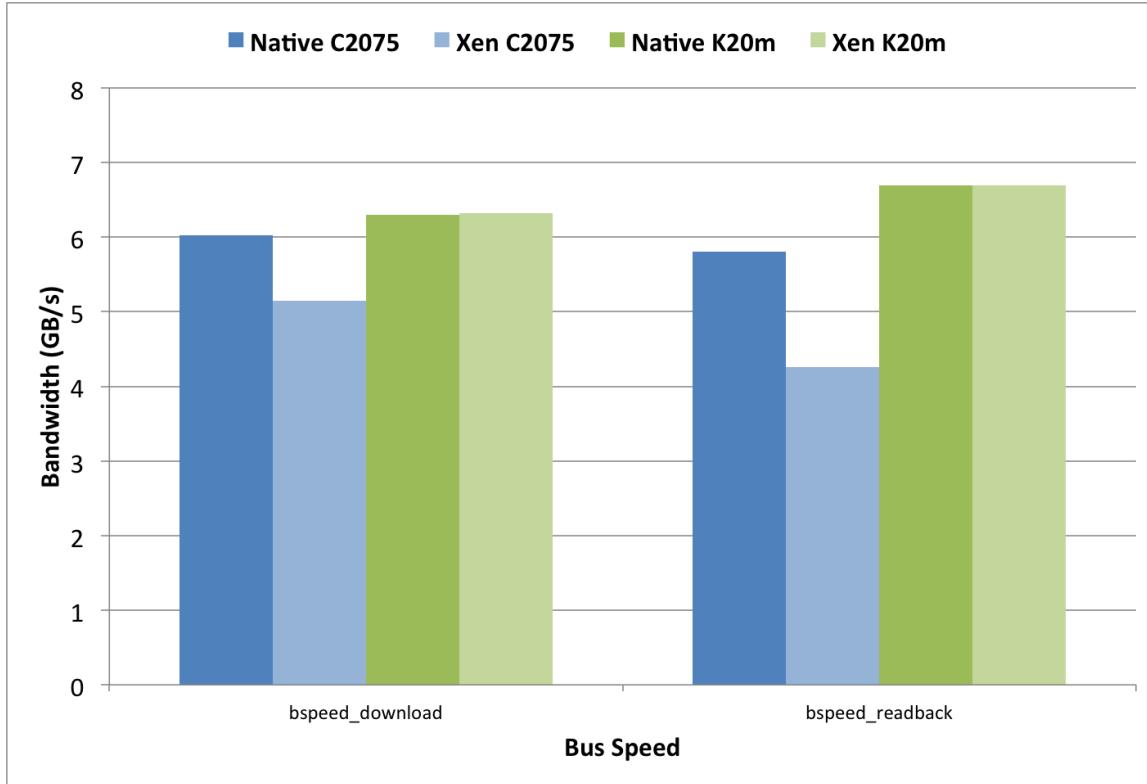


Figure 4.8 GPU PCI Express Bus Speed

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 [180], 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-

1905 tructure. Already this has been confirmed in [181], where similar a methodology has
1906 been leveraged specifically for Bioinformatics applications in the cloud.

1907 4.8 Chapter Summary and Future Work

1908 The ability to use GPUs within virtual machines represents a leap forward for sup-
1909 porting advanced scientific computing within cloud infrastructure. The method of
1910 direct PCI passthrough of Nvidia GPUs using the Xen hypervisor offers a clean, re-
1911 producible solution that can be implemented within many Infrastructure-as-a-Service
1912 (IaaS) deployments. Performance measurements indicate that the overhead of pro-
1913 viding a GPU within Xen is minimal compared to the best-case native use, however
1914 NUMA inconsistencies can impact performance. The New Kepler-based GPUs oper-
1915 ate with a much lower overhead, making those GPUs an ideal choice when designing
1916 a new GPU IaaS system.

1917 Next steps for this work could involve providing GPU-based PCI passthrough
1918 within the OpenStack nova IaaS framework. This will enable research laboratories
1919 and institutions to create new private or national-scale cloud infrastructure that have
1920 the ability to support new scientific computing challenges. Other hypervisors could
1921 also leverage GPU PCI passthrough techniques and warrant in-depth evaluation in
1922 the future. Furthermore, we hope to integrate this work with advanced interconnects
1923 and other heterogeneous hardware and provide a parallel high performance cloud
1924 infrastructure to enable mid-tier scientific computing.

1925 **Chapter 5**

1926 **GPU-Passthrough Performance: A
1927 Comparison of KVM, Xen,
1928 VMWare ESXi, and LXC for
1929 CUDA and OpenCL Applications**

1930 **5.1 Abstract**

1931 As more scientific workloads are moved into the cloud, the need for high performance
1932 accelerators increases. Accelerators such as GPUs offer improvements in both per-
1933 formance and power efficiency over traditional multi-core processors; however, their
1934 use in the cloud has been limited. Today, several common hypervisors support GPU-
1935 passthrough, but their performance has not been systematically characterized.

1936 In this chapter we show that low overhead PCI passthrough is achievable across
1937 4 major hypervisors and two processor microarchitectures. We compare the perfor-
1938 mance of two generations of NVIDIA GPUs within the Xen, VMWare ESXi, and

1939 KVM hypervisors, and we also compare the performance to that of Linux Containers
1940 (LXC). We show that GPU passthrough to KVM achieves 98–100% of the base sys-
1941 tem’s performance across two architectures, while Xen and VMWare achieve 96–99%
1942 of the base systems performance, respectively. In addition, we describe several valua-
1943 ble lessons learned through our analysis and share the advantages and disadvantages
1944 of each hypervisor/PCI passthrough solution.

1945 5.2 Introduction

1946 As scientific workloads continue to demand increasing performance at greater power
1947 efficiency, high performance architectures have been driven towards heterogeneity and
1948 specialization. Intel’s Xeon Phi, and GPUs from both NVIDIA and AMD represent
1949 some of the most common accelerators, with each capable of delivering improved
1950 performance and power efficiency over commodity multi-core CPUs.

1951 Infrastructure-as-a-Service (IaaS) clouds have the potential to democratize access
1952 to the latest, fastest, and most powerful computational accelerators. This is true
1953 of both public and private clouds. Yet today’s clouds are typically homogeneous
1954 without access to even the most commonly used accelerators. Historically, enabling
1955 virtual machine access to GPUs and other PCIe devices has proven complex and
1956 error-prone, with only a small subset of GPUs being certified for use within a few
1957 commercial hypervisors. This is especially true for NVIDIA GPUs, likely the most
1958 popular for scientific computing, but whose drivers have always been closed source.

1959 Given the complexity surrounding the choice of GPUs, host systems, and hypervi-
1960 sors, it is perhaps no surprise that Amazon is the only major cloud provider offering
1961 customers access to GPU-enabled instances. All of this is starting to change, however,
1962 as open source and other freely available hypervisors now provide sufficiently robust

1963 PCI passthrough functionality to enable GPU and other accelerator access whether
1964 in the public or private cloud.

1965 Today, it is possible to access GPUs at high performance within all of the major
1966 hypervisors, merging many of the advantages of cloud computing (e.g. custom images,
1967 software defined networking, etc.) with the accessibility of on-demand accelerator
1968 hardware. Yet, no study to date has systematically compared the performance of PCI
1969 passthrough across all major cloud hypervisors. Instead, alternative solutions have
1970 been proposed that attempt to virtualize the GPU [182] , but sacrifice performance.

1971 In this chapter, we characterize the performance of both NVIDIA Fermi and
1972 Kepler GPUs operating in PCI passthrough mode in VMWare VSphere, Linux KVM,
1973 Xen, and Linux Containers (LXC). Through a series of microbenchmarks as well as
1974 scientific and Big Data applications, we make two contributions:

- 1975 1. We demonstrate that PCI passthrough at high performance is possible for GPUs
1976 across 4 major hypervisors.
- 1977 2. We describe the lessons learned through our performance analysis, as well as
1978 the relative advantages and disadvantages of each hypervisor for GPU support.

1979 **5.3 Related Work & Background**

1980 GPU virtualization and GPU-passthrough are used within a variety of contexts, from
1981 high performance computing to virtual desktop infrastructure. Accessing one or more
1982 GPUs within a virtual machine is typically accomplished by one of two strategies: 1)
1983 via API remoting with device emulation; or 2) using PCI passthrough.

1984 **5.3.1 GPU API Remoting**

1985 rCUDA, vCUDA, GVIM, and gVirtuS are well-known API remoting solutionsFun-
1986 damentally, these approaches operate similarly by splitting the driver into a front-
1987 end/back-end model, where calls into the interposed CUDA library (front-end) are
1988 sent via shared memory or a network interface to the back-end service that executes
1989 the CUDA call on behalf of the virtual machine. Notably, this technique is not limited
1990 to CUDA, but can be used to decouple OpenCL, OpenGL, and other APIs from their
1991 local GPU or accelerator.

1992 The performance of API-remoting depends largely on the application and the
1993 remoting solution's implementation. Bandwidth and latency-sensitive benchmarks and
1994 applications will tend to expose performance bottlenecks more than compute-intensive
1995 applications. Moreover, solutions that rely on high speed networks, such as Infini-
1996 band, will compete with application-level networking for bandwidth.

1997 **5.3.2 PCI Passthrough**

1998 Input/Output Memory Management Units, or IOMMUs, play a fundamental roll in
1999 the PCI-passthrough virtualization mechanism. Like traditional MMUs that provide
2000 a virtual memory address space to CPUs [183], an IOMMU serves the fundamental
2001 purpose of connecting a direct memory access (DMA) capable I/O bus to main mem-
2002 ory. The IOMMU unit, typically within the chipset, maps device virtual addresses to
2003 physical memory addresses. This process also has the added improvement of guaran-
2004 teeing device isolation by blocking rogue DMA and interrupt requests [184], with a
2005 slight overhead, especially in early implementations [185].

2006 Currently two major IOMMU implementations exist, VT-d and AMD-Vi by In-
2007 tel and AMD, respectively. Both specifications provide DMA remapping to enable

2008 PCI-passthrough as well as other features such as interrupt remapping, hypervisor
2009 snooping, and security control mechanisms to ensure proper and efficient hardware
2010 utilization. PCI passthrough has been studied within the context of networking [186],
2011 storage [187], and other PCI-attached devices; however, GPUs have historically lagged
2012 behind other devices in their support for virtual machine passthrough.

2013 **5.3.3 GPU Passthrough, a Special Case of PCI Passthrough**

2014 While generic PCI passthrough can be used with IOMMU technologies to pass through
2015 many PCI-Express devices, GPUs represent a special case of PCI devices, and a spe-
2016 cial case of PCI passthrough. In traditional usage, GPUs usually serve as VGA
2017 devices primarily to render screen output, and while the GPUs used in this study do
2018 not render screen out, the function still exists in legacy. In GPU-passthrough, another
2019 VGA device (such as onboard graphics built into the motherboard, or a baseboard
2020 management controller) is necessary to serve as the primary display for the host, as
2021 well as providing emulated VGA devices for each guest VM. Most GPUs also have a
2022 video BIOS that requires full initialization and reset functions, which is often difficult
2023 due to the proprietary nature of the cards and their drivers.

2024 Nevertheless, for applications that require native or near-native GPU performance
2025 across the full spectrum of applications with immediate access to the latest GPU
2026 drivers and compilers, GPU passthrough solutions are preferable to API remoting.
2027 Today, Citrix Xenserver, open source Xen [188], and VMWare ESXi [189], and most
2028 recently KVM all support GPU passthrough. To our knowledge, no one has system-
2029 atically characterized the performance of GPU passthrough across a range of hyper-
2030 visors, across such a breadth of benchmarks, and across multiple GPU generations as
2031 we do.

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

2032 5.4 Experimental Methodology

2033 5.4.1 Host and Hypervisor Configuration

2034 We used two hardware systems, named Bespin and Delta, to evaluate four hypervisors.
 2035 The Bespin system at USC/ISI represents Intel's Sandy Bridge microarchitecture with
 2036 a Kepler class K20 GPU. The Delta system, provided by the FutureGrid project [190],
 2037 represents the Westmere microarchitecture with a Fermi class C2075 GPU. Table 5.1
 2038 provides the major hardware characteristics of both systems. Note that in addition,
 2039 both systems include 10 gigabit Ethernet, gigabit Ethernet, and either FDR or QDR
 2040 Infiniband. Our experiments do not emphasize networking, and we use the gigabit
 2041 ethernet network for management only.

2042 A major design goal of these experiments was to reduce or eliminate NUMA effects
 2043 (non-uniform memory access) on the PCI passthrough results in order to facilitate
 2044 fair comparisons across hypervisors and to reduce experimental noise. To this end,
 2045 we configured our virtual machines and containers to execute only on the NUMA
 2046 node containing the GPU under test. We acknowledge that the NUMA effects on
 2047 virtualization may be interesting in their own right, but they are not the subject of

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

2048 this set of experiments.

2049 We use Bespin and Delta to evaluate three hypervisors and one container-based
 2050 approach to GPU passthrough. The hypervisors and container system, VMWare
 2051 ESXi, Xen, KVM, and LXC, are summarized in Table 5.2. Note that each virtual-
 2052 ization solution imposes its own unique requirements on the base operating system.
 2053 Hence, Xen’s Linux kernel refers to the Domain-0 kernel, whereas KVM’s Linux kernel
 2054 represents the actual running kernel hosting the KVM hypervisor. Linux Containers
 2055 share a single kernel between the host and guests, and VMWare ESXi does not rely
 2056 on a Linux kernel at all.

2057 Similarly, hypervisor requirements prevented us from standardizing on a single
 2058 host operating system. For Xen and KVM, we relied on the Arch Linux 2013.10.01
 2059 distribution because it provides easy access to the mainline Linux kernel. For our
 2060 LXC tests, we use CentOS 6.4 because its shared kernel was identical to the base
 2061 CentOS 6.4 kernel used in our testing. VMWare has a proprietary software stack.
 2062 All of this makes comparison challenging, but as we describe in Section 5.4.2, we are
 2063 running a common virtual machine across all experiments.

2064 5.4.2 Guest Configuration

2065 We treat each hypervisor as its own system, and compare virtual machine guests
2066 to a base CentOS 6.4 system. The base system and the guests are all composed of
2067 CentOS 6.4 installation with a 2.6.32-358.23.2 stock kernel and CUDA version 5.5.
2068 Each guest is allocated 20 GB of RAM and a full CPU socket (either 6 or 8 CPU
2069 cores). Bespin experiments received 8 cores and Delta experiments received 6 cores.
2070 VMs were restricted to a single NUMA node. On the Bespin system, the K20m GPU
2071 was attached to NUMA node 0. On the Delta system, the C2075 GPU was attached
2072 to NUMA node 1. Hence VMs ran on NUMA node 0 for the Bespin experiments,
2073 and node 1 for the Delta experiments.

2074 5.4.3 Microbenchmarks

2075 Our experiments are composed of a mix of microbenchmarks and application-level
2076 benchmarks, as well as a combination of CUDA and OpenCL benchmarks. The
2077 SHOC benchmark suite provides a series of microbenchmarks in both OpenCL and
2078 CUDA [191]. For this analysis, we focus on the OpenCL benchmarks in order to
2079 exercise multiple programming models. Benchmarks range from low-level peak Flops
2080 and bandwidth measurements, to kernels and mini-applications.

2081 5.4.4 Application Benchmarks

2082 For our application benchmarks, we have chosen the LAMMPS molecular dynam-
2083 ics simulator [192], the GPU-LIBSVM [193], and the LULESH shock hydrodynamics
2084 simulator [194]. These represent a range of computational characteristics, from com-
2085 putational physics to big data analytics, and are representative of GPU-accelerated
2086 applications in common use.

2087 **LAMMPS** The Large-scale Atomic/Molecular Parallel Simulator (LAMMPS), is a
2088 parallel molecular dynamics simulator [192, 195] used for production MD simulation
2089 on both CPUs and GPUs [196]. LAMMPS has two packages for GPU support, the
2090 USER-CUDA and GPU packages. With the USER-CUDA package, each GPU is used
2091 by a single CPU, whereas the GPU package allows multiple CPUs to take advantage
2092 of a single GPU. There are performance trade-offs with both approaches, but we chose
2093 to use the GPU package in order to stress the virtual machine by exercising multiple
2094 CPUs. Consistent with the existing GPU benchmarking approaches, our results are
2095 based on the Rhodopsin protein.

2096 **GPU-LIBSVM** LIBSVM is a popular implementation [197] of the machine learn-
2097 ing classification algorithm support vector machine (SVM). GPU-accelerated LIB-
2098 SVM [193] enhances LIBSVM by providing GPU-implementations of the kernel ma-
2099 trix computation portion of the SVM algorithm for radial basis kernels. For bench-
2100 marking purposes we use the NIPS 2003 feature extraction gisette data set. This data
2101 set has a high dimensional feature space and large number of training instances, and
2102 these qualities are known to be computational intensive to generate SVM models.
2103 The GPU-accelerated SVM implementation shows dramatic improvement over the
2104 CPU-only implementation.

2105 **LULESH** Hydrodynamics is widely used to model continuum properties and inter-
2106 actions in materials when there is an applied force [198]. Hydrodynamics applications
2107 consume approximately one third of the runtime of data center resource throughout
2108 the U.S. DoD (Department of Defense). The Livermore Unstructured Lagrange Ex-
2109 plicit Shock Hydro (LULESH) was developed by Lawrence Livermore National Lab
2110 as one of five challenge problems in the DARPA UHPC program. LULESH is widely
2111 used as a proxy application in the U.S. DOE (Department of Energy) co-design effort

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

²¹¹² for exascale applications [194].

²¹¹³ 5.5 Performance Results

²¹¹⁴ We characterize GPGPU performance within virtual machines across two hardware
²¹¹⁵ systems, 4 hypervisors, and 3 application sets. We begin with the SHOC benchmark
²¹¹⁶ suite before describing the GPU-LIBSVM, LAMMPS, and LULESH results. All
²¹¹⁷ benchmarks are run 20 times and averaged. Results are scaled with respect to a base
²¹¹⁸ CentOS 6.4 system for both systems. That is, we compare virtualized Bespin per-
²¹¹⁹ formance to non-virtualized Bespin performance, and virtualized Delta perform-
²¹²⁰ ance to non-virtualized Delta performance. Values less than 1 indicate that the base sys-
²¹²¹ tem outperformed the virtual machine, while values greater than 1 indicate that the
²¹²² virtual machine outperformed the base system. In cases where we present geometric

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

2123 means across multiple benchmarks, the means are taken over these scaled values, and
2124 the semantics are the same: less than 1 indicates overhead in the hypervisor, greater
2125 than 1 indicates a performance increase over the base system.

2126 5.5.1 SHOC Benchmark Performance

2127 SHOC splits its benchmarks into 3 Levels, named Levels 0 through 2. Level 0 repre-
2128 sents device-level characteristics, peak Flop/s, bandwidth, etc. Level 1 characterizes
2129 computational kernels: FFT and matrix multiplication, among others. Finally, Level
2130 2 includes “mini-applications,” in this case an implementation of the S3D, a compu-
2131 tational chemistry application.

2132 Because the SHOC OpenCL benchmarks report more than 70 individual mi-
2133 crobenchmarks, space does not allow us to show each benchmark individually. In-

2134 stead, we start with a broad overview of SHOC’s performance across all benchmarks,
2135 hypervisors, and systems. We then discuss in more detail those benchmarks that
2136 either outperformed or underperformed the Bespin (K20) system by 0.50% or more.
2137 We call these benchmarks outliers. As we will show, those outlier benchmarks iden-
2138 tified on the Bespin system, also tend to exhibit comparable characteristics on the
2139 Delta system as well, but the overhead is typically higher.

2140 In Tables 5.3 and 5.4, we provide geometric means for each SHOC level across each
2141 hypervisor and system. We also include the maximum overhead for each hypervisor
2142 at each level to facilitate comparison across hypervisors and systems. Finally, we
2143 provide a comparable breakdown of only the PCIe SHOC benchmarks, based on the
2144 intuition that PCIe-specific benchmarks will likely result in higher overhead.

2145 At a high level, we immediately notice that in the cases of KVM and LXC, both
2146 perform very near native across both the Bespin and Delta platforms. On average,
2147 these systems are almost indistinguishable from their non-virtualized base systems.
2148 So much so, that experimental noise occasionally boosts performance slightly above
2149 their base systems.

2150 This is in sharp contrast to the Xen and VMWare hypervisors, which perform
2151 well on the Bespin system, but poorly on the Delta system in some cases. This is
2152 particularly evident when looking at the maximum overheads for Xen and VMWare
2153 across both systems. In this case, we see that on the Bespin system, Xen’s maximum
2154 overhead of 3.34% is dwarfed by Delta’s maximum Xen overhead of 24.0%. VMWare
2155 exhibits similar characteristics, resulting in a maximum overhead of 1.9% in the case
2156 of the Bespin system, and a surprising 36.6% in the case of the Delta system. We
2157 provide a more in-depth discussion of these overheads below.

2158 Of the Level 0 benchmarks, only four exhibited notable overhead in the case of the
2159 Bespin system: bspeed_download, lmem_readbw, tex_readbw, and ocl_queue. These

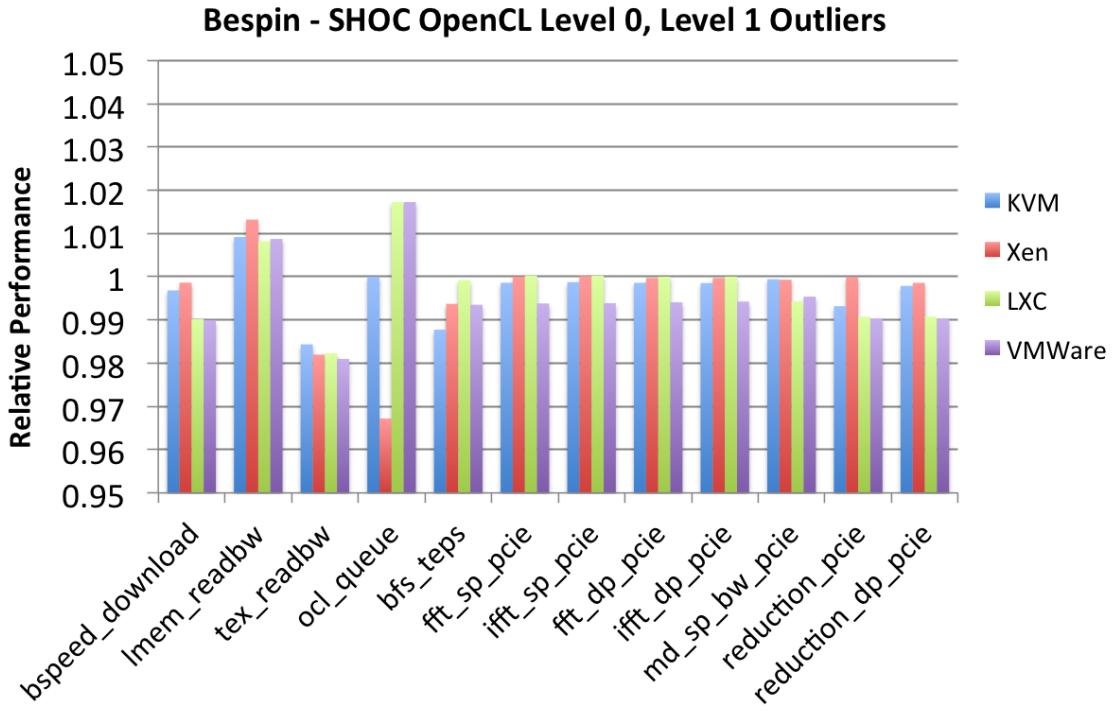


Figure 5.1 SHOC Levels 0 and 1 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

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

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.

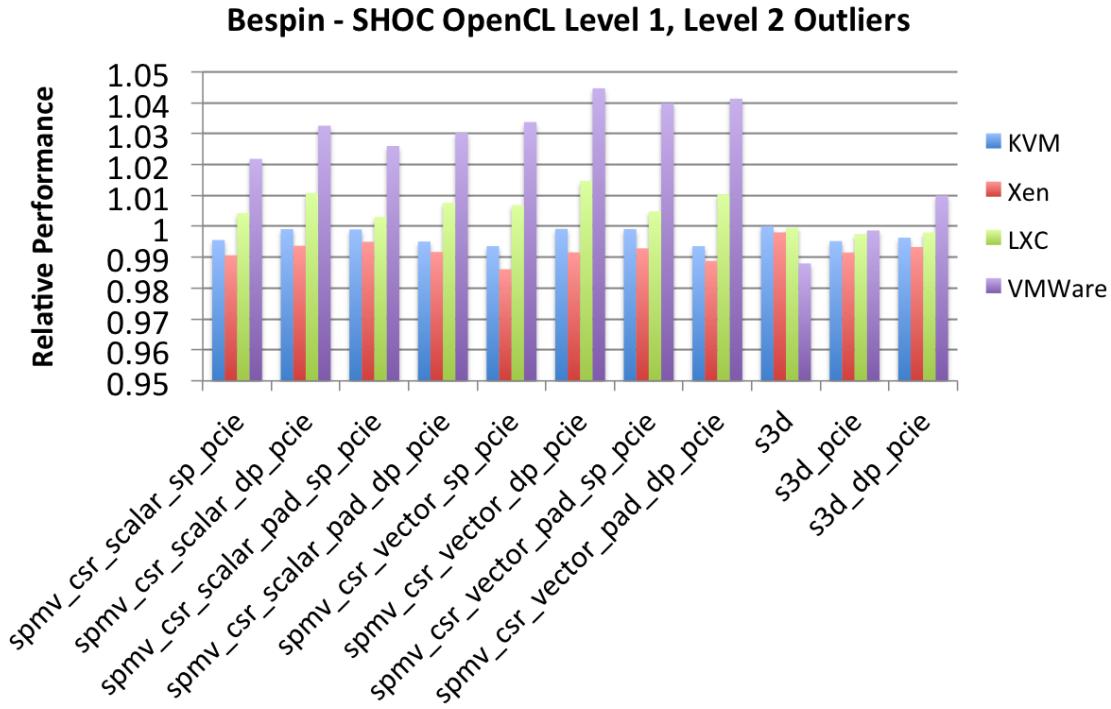


Figure 5.2 SHOC Levels 1 and 2 relative performance on Bespin system. Results show benchmarks over or under-performing by 0.5% or greater. Higher is better.

The remainder of Figure 5.1 includes a series of SHOC's Level 1 benchmarks, representing computational kernels. This includes BFS, FFT, molecular dynamics, and reduction kernels. Notably, nearly all of the benchmarks exhibiting overhead are the PCIe portion of SHOC's benchmarks. This is unsurprising, since the Level 0 benchmarks suggest PCIe bandwidth as the major source overhead. Still, results remain consistent with the bspeed_download overhead observed in the Level 0 benchmarks, further suggesting that host/device data movement is the major source of overhead.

In Figure 5.2 we present the remaining SHOC Level 1 results as well as the SHOC Level 2 results (S3D). In these results we begin to see an interesting trend, namely that VMWare consistently outperforms the base system in the Spmv and S3D microbenchmarks on the Bespin system. We believe this to be a performance regression

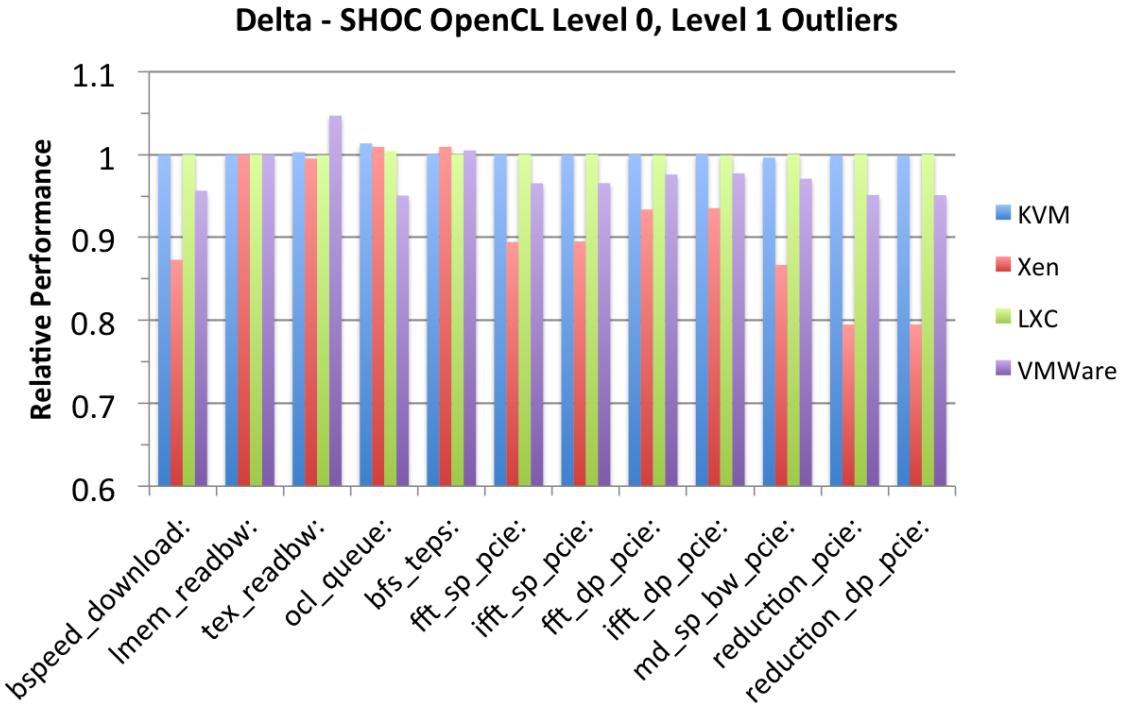


Figure 5.3 SHOC Levels 0 and 1 relative performance on Delta system. Benchmarks shown are the same as Bespin's. Higher is better.

in CentOS 6.4, rather than a unique improvement due to the VMWare ESXi hypervisor. When running the benchmarks on a non-virtualized Bespin system with Arch Linux, the VMWare ESXi performance gains were erased. An interesting finding of this was that Spmv was unique in this way - no other benchmarks were affected by this performance issue.

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

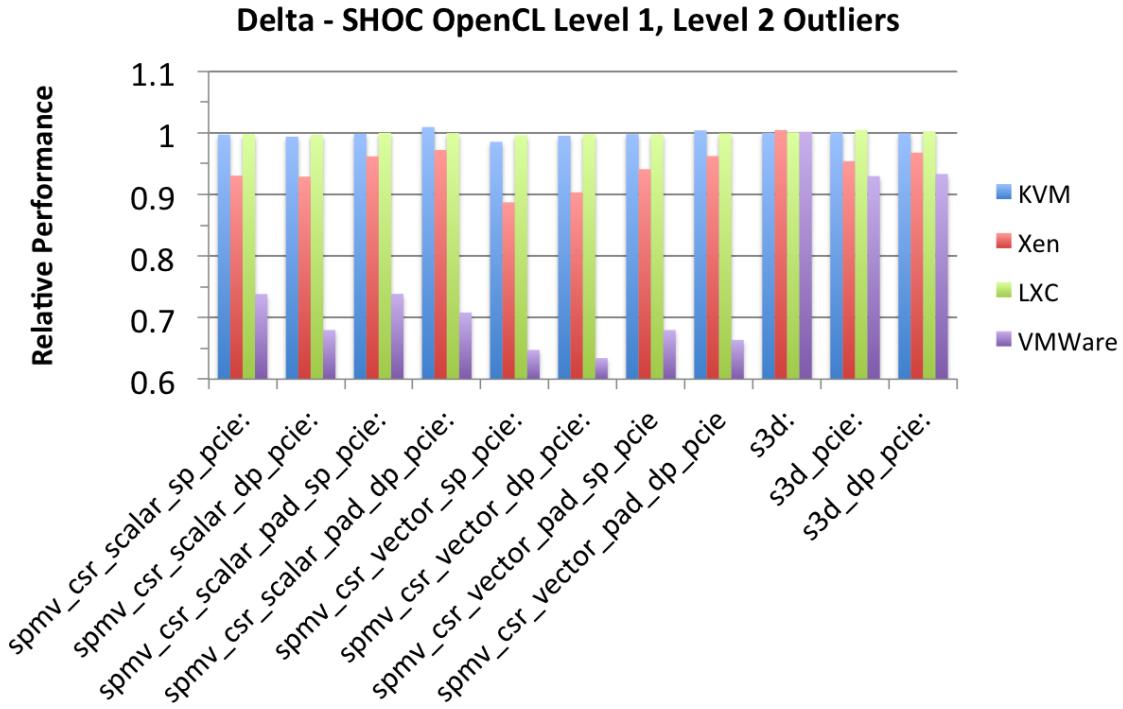


Figure 5.4 SHOC Levels 1 and 2 relative performance. Benchmarks shown are the same as Bespin's. Higher is better.

2194 for VMWare.

2195 On further examination, we determined that Xen was unable to activate IOMMU
 2196 large page tables on the Delta system. KVM successfully allocated 4k, 2M, and 1G
 2197 page table sizes, while Xen was limited to size 4k page tables. The Bespin system
 2198 was able to take advantage of 4k, 2M, and 1G page sizes on both KVM and Xen. It
 2199 appears that this issue is correctable and does not represent a fundamental limitation
 2200 to the Xen hypervisor on the Nehalem/Westmere microarchitecture. While as a
 2201 closed source product, we have limited insight into VMWare ESXi, we speculate that
 2202 VMWare may be experiencing a similar issue on the Delta system and not on our
 2203 Bespin system.

2204 In light of this, we broadly find that virtualization overhead across hypervisors and
 2205 architectures is minimal. Questions remain as to the source of the exceptionally high

overhead in the case of Xen and VMWare on the Delta system, but because KVM shows no evidence of this overhead, we believe the Westmere/Fermi architecture to be suitable for VGA passthrough in a cloud environment. In the case of the Bespin system, it is clear that VGA passthrough can be achieved across hypervisors with virtually no overhead.

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

5.5.2 GPU-LIBSVM Performance

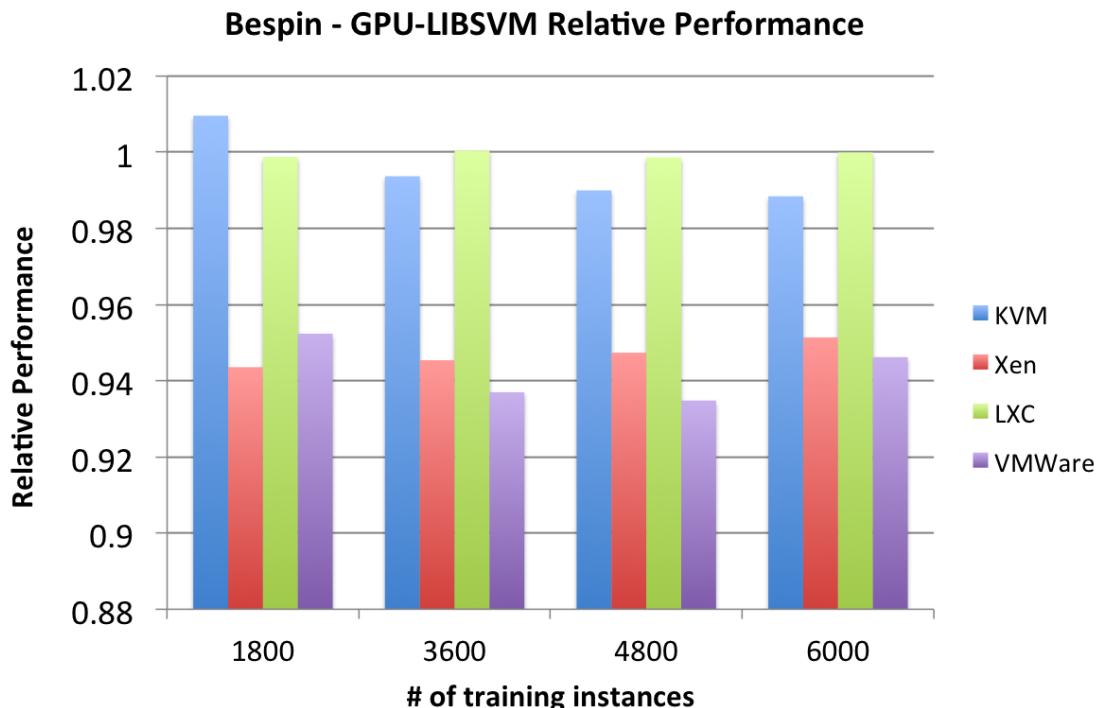


Figure 5.5 GPU-LIBSVM relative performance on Bespin system. Higher is better.

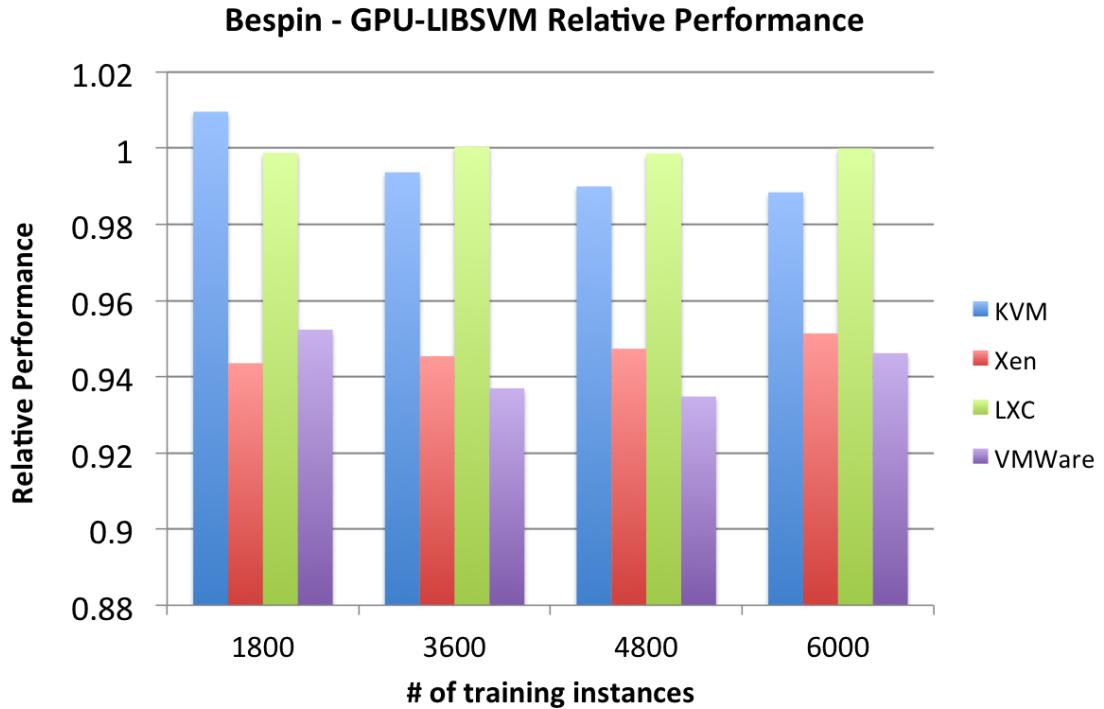


Figure 5.6 GPU-LIBSVM relative performance on Delta showing improved performance within virtual environments. Higher is better.

2217 In Figures 5.5 and 5.6, we display our GPU-LIBSVM performance results for the
 2218 Bespin (K20m) and Delta (C2075) systems. Using the NIPS 2003 gisette data set,
 2219 we show the performance across 4 problems sizes, ranging from 1800 to 6000 training
 2220 instances. The NIPS 2003 gisette dataset is a standard dataset that was used as a
 2221 part of the NIPS 2003 feature selection challenge.

2222 KVM again performs well across both the Delta and Bespin systems. In the case
 2223 of the Delta system, in fact, KVM, significantly outperforms the base system. We
 2224 determined this to be caused by KVM's support for transparent hugepages. When
 2225 sufficient memory is available, transparent hugepages may be used to back the en-
 2226 tirety of the guest VM's memory. Hugepages have previously been shown to improve
 2227 TLB performance for KVM guests, and have been shown to occasionally boost the
 2228 performance of a KVM guests beyond its host [199]. After disabling hugepages on the

2229 KVM host for the Delta system, performance dropped to 80–87% of the base system,
2230 suggesting that memory optimizations such as transparent hugepages can substan-
2231 tially improve the performance of virtualized guests under some circumstancs. LXC
2232 and VMWare perform close to the base system, while Xen achieves between 72–90%
2233 of the base system’s performance. We speculate that this may be related to Xen’s
2234 inability to enabled page sizes larger than 4k.

2235 **5.5.3 LAMMPS Performance**

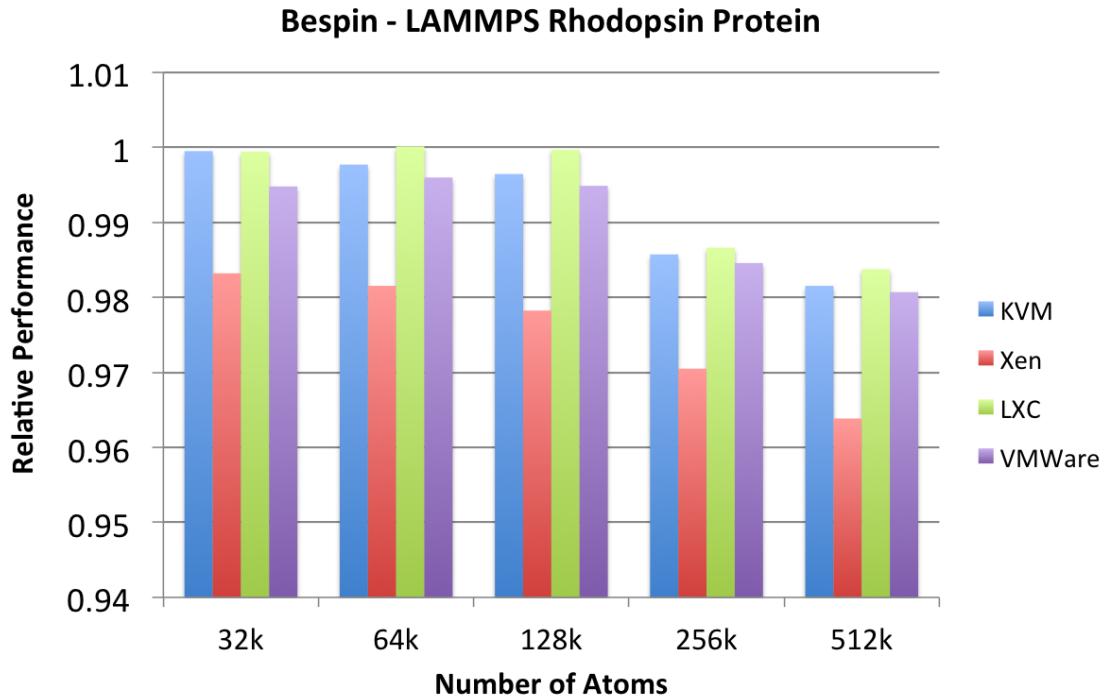


Figure 5.7 LAMMPS Rhodopsin benchmark relative performance for Bespin system. Higher is better.

2236 In Figures 5.7 and 5.8, we show the LAMMPS Rhodopsin protein simulation
2237 results. LAMMPS is unique among our benchmarks, in that it exercises both the GPU
2238 and multiple CPU cores. In keeping with the LAMMPS benchmarking methodology,
2239 we execute the benchmark using 1 GPU and 1–8 CPU cores on the Bespin system,

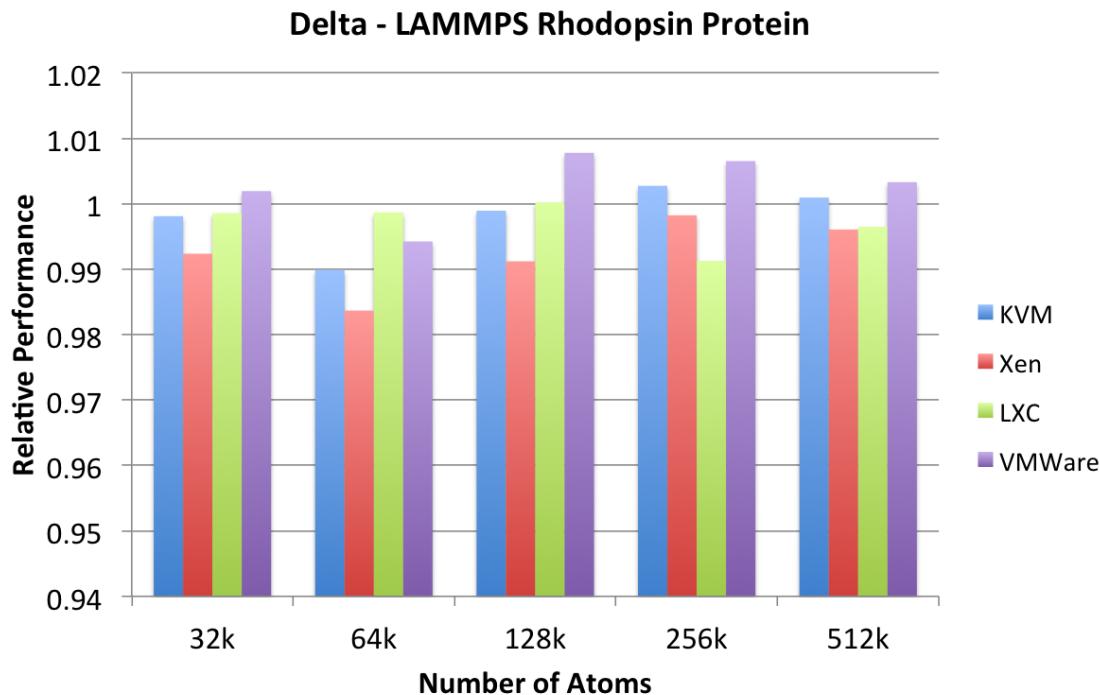


Figure 5.8 LAMMPS Rhodopsin benchmark relative performance for Delta system. Higher is better.

2240 selecting the highest performing configuration. In the case of the Delta system, we
 2241 execute the benchmark on 1–6 cores and 1 GPU, selecting the highest performing
 2242 configuration.

2243 Overall, LAMMPS performs well across both hypervisors and systems. Surpris-
 2244 ingly, LAMMPS showed better efficiency on the Delta system than the Bespin system,
 2245 achieving greater than 98% efficiency across the board, while Xen on the Bespin sys-
 2246 tem occasionally drops as low as 96.5% efficiency.

2247 This performance is encouraging because it suggests that even heterogeneous CPU
 2248 + GPU code is capable of performing well in a virtualized environment. Unlike SHOC,
 2249 GPU-LIBSVM, and LULESH, LAMMPS fully exercises multiple host CPUs, splitting
 2250 work between one or more GPUs and one or more CPU cores. This has the potential
 2251 to introduce additional performance overhead, but the results do not bear this out in

2252 the case of LAMMPS.

2253 **5.5.4 LULESH Performance**

2254 In Figure 5.9 we present our LULESH results for problem sizes ranging from mesh
2255 resolutions of $N = 30$ to $N = 150$. LULESH is a highly compute-intensive simulation,
2256 with limited data movement between the host/virtual machine and the GPU, making
2257 it ideal for GPU acceleration. Consequently, we would expect little overhead due
2258 to virtualization. We show LULESH results only on the Bespin system, because
2259 differences in the code bases between the Kepler and Fermi implementations led to
2260 unsound comparisons.

2261 While, overall, we see very little overhead, there is a slight scaling effect that
2262 is most apparent in the case of the Xen hypervisor. As the mesh resolution (N^3)
2263 increases from $N = 30$ to $N = 150$, we see that the Xen overhead decreases until Xen
2264 performs on-par with KVM, LXC, and VMWare.

2265 **5.6 Lessons Learned**

2266 Virtualizing performance-critical workloads has always proven controversial, whether
2267 the workload is CPU-only [171] or CPU with GPU. From our Westmere results, we
2268 can see that this criticism is in part legitimate, at times resulting in a performance
2269 penalty of greater than 10%, especially in the case of the VMWare ESXi and Xen
2270 hypervisors. We believe much of this to be fixable, especially in the case of the Xen
2271 hypervisor.

2272 At the same time, however, we have shown that the Sandy Bridge processor genera-
2273 tion has nearly erased those performance overheads, suggesting that old arguments
2274 against virtualization for performance-critical tasks should be reconsidered. In light of

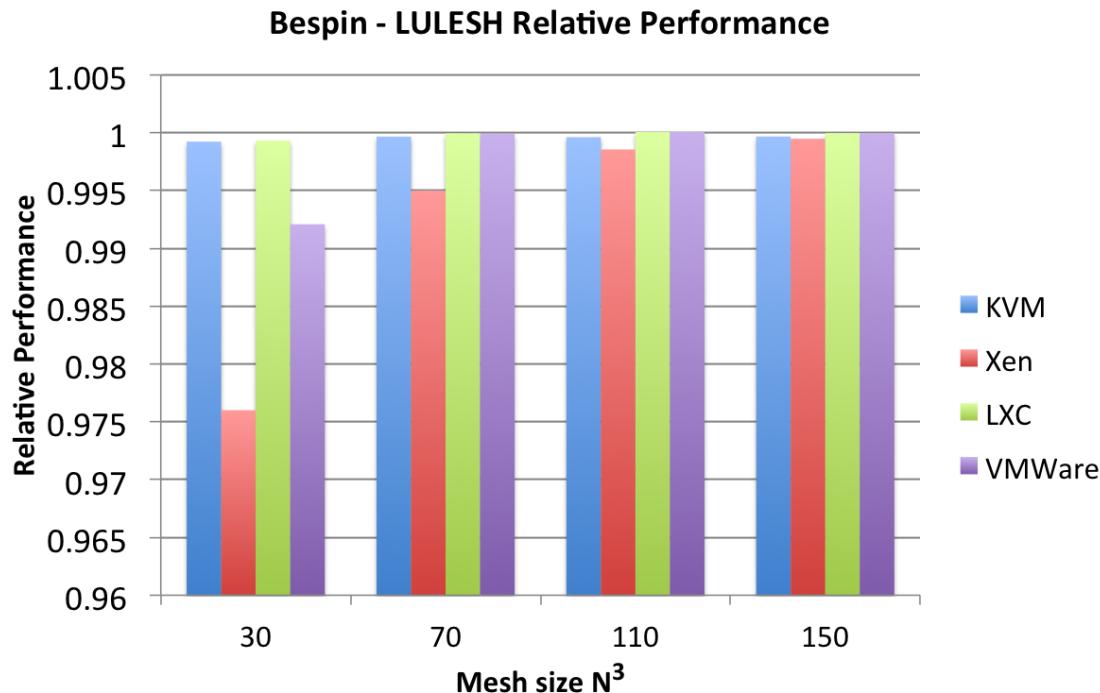


Figure 5.9 LULESH relative performance on Bespin. Higher is better.

2275 this, the primary lesson from this study is that VGA-passthrough to virtual machines
 2276 is achievable at low overhead, and across a variety of hypervisors and virtualization
 2277 platforms. Virtualization performance remains inconsistent across hypervisors for the
 2278 Westmere generation of processors, but starting with the Sandy Bridge architecture,
 2279 performance and consistency increase dramatically. In the case of the Sandy Bridge
 2280 architecture, even the lowest performing hypervisor, open source Xen, typically per-
 2281 forms within 95% of the base case.

2282 This study has also yielded valuable insight into the merits of each hypervisor.
 2283 KVM consistently yielded near-native performance across the full range of bench-
 2284 marks. Its support for transparent hugepages resulted in slight performance boosts
 2285 over-and-above even the base CentOS system in the case of the Delta system.

2286 VMWare's performance proved inconsistent across architectures, performing well

2287 in the case of Bespin, and relatively poorly in the case of the Delta system. Because
2288 hypervisor configurations were identical across systems, we can only speculate that
2289 VMWare’s performance is aided by the virtualization improvements offered by the
2290 Sandy Bridge microarchitecture.

2291 The Xen hypervisor was consistently average across both architectures, perform-
2292 ing neither poorly nor extraordinarily well in any individual benchmark. Xen and
2293 VMWare ESXi are the only two hypervisors from this study that officially support
2294 VGA passthrough. As a result, PCI passthrough support in both Xen and VMWare is
2295 more robust than KVM. We expect that this advantage will not last long, as commer-
2296 cial solutions targeting PCI passthrough in KVM are becoming common, particularly
2297 with regard to SR-IOV and networking adapters.

2298 Linux Containers (LXC), consistently performed closest to the native case. This,
2299 of course, is not surprising given that LXC guests share a single kernel with their
2300 hosts. This performance comes at the cost of both flexibility and security, however.
2301 LXC is less flexible than its full virtualization counterparts, offering support for only
2302 Linux guests. More importantly, LXC device passthrough has security implications
2303 for multi-GPU systems. In the case of a multi-GPU-enabled host with NVIDIA
2304 hardware, both GPUs must be passed to the LXC guest in order to initialize the
2305 driver. This limitation may be addressable in future revisions to the NVIDIA driver.

2306 5.7 Directions for Future Work

2307 In this chapter we have characterized the performance of 4 common hypervisors across
2308 two generations of GPUs and two host microarchitectures, and across 3 sets of bench-
2309 marks. We showed the dramatic improvement in virtualization performance between
2310 the Fermi/Westmere and the Kepler/Sandy Bridge system, with the Sandy Bridge

2311 system typically performing within 1% of the base system. Finally, this study sought
2312 to characterize the GPU and CPU+GPU performance with carefully tuned hypervi-
2313 sor and guest configurations, especially with respect to NUMA. Improvements must
2314 be made to today’s hypervisors in order to improve virtual NUMA support. Finally,
2315 cloud infrastructure, such as OpenStack, must be capable of automatically allocating
2316 virtual machines in a NUMA-friendly manner in order to achieve acceptable results
2317 at cloud-scale.

2318 The next step in this work is to move beyond the single node to show that clus-
2319 ters of accelerators can be efficiently used with minimal overhead. This will require
2320 studies in high speed networking, particularly SR-IOV-enabled ethernet and Infini-
2321 band. Special attention is needed to ensure that latencies remain tolerable within
2322 virtual environments. Some studies have begun to examine these issues [200], but
2323 open questions remain.

2324 **Chapter 6**

2325 **Supporting High Performance
2326 Molecular Dynamics in Virtualized
2327 Clusters using IOMMU, SR-IOV,
2328 and GPUDirect**

2329 **6.1 Abstract**

2330 Cloud infrastructure-as-a-Service paradigms have recently shown their utility for a
2331 vast array of computational problems, ranging from advanced web service archi-
2332 tectures to high throughput computing. However, many scientific computing ap-
2333 plications have been slow to adapt to virtualized cloud frameworks. This is due
2334 to performance impacts of virtualization technologies, coupled with the lack of ad-
2335 vanced hardware support necessary for running many high performance scientific
2336 applications at scale.

2337 By using KVM virtual machines that leverage both Nvidia GPUs and InfiniBand,

2338 we show that molecular dynamics simulations with LAMMPS and HOOMD run at
2339 near-native speeds. This experiment also illustrates how virtualized environments
2340 can support the latest parallel computing paradigms, including both MPI+CUDA
2341 and new GPUDirect RDMA functionality. Specific findings show initial promise in
2342 scaling of such applications to larger production deployments targeting large scale
2343 computational workloads.

2344 6.2 Introduction

2345 At present we stand at the inevitable intersection between High Performance Com-
2346 puting (HPC) and clouds. Various platform tools such as Hadoop and MapReduce,
2347 among others, have already percolated into data intensive computing within HPC [26].
2348 In addition, there are efforts to support traditional HPC-centric scientific computing
2349 applications in virtualized cloud infrastructure. There are a multitude of reasons for
2350 supporting parallel computation in the cloud [58], including features such as dynamic
2351 scalability, specialized operating environments, simple management interfaces, fault
2352 tolerance, and enhanced quality of service, to name a few. The growing importance
2353 of supporting advanced scientific computing using virtualized infrastructure can be
2354 seen by a variety of new efforts, including the NSF-funded Comet resource part of
2355 XSEDE at San Diego Supercomputer Center [201].

2356 Nevertheless, there exists a past notion that virtualization used in today's cloud
2357 infrastructure is inherently inefficient. Historically, cloud infrastructure has also done
2358 little to provide the necessary advanced hardware capabilities that have become al-
2359 most mandatory in supercomputers today, most notably advanced GPUs and high-
2360 speed, low-latency interconnects. The result of these notions has hindered the use
2361 of virtualized environments for parallel computation, where performance must be

2362 paramount.

2363 A growing effort is currently underway that looks to systematically identify and
2364 reduce any overhead in virtualization technologies. This effort has, thus far, proven
2365 to be a qualified success [171, 202], though further research is needed to address
2366 issues of scalability and I/O. Thus, we see a constantly diminishing overhead with
2367 virtualization, not only with traditional cloud workloads [203] but also with HPC
2368 workloads. While virtualization will almost always include some additional overhead
2369 in relation to its dynamic features, the eventual goal for supporting HPC in virtualized
2370 environments is to minimize what overhead exists whenever possible. To advance
2371 the placement of HPC applications on virtual machines, new efforts are emerging
2372 which focus specifically on key hardware now commonplace in supercomputers. By
2373 leveraging new virtualization tools such as IOMMU device passthrough and SR-IOV,
2374 we can now support the such advanced hardware as the latest Nvidia Tesla GPUs [204]
2375 as well as InfiniBand fabrics for high performance networking and I/O [205, 206].

2376 With the advances in hypervisor performance coupled with the newfound avail-
2377 ability of HPC hardware in virtual machines analogous to the most powerful super-
2378 computers used today, we see can see the possibility of a high performance cloud in-
2379 frastructure using virtualization. While our previous efforts in this area have focused
2380 on single-node advancements, it is now imperative to ensure real-world applications
2381 can also operate in distributed environments as found in today’s cluster and cloud
2382 infrastructures.

2383 Efforts to improve power efficiency and performance in data centers has led to
2384 more heterogeneous architectures. That move toward heterogeneity has, in turn, led
2385 to support for heterogeneity in the cloud. For example, Amazon EC2 supports GPU
2386 accelerators in EC2 [207], and OpenStack supports heterogeneity using flavors [208].
2387 These advancements in cloud-level support for heterogeneity combined with better

2388 support for high-performance virtualization makes the use of cloud for HPC much
2389 more feasible for a wider range of applications and platforms.

2390 In this paper we describe background a related work. Then, we describe a hetero-
2391 geneous cloud platform, based on OpenStack. This effort has been under development
2392 at USC/ISI since 2011 [165]. We describe our work towards integrating GPU and
2393 InfiniBand support into OpenStack, and we describe the heterogeneous scheduling
2394 additions that are necessary to support not only attached accelerators, but any cloud
2395 composed of heterogeneous elements.

2396 We then demonstrate running two molecular dynamics simulations, LAMMPS
2397 and HOOMD, in a virtual infrastructure complete with both Kepler GPUs and QDR
2398 InfiniBand. Both HOOMD and LAMMPS are used extensively in some of the world's
2399 fastest supercomputers and represent example simulations that HPC supports today.
2400 We show that these applications are able to run at near-native speeds within a com-
2401 pletely virtualized environment, demonstrating just small performance impacts that
2402 are usually acceptable by many users. Furthermore, we demonstrate the ability of
2403 such a virtualized environment to support cutting edge software tools such as RDMA
2404 GPUDirect, illustrating how cutting-edge HPC technologies are also possible in a
2405 virtualized environment.

2406 Following these efforts, we hope to ensure upstream infrastructure projects such
2407 as OpenStack [130, 209] are able to make effective and quick use of these features,
2408 allowing users to build private cloud infrastructure to support high performance dis-
2409 tributed computational workloads.

2410 6.3 Background and Related Work

2411 Virtualization technologies and hypervisors have been seen widespread deployment
2412 in support of a vast array of applications. This ranges from public commercial Cloud
2413 deployments such as Amazon EC2 [210, 211], Microsoft Azure [212], and Google’s
2414 Cloud Platform [213] to private deployments within colocation facilities, corporate
2415 data centers, and even national scale cyber infrastructure initiatives. All these sup-
2416 port look to support various use cases and applications such as web servers, ACID
2417 and BASE databases, online object storage, and even distributed systems, to name a
2418 few.

2419 The use of virtualization and hypervisors specifically support various HPC so-
2420 lutions has been studied with mixed results. In [171], it is found that there is a
2421 great deal of variance between hypervisors when running various distributed memory
2422 and MPI applications, finding that KVM overall performed well across an array of
2423 HPC benchmarks. Furthermore, some applications may not fit well into default
2424 virtualized environments, such as High Performance Linpack [202]. Other studies
2425 have specifically looked at interconnect performance in virtualization and found the
2426 best-case scenario to be lacking [214] with up to 60% performance penalties with
2427 conventional techniques.

2428 Recently, various CPU architectures have added support for I/O virtualization
2429 mechanisms in the CPU ISA through the use of an I/O memory management unit
2430 (IOMMU). Often, this is referred to as PCI Passthrough, as it enabled devices on the
2431 PCI-Express bus to be passed directly to a specific virtual machine (VM). Specific
2432 hardware implementations include Intel’s VT-d [215], AMD’s IOMMU [216] from
2433 x86_64 architectures, and even more recently ARM System MMU [217]. All of these
2434 implementations effectively look to aid in the usage of DMA-capable hardware to be

2435 used within a specific virtual machine. Using these features, a wide array of hardware
2436 can be utilized directly within VMs and enable fast and efficient computation and
2437 I/O capabilities.

2438 With PCI Passthrough, a PCI device is handed directly to a running (or booting)
2439 VM, thereby relinquishing control of the device within the host entirely. This is
2440 different from typical VM usage where hardware is emulated in the host and used
2441 in a guest VM, such as with bridged ethernet adapters or emulated VGA devices.
2442 Performing PCI Passthrough requires the host to seize the device upon boot using a
2443 specialized driver to effectively block normal driver initialization. In the instance of
2444 the KVM hypervisor, this is done using the *vfio* and *pci_stub* drivers. Then, this driver
2445 relinquishes control to the VM, whereby normal device drivers initiate the hardware
2446 and enable the device for use by the guest OS.

2447 **6.3.1 GPU Passthrough**

2448 Nvidia GPUs comprise the single most common accelerator in the Nov 2014 Top 500
2449 List [8] and represent an increasing shift towards accelerators for HPC applications.
2450 Historically, GPU usage in a virtualized environment has been difficult, especially
2451 for scientific computation. Various front-end remote API implementations have been
2452 developed to provide CUDA and OpenCL libraries in VMs, which translate library
2453 calls to a back-end or remote GPU. One common use case of this is rCUDA [55], which
2454 provides a front-end CUDA API within a VM or any compute node, and then sends
2455 the calls via Ethernet or InfiniBand to a separate node with 1 or more GPUs. While
2456 this method is valid, it has the drawback of relying on the interconnect itself and the
2457 bandwidth available, which can be especially problematic on Ethernet. Furthermore,
2458 as this method consumes bandwidth, it can leave little remaining for MPI or RDMA
2459 routines, thereby constructing a bottleneck for some MPI+CUDA applications that

2460 depend on inter-process communication.

2461 Recently efforts have been seen to support such GPU accelerators within VMs
2462 using IOMMU technologies, with implementations now available with KVM [204],
2463 Xen [218] and VMWare [219]. These efforts have shown that GPUs can achieve up
2464 to 99% of their bare metal performance when passed to a virtual machine using PCI
2465 Passthrough. VMWare specifically shows how the such PCI Passthrough solutions
2466 perform well and are likely to outperform front-end Remote API solutions such as
2467 rCUDA within a VM [219]. While these works demonstrate PCI Passthrough perfor-
2468 mance across a range of hypervisors and GPUs, they have been limited to investigating
2469 single node performance until now.

2470 **6.3.2 SR-IOV and InfiniBand**

2471 With almost all parallel HPC applications, the interconnect fabric which enables fast
2472 and efficient communication between processors becomes a central requirement to
2473 achieving good performance. Specifically, a high bandwidth link is needed for dis-
2474 tributed processors to share large amounts of data across the system. Furthermore,
2475 low latency becomes equally important for ensuring quick delivery of small mes-
2476 sage communications and resolving large collective barriers within many parallelized
2477 codes. One such interconnect, InfiniBand, has become the most common implemen-
2478 tation used within the Top500 list. However previously InfiniBand was inaccessible
2479 to virtualized environments.

2480 Supporting I/O interconnects in VMs has been aided by Single Root I/O Virtu-
2481 alization (SR-IOV), whereby multiple virtual PCI functions are created in hardware
2482 to represent a single PCI device. These virtual functions (VFs) can then be passed
2483 to a VM and used as by the guest as if it had direct access to that PCI device. SR-
2484 IOV allows for the virtualization and multiplexing to be done within the hardware,

2485 effectively providing higher performance and greater control than software solutions.

2486 SR-IOV has been used in conjunction with Ethernet devices to provide high perfor-
2487 mance 10Gb TCP/IP connectivity within VMs [220], offering near-native bandwidth
2488 and advanced QoS features not easily obtained through emulated Ethernet offerings.
2489 Currently Amazon EC2 offers a high performance VM solution utilizing SR-IOV en-
2490 abled 10Gb Ethernet adapters. While SR-IOV enabled 10Gb Ethernet solutions offers
2491 a big forward in performance, Ethernet still does not offer the high bandwidth or low
2492 latency typically found with InfiniBand solutions.

2493 Recently SR-IOV support for InfiniBand has been added by Mellanox in the Con-
2494 nectX series adapters. Initial evaluation of SR-IOV InfiniBand within KVM VMs
2495 has proven has found point-to-point bandwidth to be near-native, but up to 30%
2496 latency overhead for very small messages [205, 221]. However, even with the noted
2497 overhead, this still signifies up to an order of magnitude difference in latency between
2498 InfiniBand and Ethernet with VMs. Furthermore, advanced configuration of SR-IOV
2499 enabled InfiniBand fabric is taking shape, with recent research showing up to a 30%
2500 reduction in the latency overhead [206]. However, real application performance has
2501 not yet been well understood until now.

2502 6.3.3 GPUDirect

2503 NVIDIA’s GPUDirect technology was introduced to reduce the overhead of data
2504 movement across GPUs [222, 223]. GPUDirect supports both networking as well as
2505 peer-to-peer interfaces for single node multi-GPU systems. The most recent imple-
2506 mentation of GPUDirect, version 3, adds support for RDMA over InfiniBand for
2507 Kepler-class GPUs.

2508 The networking component of GPUDirect relies on three key technologies: CUDA
2509 5 (and up), a CUDA-enabled MPI implementation, and a Kepler-class GPU (RDMA

2510 only). Both MVAPICH and OpenMPI support GPUDirect. Support for RDMA over
2511 GPUDirect is enabled by the MPI library, given supported hardware, and does not
2512 depend on application-level changes to a user’s code.

2513 In this paper, our GPUDirect work focuses on GPUDirect v3 for multi-node
2514 RDMA support. We demonstrate scaling for up to 4 nodes connected via QDR
2515 InfiniBand and show that GPUDirect RDMA improves both scalability and overall
2516 performance by approximately 9% at no cost to the end user.

2517 6.4 A Cloud for High Performance Computing

2518 With support for GPU Passthrough, SR-IOV, and GPUDirect, we have the building
2519 blocks for a high performance, heterogeneous cloud. In addition, other common
2520 accelerators (e.g. Xeon Phi [224]) have similarly been demonstrated in virtualized
2521 environments. Our vision is of a heterogeneous cloud, supporting both high speed
2522 networking and accelerators for tightly coupled applications.

2523 To this end we have developed a heterogeneous cloud based on OpenStack [130].
2524 In our previous work, we have demonstrated the ability to rapidly provision GPU,
2525 bare metal, and other heterogeneous resources within a single cloud [165]. Building
2526 on this effort we have added support for GPU passthrough to OpenStack as well as
2527 SR-IOV support for both ConnectX-2 and ConnectX-3 Infiniband devices. Mellanox
2528 separately supports an OpenStack InfiniBand networking plugin for OpenStack’s Neu-
2529 tron service [225], however the Mellanox plugin depends on the ConnectX-3 adapter.
2530 Our institutional requirements depend on ConnectX-2 SR-IOV support, requiring
2531 an independent implementation.

2532 OpenStack supports services for networking (Neutron), compute (Nova), identity
2533 (Keystone), storage (Cinder, Swift), and others. Our work focuses entirely on the

2534 compute service.

2535 Scheduling is implemented at two levels: the cloud-level and the node-level. In our
2536 earlier work, we have developed a cloud-level heterogeneous scheduler for OpenStack,
2537 allowing scheduling based on architectures and resources [165]. In this model, the
2538 cloud-level scheduler dispatches jobs to nodes based on resource requirements (e.g.
2539 Kepler GPU) and node-level resource availability.

2540 At the node, a second level of scheduling occurs to ensure that resources are
2541 tracked and not over-committed. Unlike traditional cloud paradigms, devices passed
2542 into VMs cannot be over-committed. We treat devices, whether GPUs or InfiniBand
2543 virtual functions, as schedulable resources. Thus, it is the responsibility of the indi-
2544 vidual node to track resources committed and report availability to the cloud-level
2545 scheduler. For reporting, we piggyback on top of OpenStack’s existing reporting
2546 mechanism to provide a low overhead solution.

2547 6.5 Benchmarks

2548 We selected two molecular dynamics (MD) applications for evaluation in this study:
2549 LAMMPS and HOOMD [226, 227]. These MD simulations are chosen to represent a
2550 subset of advance parallel computation for a number of fundamental reasons:

- 2551 • MD simulations provide a practical representation of N-Body simulations, which
2552 is one of the major computational *Dwarfs* [228] in parallel and distributed com-
2553 puting.
- 2554 • MD simulations are one of the most widely deployed applications on large scale
2555 supercomputers today.
- 2556 • Many MD simulations have a hybrid MPI+CUDA programming model, which

2557 has often become commonplace in HPC as the use of accelerators increases.

2558 As such, we look to LAMMPS and HOOMD to provide a real-world example for
2559 running cutting-edge parallel programs on virtualized infrastructure. While these
2560 applications by no means represent all parallel scientific computing efforts (as justi-
2561 fied by the 13 Dwarfs defined in [228]), we hope these MD simulators offer a more
2562 pragmatic viewpoint than traditional synthetic HPC benchmarks such as High Per-
2563 formance Linpack.

2564 **LAMMPS** The Large-scale Atomic/Molecular Parallel Simulator is a well-understood
2565 highly parallel molecular dynamics simulator. It supports both CPU and GPU-
2566 based workloads. Unlike many simulators, both MD and otherwise, LAMMPS is
2567 heterogeneous. It will use both GPUs and multicore CPUs concurrently. For this
2568 study, this heterogeneous functionality introduces additional load on the host, allow-
2569 ing LAMMPS to utilize all available cores on a given system. Networking in LAMMPS
2570 is accomplished using a typical MPI model. That is, data is copied from the GPU
2571 back to the host and sent over the InfiniBand fabric. No RDMA is used for these
2572 experiments.

2573 **HOOMD-blue** The Highly Optimized Object-oriented Many-particle Dynamics
2574 – Blue Edition is a particle dynamics simulator capable of scaling into the thou-
2575 sands of GPUs. HOOMD supports executing on both CPUs and GPUs. Unlike
2576 LAMMPS, HOOMD is homogeneous and does not support mixing of GPUs and
2577 CPUs. HOOMD supports GPUDirect using a CUDA-enabled MPI. In this paper
2578 we focus on HOOMD’s support for GPUDirect and show its benefits for increasing
2579 cluster sizes.

2580 **6.6 Experimental Setup**

2581 Using two molecular dynamics tools, LAMMPS [226] and HOOMD [227], we demon-
2582 strate a high performance *system*. That is, we combine PCI passthrough for Nvidia
2583 Kepler-class GPUs with QDR Infiniband SR-IOV and show that high performance
2584 molecular dynamics simulations are achievable within a virtualized environment.

2585 For the first time, we also demonstrate Nvidia GPUDirect technology within such
2586 a virtual environment. Thus, we look to not only illustrate that virtual machines
2587 provide a flexible high performance infrastructure for scaling scientific workloads in-
2588 cluding MD simulations, but also that the latest HPC features and programming
2589 environments are also available in this same model.

2590 **6.6.1 Node configuration**

2591 To support the use of Nvidia GPUs and InfiniBand within a VM, specific and exact
2592 host configuration is needed. This node configuration is illustrated in Figure 6.1.
2593 While our implementation is specific to the KVM hypervisor, this setup represents a
2594 design that can be hypervisor agnostic.

2595 Each node in the testbed uses CentOS 6.4 with a 3.13 upstream Linux kernel for
2596 the host OS, along with the latest KVM hypervisor, QEMU 2.1, and the *vfio* driver.
2597 Each Guest VM runs CentOS 6.4 with a stock 2.6.32-358.23.2 kernel. A Kepler GPU
2598 is passed through using PCI Passthrough and directly initiated within the VM via
2599 the Nvidia 331.20 driver and CUDA release 5.5. While this specific implementation
2600 used only a single GPU, it is also possible to include as many GPUs as one can fit
2601 within the PCI Express bus if desired. As the GPU is used by the VM, an on-board
2602 VGA device was used by the host and a standard Cirrus VGA was emulated in the
2603 guest OS.

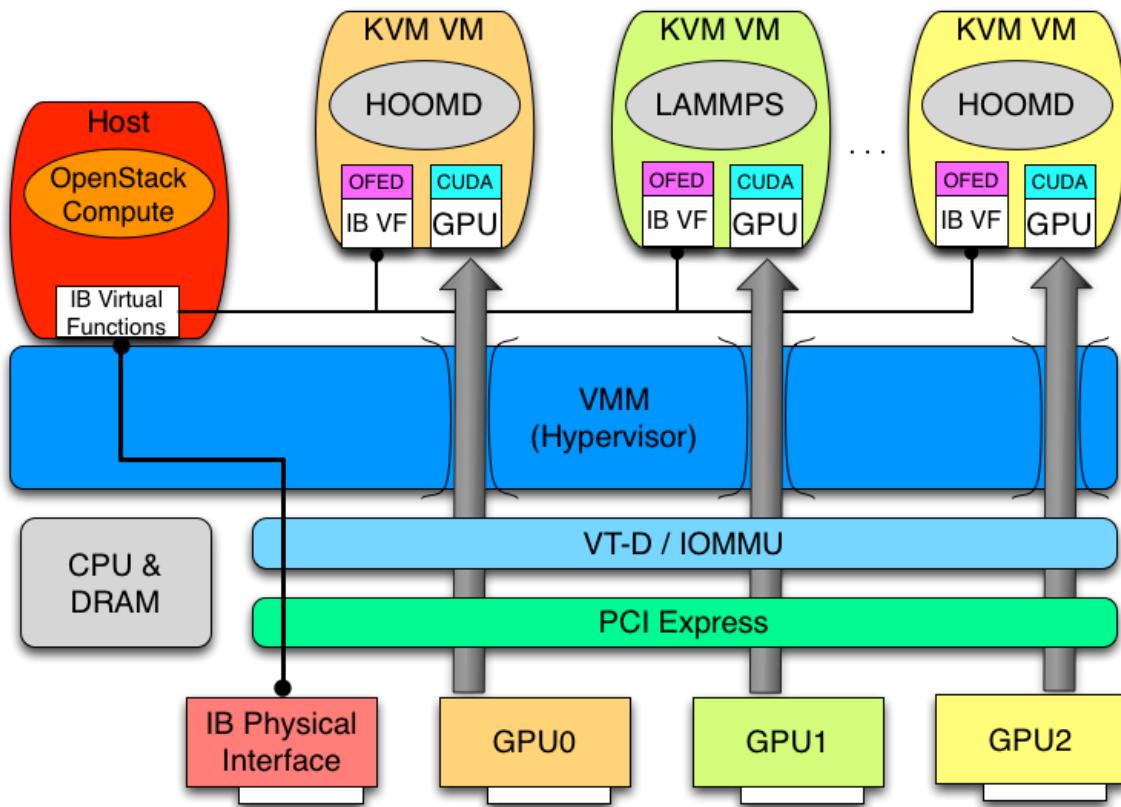


Figure 6.1 Node PCI Passthrough of GPUs and InfiniBand

2604 With using SR-IOV, the OFED drivers version 2.1-1.0.0 are used with Mellanox
 2605 ConnectX-3 VPI adapter with firmware 2.31.5050. The host driver initiates 4 VFs,
 2606 one of which is passed through to the VM where the default OFED mlnx_ib drivers
 2607 are loaded.

2608 6.6.2 Cluster Configuration

2609 Our test environment is composed of 4 servers each with a single Nvidia Kepler-
 2610 class GPU. Two servers are equipped with K20 GPUs, while the other two servers
 2611 are equipped with K40 GPUs, demonstrating the potential for a more heterogeneous
 2612 deployment. Each server is composed of 2 Intel Xeon E5-2670 CPUs, 48GB of DDR3
 2613 memory, and Mellanox ConnectX-3 QDR InfiniBand. CPU sockets and memory are

2614 split evenly between the two NUMA nodes on each system. All InfiniBand adapters
2615 use a single Voltaire 4036 QDR switch with a software subnet manager for IPoIB
2616 functionality.

2617 For these experiments, both the GPUs and InfiniBand adapters are attached to
2618 NUMA node 1 and both the guest VMs and the base system utilized identical software
2619 stacks. Each guest was allocated 20 GB of RAM and a full socket of 8 cores, and
2620 pinned to NUMA node 1 to ensure optimal hardware usage. While all VMs are
2621 capable of login via the InfiniBand IPoIB setup, a 1Gb Ethernet network was used
2622 for all management and login tasks.

2623 For a fair and effective comparison, we also use a native environment without any
2624 virtualization. This native environment employs the same hardware configuration,
2625 and like the Guest OS runs CentOS 6.4 with the stock 2.6.32-358.23.2 kernel.

2626 6.7 Results

2627 In this section, we discuss the performance of both the LAMMPS and HOOMD
2628 molecular dynamics simulation tools when running within a virtualized environment.
2629 Specifically, we scale each application to 32 cores and 4 GPUs, both in a native bare-
2630 metal and virtualized environments. Each application set was run 10 times, with the
2631 results averaged accordingly.

2632 6.7.1 LAMMPS

2633 Figure 6.2 shows one of the most common LAMMPS algorithms used; the Lennard-
2634 Jones potential (LJ). This algorithm is deployed in two main configurations - a 1:1
2635 core to GPU mapping, and a 8:1 core to GPU mapping. With the LAMMPS GPU
2636 implementation, a delicate balance between GPUs and CPUs is required to find the

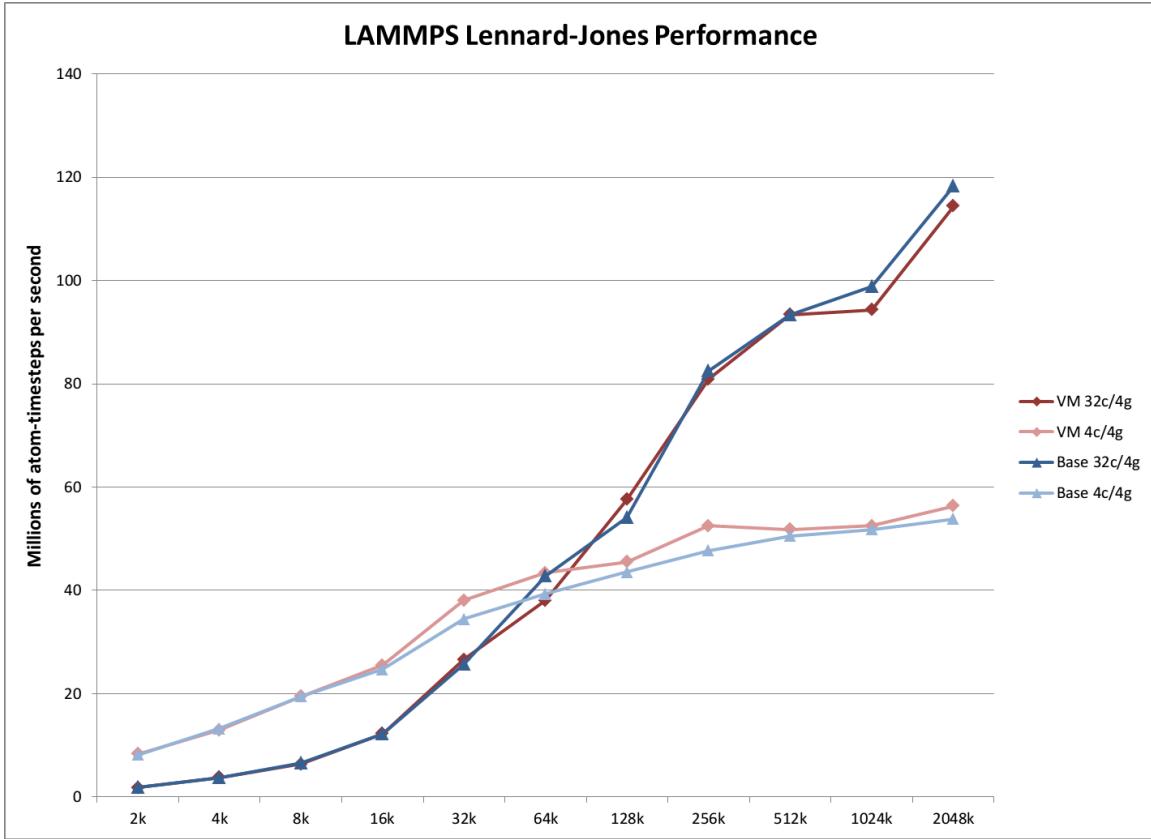


Figure 6.2 LAMMPS LJ Performance

2637 optimal ratio for fastest computation, however here we just look at the two most
 2638 obvious choices. With small problem sizes, the 1:1 mapping outperforms the more
 2639 complex core deployment, as the problem does not require the additional complexity
 2640 provided with multi-core solution. As expected the multi-core configuration quickly
 2641 offers better performance for larger problem sizes, achieving roughly twice the perfor-
 2642 mance with all 8 available cores. This is largely due to the availability of all 8 cores
 2643 to keep the GPU running 100% with continual computation.

2644 The important factor for this manuscript is the relative performance of the virtu-
 2645 alized environment. From the results, it is clear the VM solution performs very well
 2646 compared to the best-case native deployment. For the multi-core configuration across
 2647 all problem sizes, the virtualized deployment averaged 98.5% efficiency compared to

native. The single core per GPU deployment reported better-than native performance at 100% native. This is likely due to caching effects, but further investigation is needed to fully identify this occurrence.

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

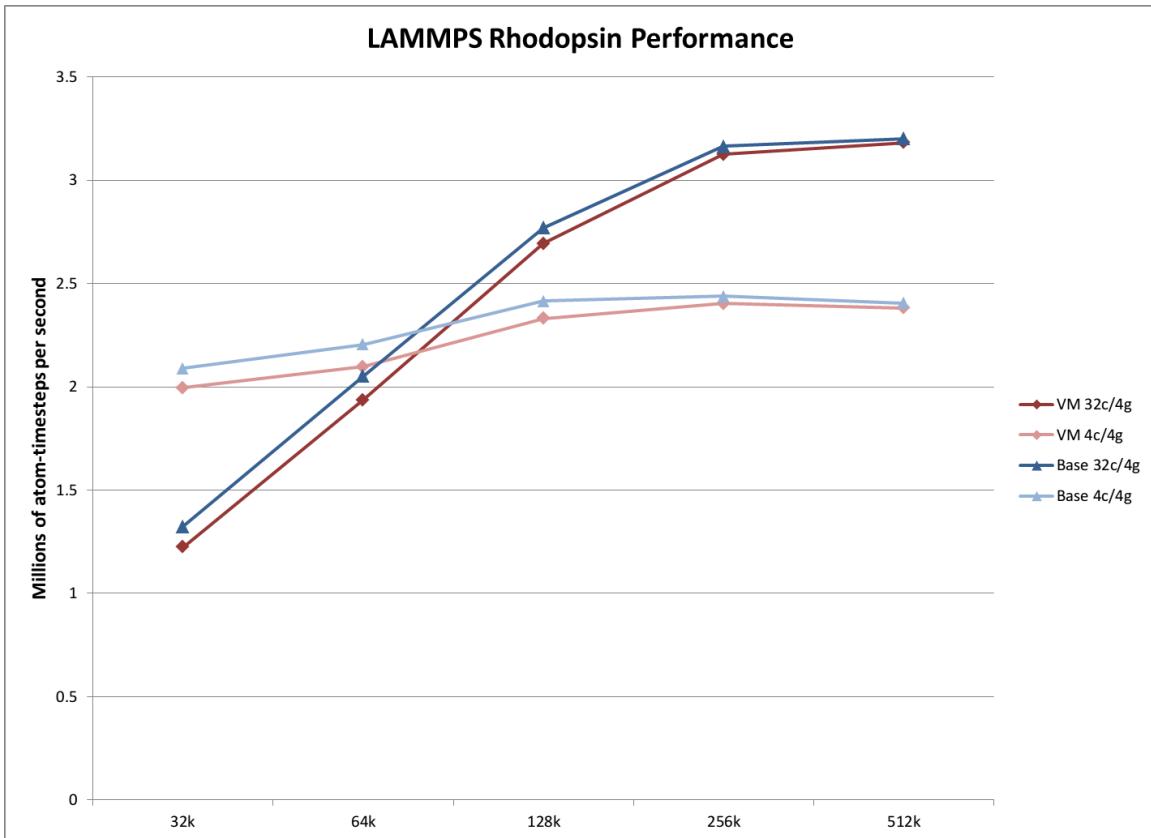


Figure 6.3 LAMMPS RHODO Performance

Again, the overhead of the virtualized configuration remains low across all configurations and problem sizes, with an average 96.4% efficiency compared to native. Interestingly enough, we also see the performance gap decrease as the problem size

2659 increases, with the 512k problem size in yielding 99.3% of native performance. This
2660 finding leads us to extrapolate that a virtualized MPI+CUDA implementation would
2661 scale to a larger computational resource with similar success.

2662 **6.7.2 HOOMD**

2663 In Figure 6.4 we show the performance of a Lennard-Jones liquid simulation with 256K
2664 particles running under HOOMD. HOOMD includes support for CUDA-aware MPI
2665 implementations via GPUDirect. The MVAPICH 2.0 GDR implementation enables
2666 a further optimization by supporting RDMA for GPUDirect. From Figure 6.4 we
2667 can see that HOOMD simulations, both with and without GPUDirect, perform very
2668 near-native. The GPUDirect results at 4 nodes achieve 98.5% of the base system's
2669 performance. The non-GPUDirect results achieve 98.4% efficiency at 4 nodes. These
2670 results indicate the virtualized HPC environment is able to support such complex
2671 workloads. While the effective testbed size is relatively small, it indicates that such
2672 workloads may scale equally well to hundreds or thousands of nodes.

2673 **6.8 Discussion**

2674 From the results, we see the potential for running HPC applications in a virtualized
2675 environment using GPUs and InfiniBand interconnect fabric. Across all LAMMPS
2676 runs with ranging core configurations, we found only a 1.9% overhead between the
2677 KVM virtualized environment and native. For HOOMD, we found a similar 1.5%
2678 overhead, both with and without GPU Direct. These results go against conventional
2679 wisdom that HPC workloads do not work in VMs. In fact ,we show two N-Body
2680 type simulations programmed in an MPI+CUDA implementation perform at roughly
2681 near-native performance in tuned KVM virtual machines.

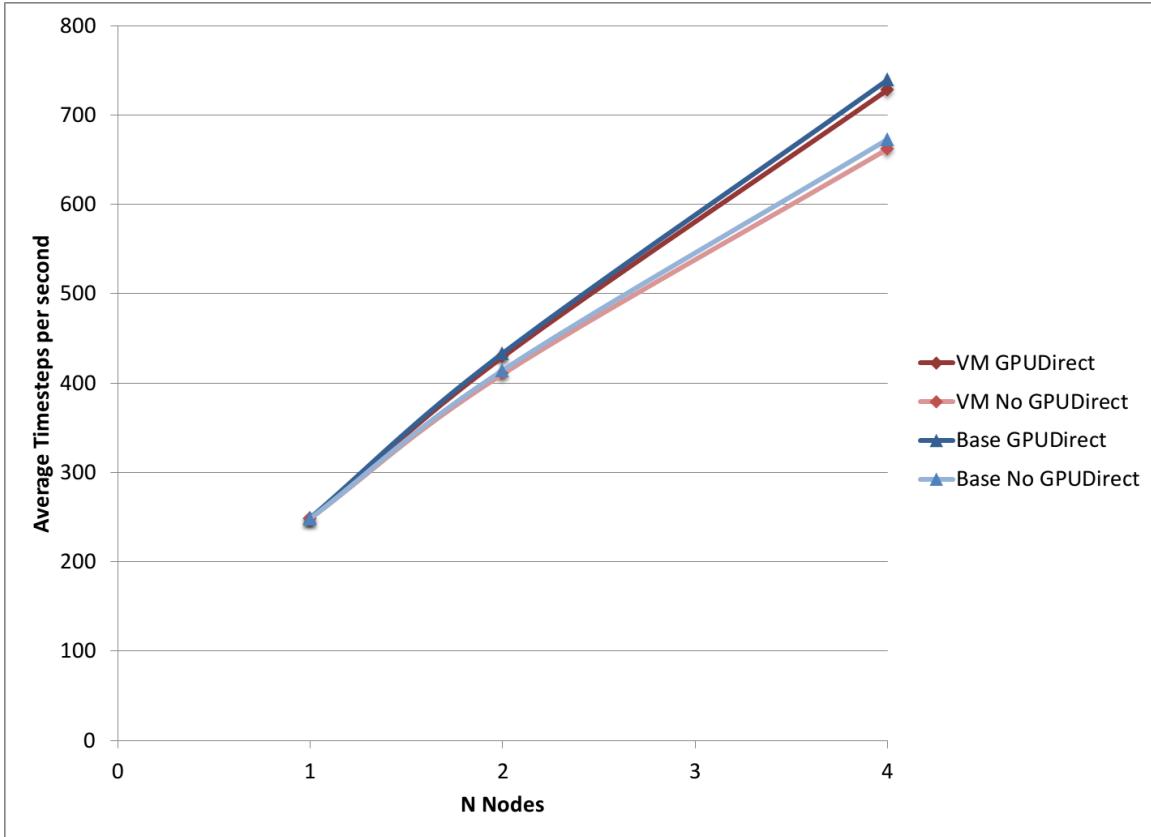


Figure 6.4 HOOMD LJ Performance with 256k Simulation

With HOOMD, we see how GPUDirect RDMA shows a clear advantage over the non-GPUDirect implementation, achieving a 9% performance boost in both the native and virtualized experiments. While GPUDirect's performance impact has been well evaluated previously [222], it is the author's belief that this manuscript represents the first time GPUDirect has been utilized in a virtualized environment.

Another interesting finding of running LAMMPS and HOOMD in a virtualized environment is as workload scales from a single node to 32 cores, the overhead does not increase. These results lend credence to the notion that this solution would also work for a much larger deployment. Specifically, it would be possible to expand such computational problems to a larger deployment in FutureGrid [229], Chameleon Cloud [230], or even the planned NSF Comet machine at SDSC, scheduled to provide

2693 up to 2 Petaflops of computational power. Effectively, these results help support
2694 the theory that a majority of HPC computations can be supported in virtualized
2695 environment with minimal overhead.

2696 **6.9 Chapter Summary**

2697 With the advent of cloud infrastructure, the ability to run large-scale parallel sci-
2698 entific applications has become possible but limited due to both performance and
2699 hardware availability concerns. In this work we show that advanced HPC-oriented
2700 hardware such as the latest Nvidia GPUs and InfiniBand fabric are now available
2701 within a virtualized infrastructure. Our results find MPI + CUDA applications such
2702 as molecular dynamics simulations run at near-native performance compared to tra-
2703 ditional non-virtualized HPC infrastructure, with just an averaged 1.9% and 1.5%
2704 overhead for LAMMPS and HOOMD, respectively. Moving forward, we show the
2705 utility of GPUDirect RDMA for the first time in a cloud environment with HOOMD.
2706 Effectively, we look to pave the way for large-scale virtualized cloud Infrastructure
2707 to support a wide array of advanced scientific computation commonly found running
2708 on many supercomputers today. Our efforts leverage these technologies and provide
2709 them in an open source Infrastructure-as-a-Service framework using OpenStack.

2710 **Chapter 7**

2711 **Virtualization advancements to**
2712 **support HPC applications**

2713 Throughout this dissertation, the question of whether cloud infrastructure, using vir-
2714 tualization, can support mid-tier scientific computation has been investigated. These
2715 scientific problems are tightly coupled distributed memory computations, and in the
2716 past have failed to perform well on traditional public and private cloud systems. It is
2717 our estimate that this is due to a number of reasons, including hypervisor design and
2718 implementation, lack of hardware advances, VM placement inefficiencies, and the lack
2719 of hardware availability in virtualization. In Chapter 3, we discussed the base case
2720 with off-the-shelf single node configurations. This showed that hypervisor selection
2721 matters a great deal in performance and reliability, and that some applications can
2722 perform well on a single node. This study expressly avoided multi-node configurations
2723 due to the previously documented issues with Ethernet interconnects in virtualized
2724 environments [47].

2725 With the rise of GPUs in HPC, Chapter 4 offers a first solution in Xen to GPU
2726 passthrough, as an alternative to either no GPU availability or providing GPUs access

2727 across a network by virtualizing the front end API libraries. It was found that our
2728 method of GPU passthrough incurred some overhead with Xen on older x86 hardware
2729 when transferring data across the PCI-Express bus, but provided the best option at
2730 the time. As methods were derived for other hypervisors such as KVM and VMWare,
2731 we looked to evaluate all options for a multitude of computational problems, including
2732 both CUDA and OpenCL codes in Chapter 5. This research found that again newer
2733 hardware without the QPI interconnect between the socket and the PCI-Express
2734 bus (Sandy-Bridge and up CPUs) can yield near-native performance for a range
2735 of applications. Specifically it was found that the KVM hypervisor was the most
2736 performant and stable hypervisor, and LXC, a container option, also performed very
2737 well (although with security limitations).

2738 Chapter 6 combines the lessons learned from KVM tuning, GPU passthrough, and
2739 intersects our other related efforts in providing a high performance interconnect, in
2740 this particular case InfiniBand [206], into a virtualized cluster environment. Specifi-
2741 cally, a test-bed was created across 4 nodes, each with Kepler GPUs and FDR Infini-
2742 Band passed through to VMs spanning the entire CPU set. InfiniBand was specifically
2743 set up to utilize SR-IOV, which allows for the multiplexing of the ConnectX-3 VPI
2744 card to the guest instances. From here, two Molecular Dynamics simulations were
2745 run both in this tuned KVM configuration and on bare-metal. The results indicate
2746 that overhead in virtualization is on the order of 1-2% for these HPC applications
2747 given a few different configurations and problem sizes.

2748 From this work, a number of observations start to emerge. First, that virtual-
2749 ization may indeed be able to support HPC workloads given the advances. While
2750 the scale of the experiments of the virtual cluster is small at only a few dozen cores,
2751 the performance trends from scaling up HOOMD-blue in Figure 6.4 look very closely
2752 correlated between the virtualized and bare metal experiments. While this applica-

2753 tion, as well as other HPC applications using similar distributed memory models, all
2754 need to be replicated at a much higher scale. We have shown there are currently no
2755 limitations in doing so, aside from the availability of the infrastructure itself. It is our
2756 hope that with this knowledge, future test-beds can be assembled at a larger scale to
2757 further move this effort forward.

2758 Another observation is that the KVM hypervisor continually evaluates to be the
2759 most performance oriented hypervisor studied in this dissertation. While this is a
2760 bit of surprise given it is a Type 2 hypervisor which introduces additional potential
2761 for host noise and overhead, it none-the-less showed the smallest degree of variation
2762 between results, both in Chapter 3 experiments and again in Chapter 5. KVM also
2763 offers the best performance of all hypervisors overall, and with many workloads such
2764 as seen in Figure 5.2, where KVM often performs within 0.5% of native in SOC bench-
2765 marks. Furthermore, KVM has support for CPU pinning, NUMA socket binding, PCI
2766 passthrough and SR-IOV, and even transparent huge pages and advanced migration
2767 mechanisms (described in more detail later in this Chapter). While it is possible
2768 other specialized hypervisors, such as Palacios [31] (not studied) could also perform
2769 similarly, the KVM hypervisor has a large community support, industry backing, and
2770 production-level integration into the latest private infrastructure such as OpenStack.

2771 A 3rd observation regarding high performance virtual clusters is that the SR-IOV
2772 InfiniBand integration described in Chapter 6 provides a drastic shift in the outlook
2773 for distributed memory applications. While SR-IOV InfiniBand does have a 15-30%
2774 overhead in latency compared to native implementations for small messages [206],
2775 this is still an order of magnitude better than current Ethernet options available from
2776 cloud providers such as Amazon. Furthermore, bandwidth of InfiniBand solutions
2777 in virtual clusters looks to be near-native, also surpassing current Ethernet deploy-
2778 ments. It is also possible for other interconnects, such as Intel's emerging Omni-Path

2779 interconnect, to demonstrate similar or better results in a virtualized ecosystem, and
2780 future experimentation should try to leverage other interconnection options if the
2781 hardware supports it.

2782 While these now smaller and better defined performance differences may not be
2783 satisfactory for extreme-scale distributed memory applications, we expect a large
2784 amount of users with mid-tier scientific computational problems to be accepting of
2785 these small overheads when considering the value added by working in a virtualized
2786 environment. Armed with this knowledge, we find that the outlook for high perfor-
2787 mance virtual clusters to be promising.

2788 However, next steps are needed to demonstrate the value of virtualization, pro-
2789 viding a more rich user experience while simultaneously further enhance performance
2790 for many users. These value-added techniques such as efficient tuning, scheduling,
2791 VM cloning, and compute-migration may in fact help enable new classes of scientific
2792 computations, not only within HPC applications but also with big data platform ser-
2793 vices. We specifically focus on leverage the KVM hypervisor in conjunction with a
2794 high speed, low latency interconnect to provide new features that otherwise have yet
2795 to be made possible. While much of this work is under construction, this nonetheless
2796 gives a glimpse at some future directions in high performance virtualization.

2797 In the rest of this chapter, we look to review the methods utilized in this disser-
2798 tation regarding virtualization, then identify research, design, and future implemen-
2799 tation of advanced virtualization techniques to enable a new class of infrastructure
2800 with added performance and features that have yet to be realized. We focus on guest
2801 memory optimizations, live migration deployments, and integration with an RDMA-
2802 enabled interconnect for novel VM migration and cloning. It may be possible for
2803 these advancements, once implemented, to have an impact not only on cloud infras-
2804 tructure, but also on dedicated environments, which support big data applications or

2805 other distributed memory applications that focus on both usability and performance.

2806 7.1 PCI Passthrough

2807 As cloud computing's reach into distributed systems increases, so does the require-
2808 ment of virtualization to perform at near-native speeds and take full advantage of
2809 the underlying hardware. While this does equate to fast and efficient hypervisors,
2810 it also alludes to the effective utilization of devices beyond CPU and memory. Such
2811 devices can vary widely and include hardware such as graphics processing units, net-
2812 working adapters, I/O hubs, web cameras, secondary and tertiary storage, to name a
2813 few. Within virtualization, often times it is necessary to emulate these devices, which
2814 provides a virtual hardware set that the guest VM can interact with using a spe-
2815 cialized driver, whereby the hypervisor translates requests to the underlying physical
2816 hardware.

2817 Emulated devices are often the most common method for device interaction with
2818 VMs, and can be a critical component in virtualization. Emulated drivers create
2819 a feature set in software, where all I/O requests are intercepted by the hypervisor
2820 and emulated on the real underlying hardware. Often times, the emulated hardware
2821 provided to the guest OS within a VM is older or more generalized than the given
2822 architecture set. This is due to the fact that the effort for constructing emulated
2823 devices is large, and the emulation of older hardware often helps with overall com-
2824 patibility. However, this emulation process can often be slow and lead to significant
2825 overhead when utilizing the underlying hardware, not to mention the lack of newer
2826 features provided by more recent hardware.

2827 Para-virtualized devices are specially tuned device software implementations of
2828 hardware, where para-virtualized device drivers are installed in a guest VM that

2829 operate on a particular I/O API. While this leads to improvements in performance
2830 over emulation methods, it requires the guest to be modified in order to communicate
2831 effectively with the hypervisor. With Xen, the entire guest OS can be para-virtualized,
2832 whereas in other solutions such as KVM using `virtio`, device drivers can be para-
2833 virtualized. The performance enhancement of para-virtualization is derived from the
2834 removal of the hardware compatibility that must be in place for emulated drivers,
2835 instead potentially using a tuned and customized API specific to the hardware at
2836 hand, at the cost of compatibility.

2837 A recent method for guest device interaction is through the use of direct I/O
2838 device passthrough, whereby the hypervisor (and controlling host OS) relinquish the
2839 entire control of a given device to the guest VM. This allows for the guest to have
2840 direct interaction with the physical hardware, removing the need for complicated
2841 para-virtualized methods or slow emulated hardware. As the Peripheral Component
2842 Interconnect (PCI) bus, and the updated PCI-Express bus is the most common hard-
2843 ware interface for devices on modern CPU architectures, this method often viewed as
2844 PCI passthrough.

2845 An I/O Memory Management Unit is responsible for managing the connection of
2846 direct memory access (DMA) capable hardware along an I/O bus to main memory.
2847 Just like a CPU MMU, the I/O MMU maps device-visible memory addresses to physi-
2848 cal memory addresses. However, utilizing DMA capable drivers within a guest directly
2849 without I/O MMU virtualization technology would result in the guest attempting to
2850 perform DMA operations on incorrect guest physical addresses that wouldn't map to
2851 proper machine addresses. In order to safely and securely enable PCI passthrough in
2852 virtualized settings, such CPU architecture mechanisms are needed.

2853 As the popularity of virtualization increased, CPU architectures have met this
2854 demand with I/O MMU virtualization extensions. Intel and AMD have introduced

2855 VT-d and AMD-Vi (also named IOMMU) processor support for such operations.
2856 These extensions provide the necessary mechanisms to isolate and restrict I/O
2857 devices specific to their owned partition space [231], through the use of I/O device
2858 assignment, DMA remapping, interrupt remapping, interrupt posting, and error re-
2859 porting. DMA remapping ensures device DMAs not only find correct virtual memory
2860 addresses but also act on only those memory addresses that are allowed, which pro-
2861 vides the necessary VM isolation. Thus, hardware DMA remapping enables direct
2862 device assignment to VMs without device-specific knowledge in the hypervisor.

2863 Using these IOMMU virtualization techniques for direct I/O passthrough with
2864 various hypervisors depends greatly on the hypervisor at hand. First, the BIOS must
2865 have IOMMU virtualization enabled and the OS kernel must initialize such hardware
2866 extensions at boot. This initialization can be done with the `intel_iommu=on` Linux
2867 kernel parameter for all Intel VT-d enabled architectures. With Xen, a specialized
2868 device driver, called `xen-pciback`, is used to "grab" the PCIE device on boot to
2869 keep the device state uninitialized and ready to be passed through to a VM. This
2870 is described in more detail in Section 4. With KVM, a similar method is utilized,
2871 whereby either `pci-stub` or `vfio-pci` (the former for pre-3.9 Linux kernels, the latter
2872 for current Linux systems) is utilized to bind to the PCI device on boot before the
2873 kernel or other add-on modules attempt to initiate the device. Either method is built
2874 into the kernel and any device drivers must be blacklisted so as to ensure proper
2875 ordering. `vfio-pci`, similar to `xen-pciback`, take the PCI device ID as a parameter
2876 to determine which device to bind to (this can be found from the output of `lspci`),
2877 and holds all device initiation until the device is handed to a booting guest.

2878 Generally, PCI passthrough can be utilized for most PCI devices, however GPUs
2879 devices can require some added configuration and considerations. This is largely due
2880 to the fact that GPUs are VGA devices, which represent a special case. Specifically,

2881 VGA devices to be used for PCI passthrough must not be the primary VGA displays.
2882 Second, the VGA BIOS has to be loaded by the guest VM before actual BIOS boot.
2883 This can be done using a specific emulated BIOS, which with KVM can be either
2884 a modified SeaBIOS or a EUFI-enabled Open Virtual Machine Firmware (OVMF)
2885 configuration. For Nvidia devices, only approved devices (often Tesla and QUADRO
2886 adapters) can be used for GPU passthrough due to proprietary VGA BIOS config-
2887 urations. Of further note, some GPU devices come with attached PCI Bridges due
2888 to packaging and compatibility reasons. This includes add-on GPU servers such as
2889 the Nvidia S2050, or dual-GPU units such as the Nvidia Tesla K80 GPU. Other de-
2890 vices may be packaged with such PCI Bridges for including onboard sound controllers
2891 too. However, these PCI Bridges handle PCIE connections to the devices and yet
2892 are often in separate IOMMU domains, which cause GPU passthrough to fail due to
2893 Access Controller Services (ACS) errors in the IOMMU hardware. While there are
2894 new kernel patches coming available to override the ACS mechanisms with KVM,
2895 this obviously provides a significant security vulnerability that may be problematic
2896 for deployments outside of the academic realm.

2897 In Chapters 5 and 6, KVM’s PCI passthrough is detailed for use with Nvidia
2898 GPUs, where we find that this method can, with NUMA placement of VMs and
2899 newer hardware on Sandy Bridge architectures, perform at near-native speeds. With-
2900 out GPU passthrough, GPU usage within a virtualized domain was either not pos-
2901 sible, or required the use of a front-end API solutions. As discussed in Section 4.3
2902 of this dissertation, such remote API methods are suboptimal due to performance
2903 considerations and the lack of full-feature support. Some of the best methods de-
2904 veloped thus far for utilizing remote GPUs in a virtualized architecture come from
2905 the rCUDA project [55], which sends CUDA commands across an interconnect to a
2906 remote device. However, this approach is fundamentally limited by the interconnect

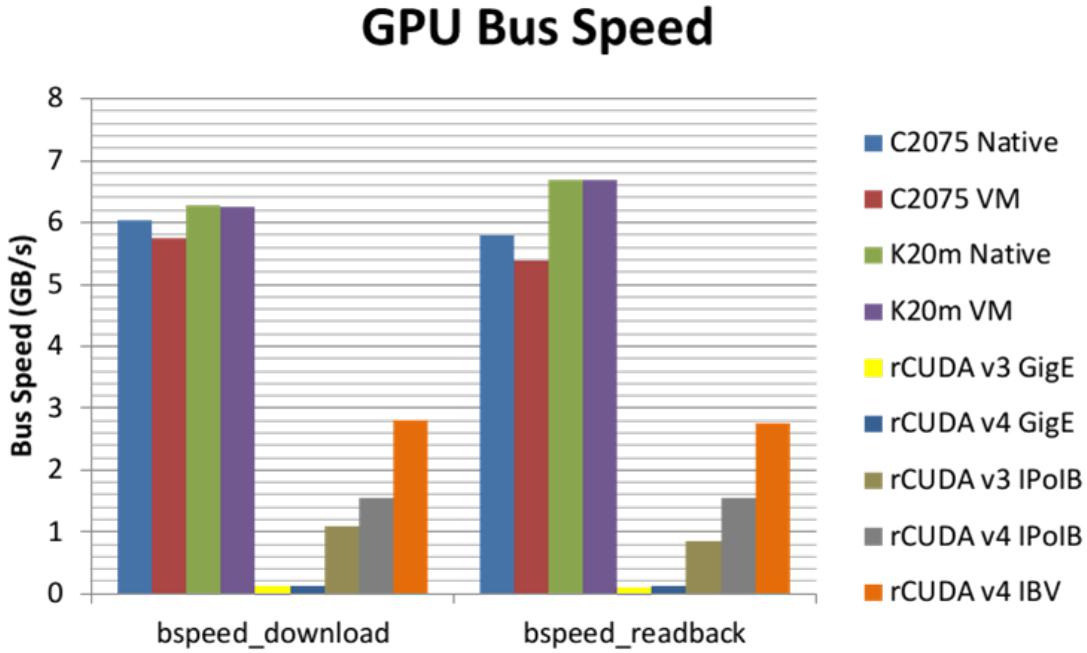


Figure 7.1 Comparison of GPU Passthrough in Xen to rCUDA across various interconnects

2907 itself. Using data from the rCUDA [232] project we compare GPU passthrough per-
 2908 formance of both C2075 and K20 cards in Xen to various interconnects reliant on
 2909 rCUDA. In Figure 7.1 it is illustrated that even in the best case with high speed
 2910 InfiniBand, rCUDA is still limited by the interconnect fabric, which can only operate
 2911 as fast as the same PCIE bus the GPU is attached to. In reality, even the top-end
 2912 IB adapters are not capable of saturating full 16x PCIE lanes. Furthermore, all GPU
 2913 data transfers saturate the interconnect, leaving no available bandwidth for communi-
 2914 cation operations as found with many distributed memory HPC applications. Sockets
 2915 and shared memory approaches found in API-remoting methods such as gVirtus [175]
 2916 suffer even worse performance impacts due to the necessity to buffer memory, as seen
 2917 in [233]. In summary, these front-end API methods are suboptimal in comparison to
 2918 GPU passthrough, as the transfer time between CPU and GPU memory often can

2919 have a drastic impact on overall application performance.

2920 While PCI passthrough works well for providing dedicated accelerator resources
 2921 such as GPUs, a sharing model of PCI passthrough does not hold. This is because
 2922 PCI passthrough is a simple 1-to-1 relationship between a guest and PCI device.
 2923 While it is possible to have a 1-to-many relationship (e.g. a single VM with multiple
 2924 GPUs connected), there is no way to share a single device across multiple guests or
 2925 the guest and the host. While this is not an issue and in fact a desirable effect with
 2926 GPUs (GPUs are not designed for multi-application sharing and such a solution would
 2927 largely lead to significant inefficiencies), high speed networking adapters attached on
 2928 a PCIE bus do have a necessity to be shared in virtualized environments.

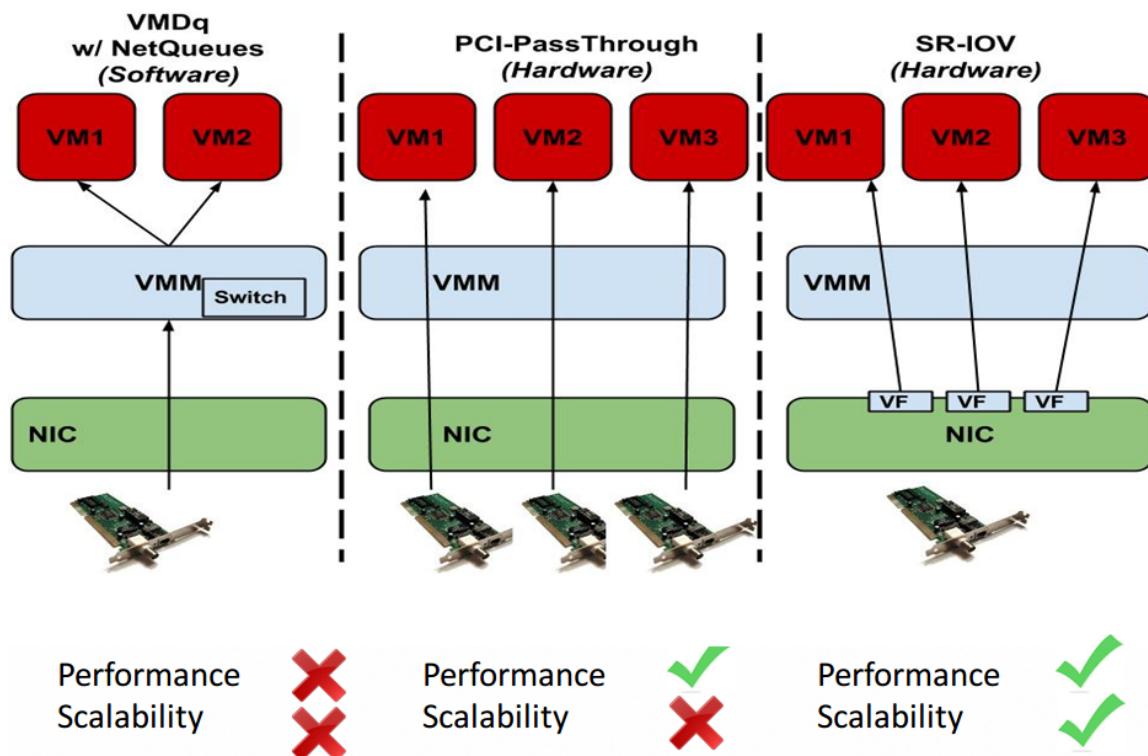


Figure 7.2 Efficient VM networking comparison [206]

2929 With this requirement for sharing PCI networking adapters, a few options exist
 2930 and are illustrated in Figure 7.2. The first option is to use software multiplexing that

2931 sorts inbound and outbound traffic and forwards packets to queues set up for each
2932 VM as necessary, as implemented with Virtual Machine Device Queues (VMDq) [234].
2933 While this solution of software queues has a workable Ethernet solution and can allows
2934 VMs to share a PCI based NIC, there are performance limitations. This is due to
2935 the large amount of hypervisor involvement necessary in sorting the queues, as well
2936 as the fact that all data is buffered between the hypervisor and guest memory. The
2937 next and simplest option is simply to add more PCI adapters to a given server. While
2938 multiple PCI passthrough does work, there is a fundamental limitation to this method
2939 as the number of cores and NUMA sockets is expanding faster than the number of
2940 PCI laines available per socket.

2941 With these limitations in mind, Single Root I/O Virtualization (SR-IOV) has
2942 been developed. This standardization effectively enables multiplexing of a PCI based
2943 communications adapter (typically Ethernet but also InfiniBand adapters) within
2944 the hardware. With SR-IOV, multiple Virtual Functions (VFs) are created and con-
2945 figured in hardware. Each VF has a dedicated resource pool with specific Tx and
2946 Rx queues, along with other lightweight PCI resources such as registers, BARs, and
2947 hardware descriptors. Adapters also maintain a Physical Function (PF), which is
2948 a fully-implemented PCI device that acts not only as a standard controller card,
2949 but also as a control mechanism for the VFs (given firmware adjustments). These
2950 mechanisms are illustrated in Figure 7.3 provided by Intel [235].

2951 To use SR-IOV within a virtualized setting, a VF is given to a guest VM on
2952 boot. This happens using the PCI passthrough mechanisms, except the specific VF
2953 PCI identifier is used, instead of the actual adapter card. Within the guest, the VM
2954 loads a specific driver (usually supplied by the hardware vendor) that is able to probe
2955 and detect VF functionality. This driver fills in the descriptors and sets up memory
2956 allocations for DMA directly within the guest OS, effectively creating dedicated data

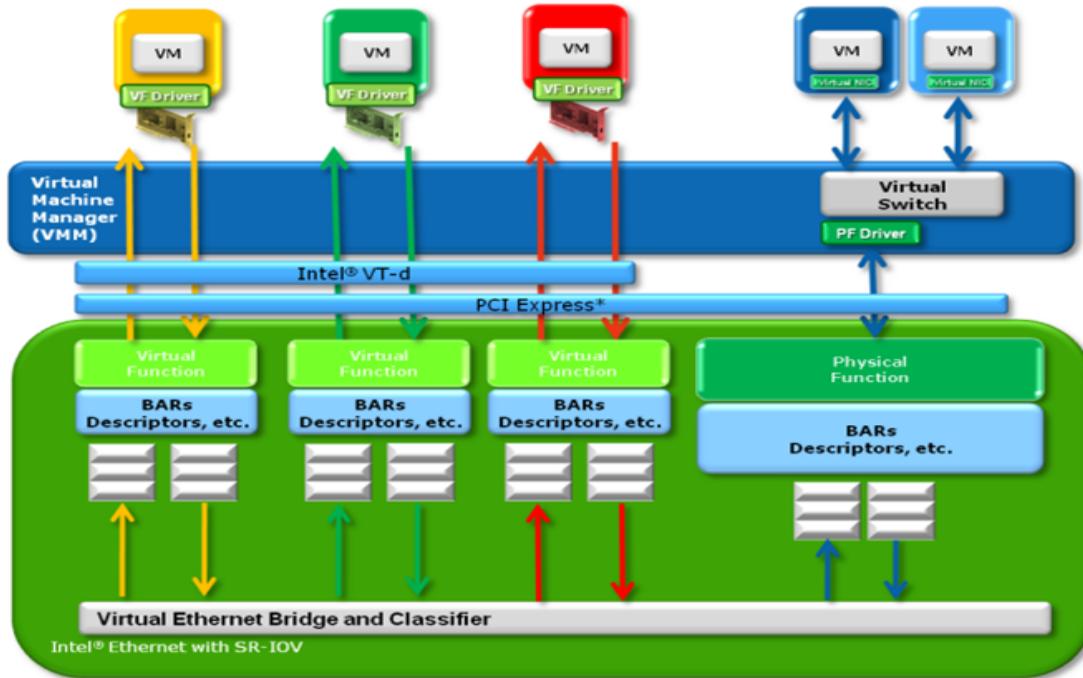


Figure 7.3 SR-IOV Architecture with Virtual Functions, PCI SIG [235]

2957 queues within the VM.

2958 When a datum (could be a packet or a message, depending on the adapter type)
 2959 arrives at the physical card, it is sent through a hardware switch which places the
 2960 data into a pool specific to the target VF. The data is then immediately DMAed to
 2961 the guest based on the preallocated memory buffers. This is possible through the use
 2962 of VT-d mechanisms which enable the translation between device addresses and the
 2963 guest virtual addresses, bypassing the need for hypervisor memory translation. This
 2964 entire process happens without any CPU interaction, other than occasional interrupts
 2965 to provide notification of completed data work queues. This essentially enabled direct
 2966 hardware DMA to a guest without any hypervisor involvement.

2967 Using modern 10GbE and InfiniBand adapters, SR-IOV can easily configurable to
 2968 have up to 64 VFs, enabling a great deal of scalability within a given host. Without
 2969 any hypervisor interaction or context switching, bandwidth and latency can start to

2970 approach near-native levels, which was previously not possible with other methods.
2971 Looking at InfiniBand, research has found that Mellanox adapters with SR-IOV in
2972 KVM can achieve near-native bandwidth, while incurring only a slight 15-30% over-
2973 head on small (1 byte) messages [205, 206]. This latency overhead is due to the extra
2974 hardware switching that has to happen between the VFs in the adapter itself, as well
2975 as the IOMMU involvement.

2976 *TODO: Clean up citations*

2977 7.2 Memory Page Table Optimizations

2978 As we have seen both in Chapter 3, as well as in other supported literature [47],
2979 virtualization of memory structures is a point of contention and potential overhead.
2980 This is often due to the extensive effort a hypervisor has to perform in order to
2981 translate memory addresses from guest-virtual addresses to host-virtual addresses,
2982 and then again to machine-physical addresses. While this is a necessary function of
2983 virtualization and a hypervisor, there are various methods developed both in hardware
2984 and software to provide such address translation functionality, each with their own
2985 advantages and disadvantages.

2986 Modern computing systems provide a mapping of virtual memory address space to
2987 physical machine memory. This memory management technique allows processes to
2988 have independent virtual address spaces, which provides security and process isolation.
2989 Today's x86 CPUs, as well as many other CPU architectures, have a hardware memory
2990 management unit (MMU) that provides translation abilities within hardware, that
2991 often have page table entries (PTE) for storing virtual to machine memory mappings,
2992 and a translation lookaside buffer (TLB) that provides an effective cache for the most
2993 commonly used virtual addresses. Specifically, x86 page tables are walked iteratively

2994 in hardware with their layout specified by the x86 hardware specification, where the
2995 CR3 register holds the page table base, and a 4-level radix tree structure represents
2996 the page table hierarchy for 4KB pages.

2997 With virtual machines, a 2-level address translation is needed, where guest virtual
2998 memory is translated to guest physical memory, and then again to physical machine
2999 memory. This direct two-level memory mapping is classically handled by the hyper-
3000 visor as the guest cannot access machine memory directly. The hypervisor functions
3001 in software without hardware support, so it can be expensive for it to update and
3002 maintain guest memory translation. With x86 virtualization, memory virtualization
3003 is handled using shadow page tables [236]. Shadow page tables eliminate the need
3004 for emulation of physical memory inside the VM by creating a page table mapping
3005 from guest virtual to machine memory. However, these page tables are not walkable
3006 by hardware like a TLB, and as such, guest OS page tables require updating of the
3007 shadow page table by the hypervisor. This can be costly not only in the additional
3008 management, but also by the cost of VMexit and VMentry calls which are known
3009 to add thousands CPU cycles of overhead for each call. If there is a memory bound
3010 application, continual shadow page table management by the hypervisor can add a
3011 notable overhead, impacting overall application performance.

3012 Recently, Intel and AMD have implemented Extended Page Tables (EPT) and
3013 nested paging (NPT), respectively, to cope with the issues of shadow page tables.
3014 With nested paging, a guest page table converts guest virtual addresses to get phys-
3015 ical addresses, and another second level table converts guest physical addresses to
3016 machine addresses. Each address translation in guest mode requires a 2D page walk
3017 where the guest page table is traversed and each guest physical address requires a
3018 second level page table walk to obtain the machine address. This support means the
3019 TLB hardware is able to keep track of both guest pages and hypervisor pages concur-

3020 recently, effectively removing the need for shadow page tables and resulting hypervisor
3021 intervention entirely. Nested paging provides a simple design with no necessary hy-
3022 pervisor traps leading to less overall overhead and swapping, less TLB flushes, and a
3023 reduced memory footprint.

3024 The downside of nested paging for virtual machines occurs when there is a TLB
3025 miss. A TLB miss is when a requested page is not in the TLB, and in such cases
3026 the cost of a TLB miss is substantially higher in a guest VM. Natively, a TLB miss
3027 requires a walk of 4 address entries for 4KB pages. However, in a guest, each of those
3028 4 address entries require another second level walk to find the system physical address
3029 for each guest page entry, plus a final nested page walk to translate the final guest
3030 physical address to a usable machine address.

3031 The TLB miss can be illustrated in the efforts in Figure 7.4 from Bhargava et.
3032 al [237], whereby we can see the steps for handling a TLB miss on an AMD CPU in
3033 both native (left) and virtualized (right) modes. With 2D nested or extended page
3034 tabling given in Figure 7.4, each gLn entry cannot be directly read using a guest
3035 physical address, and as such the nested page table walk is necessary to translate the
3036 guest physical address before the entry can be read. This has to happen recursively
3037 for each level and in this example with 4KB pages, that includes 4 levels. Finally, a
3038 final nested page walk is required to translate the guest physical address o the data
3039 to a physical machine address.

3040 The cost of a 2D page table can be evaluated in the number of references, and
3041 is easily calculated. If a guest page walk has n levels and a nested page walk has m
3042 levels, the virtualized 2D walk has a cost calculated by:

3043

3044 $nm + n + m$

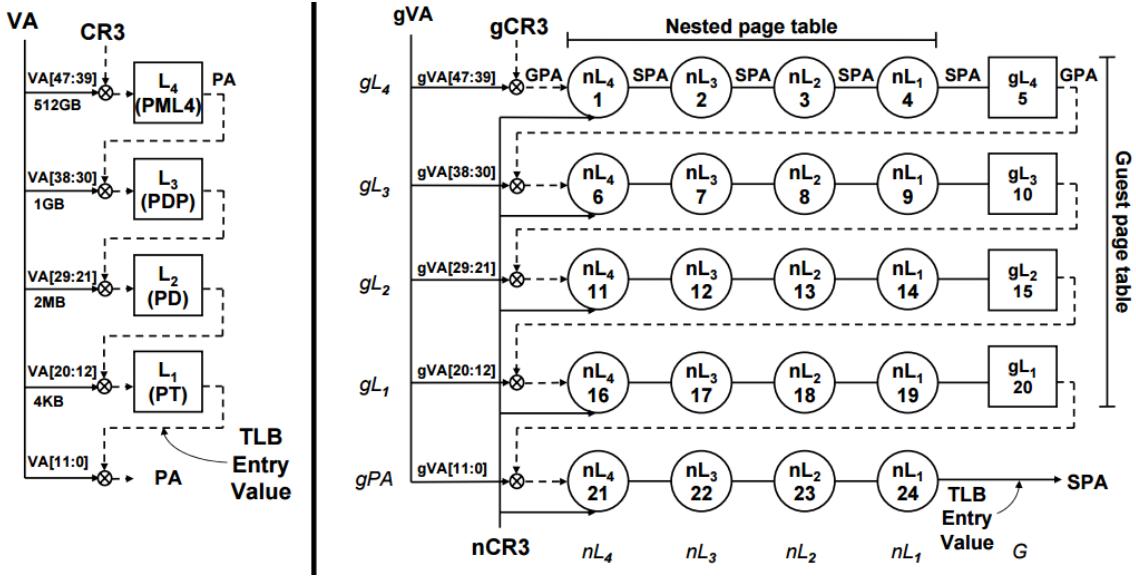


Figure 7.4 Native TLB miss walk compared with 2D virtualized TLB miss walk from [237]. On the left illustrates a native page table walk. On the right illustrates the lengthy 2D nested page table walk for a VM.

3045 For 4KB pages, this requires 24 references in a virtual TLB miss (4 page walks
 3046 and 4 nested page walks), compared to a cost of just 4 references for a single TLB
 3047 walk natively, as indicated in Figure 7.4. While many applications illustrate this TLB
 3048 miss cost is much less than that of managing shadow page tables, it still can lead to
 3049 a significant gap in performance between non-virtualized applications, especially as
 3050 VM count or an application's memory footprint increases.

3051 One potential way to decrease the chance of a TLB miss (and therefore the cost of
 3052 a miss) is by using a larger page size. By default, x86 hardware uses 4KB pages sizes,
 3053 but newer hardware can support 2M and 1G page sizes as well, effectively named
 3054 *transparent huge pages* or THP. Using the KVM hypervisor with transparent huge
 3055 pages enabled, we can create guest VMs backed entirely by 2M huge pages [199]. We
 3056 can also enable transparent huge page support within the guest as well, to have the
 3057 entire guest OS (including kernel and modules) backed by 2M pages.

3058 The result of THP-enabled guest VMs can be significant. With huge pages on

3059 Intel x86 CPUs with EPT, there exists an entirely separate TLB for huge pages. This
3060 will naturally alleviate TLB pressure and therefore reduce TLB contention between
3061 guest and host operating systems (whereby the host is still utilizing 4KB paging for
3062 kernel space). Most importantly, the TLB reach is increased, because 2MB pages
3063 cover a larger addressable memory space. The size of the page tables themselves also
3064 decreases with the use of huge pages.

3065 If huge pages are used in the host, there are then 3 levels of a page table walk,
3066 and if huge pages are used in the guest, there is also only a nested page walk of
3067 3 levels as well. Using the formula, if THP is enabled in the host, there is a TLB
3068 miss cost of 19 references. If huge page support is enabled in both the guest and
3069 the host, this drops to only 15 references. While this is still more than the native
3070 miss cost of 4 or 3 references (for 4KB and 2MB pages, respectively), huge pages
3071 provide a 37.5% reduction in the TLB miss cost compared to 4KB pages, and still
3072 significantly better than the VMexit/entry costs associated with shadow page tables.
3073 While it would be possible to only enable huge pages within the guest, this would
3074 represent a suboptimal configuration. Here, 2MB pages will be splintered into 4KB
3075 pages, and thus negating most performance benefits [238]. Most importantly, the
3076 overall reduction of the number of TLB misses due to the increased TLB reach with
3077 huge pages can have a major impact on application performance.

3078 To evaluate the effect of 2M transparent huge pages on guest performance, we
3079 leverage the same KVM setup in Chapter 5 on the Bespin hardware. Specifically,
3080 THP is enabled both in the host as well as the guest OS kernels, and the same
3081 LibSVM application using GPUs is re-run. The libSVM GPU application can have
3082 significant memory requirements, as large chunks of the SVM datasets are stored in
3083 CPU memory and transferred in a sudo-random order to and from GPU memory,
3084 making it an ideal application to use for evaluating THP.

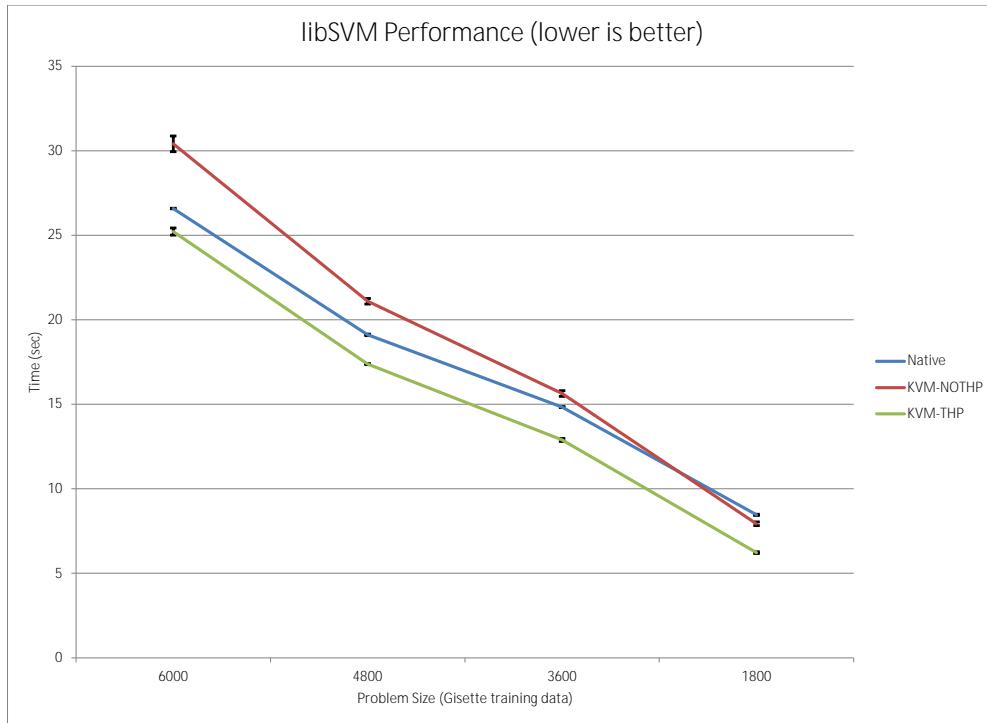


Figure 7.5 Transparent Huge Pages with KVM

The results of running libSVM in a THP-enabled VM, a VM with no THP, and natively without virtualization are all outlined in Figure 7.5. Comparing first just the KVM results without THP to the native solution, we can see the impact of THP on the overall application runtime, especially at larger problem sizes (6000 training sets). However, when TLB is enabled in the guest and host, we actually see the KVM VM solution *outperform* the native solution. This is because guest privileged OS memory used to buffer to/from GPU memory is backed by 2MB pages in kernel space, instead of the normal 4k pages, as in the native solution (which has to use 4KB pages). This means data transfers can take advantage of a larger TLB reach, resulting in improved performance. While this is likely a special case for THP usage

3095 with the libSVM application, the fact that a VM can even occasionally outperform a
3096 native runtime is a noteworthy accomplishment. This also underscores the need for
3097 careful tuning and best-practices for hypervisors when supporting advanced scientific
3098 workloads, with huge page support in both the guest and host environments as a key
3099 aspect, especially as the use of big data scientific applications increases.

3100 **7.3 Live Migration Mechanisms**

3101 Migration of VMs represents one of the fundamental advantages to virtualization, and
3102 also one of the greatest challenges to efficiency. With VM migration, the complete
3103 VM state is copied from a source to an unallocated destination host, where disk,
3104 memory, and network connections are kept intact. For disk continuity, a distributed
3105 and/or shared filesystem is utilized, most commonly but not exclusively NFS, where
3106 both the source and destination hosts have access to the VM disk. Network continuity
3107 is preserved as long as the destination guest is within the same LAN and generates an
3108 unsolicited ARP reply to maintain the original IP after migration. VM vCPU states
3109 and machine states are recorded from the source and quickly sent to the destination
3110 host when the VM is paused. For live migration, the source VM is paused only
3111 after all state and memory contents are copied to the destination. The last of the
3112 dirtied memory pages are copied over, and the newly formed destination VM is then
3113 un-paused. This pause and transfer time represents the entirety of a VM downtime
3114 during live migration, and is often at or under 100 milliseconds across commodity
3115 Ethernet networks (given a VM with low memory utilization).

3116 The memory transfer stages are often the main performance consideration for
3117 overhead during live migration. This is not only due to the potentially large amount
3118 of memory to be sent across the network, but the veracity at which the memory is

3119 changed. This is defined directly by the amount of main memory allocated (or in use)
3120 by the source VM. However, as a VM's memory is sent, the VM is still running and
3121 therefor memory pages can be dirtied, creating the need for any written page to be
3122 retransmitted. Given a small network and memory bound processes running within
3123 a VM, this can be an infinitely long process of page dirtying. Many live migration
3124 strategies provide an iterative timeout mechanism to avoid this infinite state, but this
3125 will lead to increased downtime during migration.

3126 The copying of memory pages for live migration can take multiple implementa-
3127 tions. Three common options are summarized below:

3128 • **Pre-copy Migration -** All memory pages are transmitted to the destination
3129 before the VM is paused. The hypervisor will note and track all dirtied memory
3130 pages, and retransmit those pages in iterative rounds. The rounds end when
3131 either no dirtied pages exist or a max iteration count has been reached. The
3132 VM state is then transmitted and resumed on the destination. This method
3133 was the first live migration technique used in Clark et al [239] and is by far the
3134 most common.

3135 • **Post-copy Migration -** The VM state is paused and sent to the destination
3136 hypervisor, and immediately resumed. If the new destination VM generates
3137 a page fault, the VM is paused, and faulted pages are transmitted across the
3138 network on demand from the source and the VM resumed. This methodology
3139 is proposed for use in the Xen hypervisor by Hines et al [240].

3140 • **Hybrid-copy Migration -** Provides a compromise solution to memory paging.
3141 First, a single copy of the VM memory pages, or a subset of known-necessary
3142 memory pages, are sent to the destination. Then the source VM is paused,
3143 its VM state sent to the destination, and resumed on the destination. Known

3144 dirtied source pages, or missing pages, are then copied to the destination upon
3145 a triggered page fault utilizing the same mechanism as post-copy migration. An
3146 example of hybrid migration can be found via Lu et al [241].

3147 While pre-copy migration is the traditional and most used live migration tech-
3148 nique, there will soon be opportunities to implement other migration techniques to
3149 advance the mobility of distributed computing in high performance virtual clusters
3150 with virtualization.

3151 7.3.1 RDMA-enabled VM Migration

3152 Currently, most live migration in production environments occurs over TCP/IP con-
3153 nections due to the prevalence of commodity Ethernet connections within cloud in-
3154 frastructure. However, even if RDMA-capable interconnects are available in such
3155 infrastructure as described with InfiniBand in Chapter 6, live migration still usu-
3156 ally occurs over TCP/IP. For the case of InfiniBand, this is via IP over InfiniBand
3157 (IPoIB) [242], which can be an inefficient use of the interconnect bandwidth and add
3158 extra latency [243].

3159 The time it takes to migrate the memory contents from a source to destination
3160 VM can be significantly decreased by using an RDMA based mechanisms. Huang et
3161 al first provided a proposed pre-copy method for RDMA-based migration using the
3162 Xen hypervisor [244]. Specifically, they found an 80% decrease in migration time with
3163 RDMAwrite operations. This speedup is largely due to the removal of overhead nec-
3164 essary for processing TCP/IP communications, largely in CPU utilization for copying
3165 buffers, packet processing, and the included context switch overhead when competing
3166 for resources in a CPU-bound application (which are common in HPC environments).

3167 The live migration algorithm proposed in [244] uses the standard pre-copy mecha-

3168 nism that sends the entire memory contents across the network as RDMA operations,
3169 then iteratively copies dirtied pages before switching the running states. As discussed
3170 in the previous section, a post-copy migration strategy may have benefits for quick
3171 VM migration in high performance virtual clusters, or even for VM cloning as de-
3172 scribed later in Section 7.4. Furthermore, efforts in Chapter 3 have found that the
3173 Xen hypervisor is not best suited for HPC workloads. As such, there is a need to
3174 redefine the use of RDMA for VM migration using a hybrid post-copy mechanism
3175 in a high performance hypervisor. This post-copy migration mechanism is provided
3176 defined:

- 3177 ● Transfer initial CPU state, registers
- 3178 ● Start the page table pages translation process: MFN to PFN and use copy-base
3179 approach
- 3180 ● Concurrently, allocate remote memory on destination VM.
- 3181 ● Set up other machine state settings in destination
- 3182 ● Start destination VM, pause source VM.
- 3183 ● Initiate RDMAwrite of entire memory contents from source to destination.
- 3184 ● As page faults occur in destination VM, catch faults and perform RDMAread
3185 requesting pages.

3186 Currently efforts are underway to provide post-copy live migration in KVM/QEMU,
3187 using the `migrate_set_capability x-postcopy-ram` on mode within KVM [245].
3188 This method uses the Linux *userfaultfd* kernel mechanisms from a kernel 4.3 or newer.
3189 At the start, all memory blocks are registered as *userfaultfd*, so all faults cause the
3190 running thread to pause. In kernel space, the missing page is requested from the

3191 sender, which is prioritized over other pages being sent and is returned and mapped
3192 to the destination guest memory space and the thread or process is un-paused. This
3193 mechanism operates asynchronously, so that multiple outstanding page faults will not
3194 stop other executables within the VM. The `xpostcopy-ram` extension has been iden-
3195 tified as an opportune place to implement an RDMA based implementation within
3196 KVM.

3197 To provide RDMA functionality, the InfiniBand interconnect could first be used
3198 as a proof-of-concept. This choice is due to InfiniBand's rise in popularity, increased
3199 prevalence in virtualized systems with SR-IOV, as noted in Chapter 6, and RMDA
3200 functionality. However, other interconnect options exist that may be better suited for
3201 enhanced functionality and performance, such as Intel's new Omnipath [246], or an
3202 Ethernet solution such as RoCE [247], to name a few. While RDMA can be managed
3203 through the kernel level, it is best used in user-level APIs, such as MPI, or in the
3204 case of InfiniBand, ibVerbs. However, it may be ideal to select a interconnect-agnostic
3205 middleware for RDMA implementations to add future support for other interconnects,
3206 such as Photon [248].

3207 RDMA semantics can be used to either read or write contents of remote memory,
3208 in this case VM guest memory pages. However before such operations can take
3209 place, the target side of the operation must register the remote memory buffers and
3210 send the remote key to the initiator, effectively providing the DMA addressing to
3211 be used. Beyond the increased bandwidth and decreased latency benefits provided
3212 by an advanced interconnect with InfiniBand, RDMA also allows VM memory to be
3213 sent without involving the OS. This is due to InfiniBand's zero-copy, kernel bypass
3214 mechanisms, and asynchronous operations. This keeps the CPU load down, allowing
3215 for the CPU to spend time on the computation at hand rather than the I/O transfer,
3216 as it often has to when utilizing a TCP/IP stack.

3217 Selecting the proper RDMA based communication operations to use can make a
3218 notable difference in the overall performance of the migration. Some RDMA oper-
3219 ations are largely a one-sided involvement, which can have performance impact and
3220 should be carefully considered for use in post-copy live migration. The RDMAread
3221 operation requires more effort at the destination host, while RDMAwrite operation
3222 puts burden on the source host. One way to determine which method is best is by
3223 evaluating both source and sink CPU loads. However, this method likely will not
3224 be as obvious when we consider VMs will be not be running in an over-subscribed
3225 virtualized environment, but rather a highly optimized one.

3226 When the destination VM page faults, it is necessary to retrieve the page with as
3227 little latency as possible, as the running thread is paused. This is one of the advantages
3228 of using RDMA, but implementation details can also effect performance. Using the
3229 method developed in [245], the *userfaultfd* will trigger an RDMAread to retrieve the
3230 missing page. RDMAread requires no interaction from the source VM, as RDMAread
3231 is one sided and the memory buffers have been passed in the setup phase. To further
3232 enable quick return time for the missing page, enabling polling on the source host
3233 may further reduce latency, however verification of this will be necessary. When the
3234 RDMAread operation is finished, the running thread will resume and execution will
3235 continue.

3236 7.3.2 Moving the Compute to the Data

3237 While pre-copy migration is dominant in the live migration techniques of nearly every
3238 mainstream hypervisor today, the proposed post-copy migration could provide some
3239 key new advances for HPC and big data applications. One particular use case would
3240 be to send a lightweight VM to act directly on a large or set of large datasets and
3241 return a slimmed down result set. This would reduce the requirement of transmitting

3242 the data across a network entirely, and could potentially speed up data access latency
3243 drastically, as the returning information is transmitted in the form of memory pages
3244 and VM state. Essentially, the proposal is to send the compute to the data, instead
3245 of visa versa.

3246 With post-copy migration, one could move the computation at hand close to a
3247 data source in significantly less time than full pre-copy live migration. This data
3248 source, and lightweight VM sink, could potentially be something similar to a Burst
3249 Buffer system [249,250] or a classic HPC I/O node with a distribute filesystem such as
3250 Lustre or GPFS [251]. This data source could even potentially be a remote scientific
3251 instrument completely separate from the HPC infrastructure itself, especially if the
3252 network at hand is capable of RoCE [247] or iWARP [252]. A VM would initiate
3253 post-copy live migration, transmitting only the necessary CPU state, registers, and
3254 non-paged memory, rather than the full VM memory state. Once migrated, the
3255 VM could connect to a (now local) I/O or storage device, accessing data fast and
3256 performing the necessary calculations. The VM could potentially even forgo the copy
3257 of the majority of its memory. During this time, only the necessary memory pages
3258 required to complete the immediate calculation would generate a fault and trigger
3259 their transmission from the source. The VM could even return the result (rather
3260 than a very large dataset) to the original source VM.

3261 This post-copy live migration technique for remote data computation avoids the
3262 cost of spawning a whole new job and/or process with associated running parameters,
3263 as well as the extremely high cost of a full VM live migration using the pre-copy
3264 method. However, one potential downside of post-copy live migration would be the
3265 non-deterministic runtime, as it would be unknown how much remote memory paging
3266 would be required. This could lead to more time spent with the destination VM in
3267 a paused state awaiting remote memory pages, rather than if the entire VM memory

3268 contents were transmitted completely. Careful analysis of memory usage, or a hybrid
3269 copy method based on predetermined memory sections, could help overcome this
3270 issue, but may require a more advanced migration architecture.

3271 7.4 Fast VM Cloning

3272 In many distributed system environments, concurrency is achieved through the use
3273 of homogeneous compute nodes that handle the bulk of the computational load in
3274 parallel. This can take many forms, including master/slave configurations, or even
3275 a traditional HPC cluster with identical compute nodes, as are often used to sup-
3276 port Single Process Multiple Data (SPMD) computational models [253]. With high
3277 performance virtual clusters, there is a need to efficiently deploy and manage near
3278 identical virtual machines for distributed computation.

3279 In Snowflock [254,255], the notion of VM cloning is given. Specifically, Lagar et al.
3280 define the notion of VM Fork, where VMs are treated similarly to a fork system call for
3281 processes. This process is conceptually similar to VM migration, with the exception
3282 being that the source VM is not destroyed after the migration. Starting with a master
3283 VM, an impromptu cluster can be created across a network using the Xen hypervisor.
3284 Snowflock specifically uses Multicast to linearly scale out VM creation to many hosts,
3285 only coping a minimal state and then remotely coping memory pages when requested.
3286 This fetched memory on-demand is similar in principal to the post-copy live migration
3287 technique described in the previous section. This is further augmented with blocktap-
3288 based virtual disks with Copy-on-Write (CoW) functionally, delivering CoW slices for
3289 each child VM.

3290 While Snowflock provides an excellent framework for VM cloning, it is not suit-
3291 able for the current implementation. First, it uses Xen, which in previous research

3292 described in Chapter 3 has been found to have limited performance for HPC work-
3293 loads [171]. Second, SnowFlock is designed for Ethernet and IP based networks,
3294 which have significantly higher latency and lower bandwidth when compared to In-
3295 finiBand solutions. Developing analogous mechanisms, like what is listed below, with
3296 a high performance hypervisor such as KVM or Palacios [31] to use RDMA for VM
3297 memory paging could have an effect on the way in which virtual cluster environments
3298 are deployed.

- 3299 ● Prepare parent VM state, including registers and info.
- 3300 ● Prepare CoW disk images.
- 3301 ● Create large buffer for all VM memory on child hosts.
- 3302 ● Send vmstate via RDMAwrite or IB Send to N child nodes, where N is the clone
3303 size.
- 3304 ● Resume/start child VMs in tandem.
- 3305 ● Parent VM set up RDMA multicast (unreliable connection) and initiate sending
3306 of memory pages to all child VMs
- 3307 ● Child clone VM joins RDMA multicast.
- 3308 ● If child page faults, perform RDMAread operation for specific page.

3309 This method allows for efficient cloning of VMs based on a running parent VM.
3310 First, the VM state and images are copied to all child nodes to receive the VM,
3311 and blank memory is allocated. Then, each child VM is started, and page faults
3312 are handled via RDMA read. This will result in an initial slowdown, but as pages
3313 are received the child VMs will start to run. The rest of the memory is eventually

3314 sent via multicast to all VMs simultaneously. As multicast is unreliable, a delivery
3315 failure will just trigger a page fault and subsequent page retransmission via RDMA.
3316 It allows for direct page fault handling, while still allowing child VMs to start and
3317 run immediately. As RDMA multicast mechanisms are relatively questionable, more
3318 investigation is needed to evaluate the feasibility of this situation.

3319 It is expected that post-copy VM cloning wil work most efficiently if used in
3320 conjunction with guest VMs backed with huge pages. Transferring memory in 2M
3321 chunks will more effectively utilize network bandwidth by eliminating send/receive
3322 overhead. This also will hide the latency found in SR-IOV enabled interconnects
3323 for small messages. Furthermore, it will reduce the overhead of page fault handling
3324 mechanisms, as less overall pages will fault and be transferred. While huge pages are
3325 expected to improve VM cloning efficiency, empirical testing will still be necessary to
3326 properly evaluate their viability.

3327 While a VM fork mechanism leveraging post-copy live migration in KVM will
3328 quickly spool up cloned VMs, the eventual memory transfer will eventually fail to
3329 scale past the network's capacity. This could happen if hundreds or thousands of child
3330 clone VMs are started simultaneously, as is likely in large scale deployments. As such,
3331 a hierarchical distribution may be necessary. One possible method for this would be
3332 a two-stage cloning mechanism, where child VMs are cloned one to each cabinet, and
3333 the entire memory contents copied using the pre-copy migration mechanism. From
3334 there, further cloning occurs to deploy many cloned VMs to individual nodes within
3335 the cabinet. Organization of mid-tier cloned VMs would likely be determined based
3336 on RDMA fabric configurations within cabinets, as this is a network-bound process.

3337 7.5 Virtual Cluster Scheduling

3338 Historically, cloud infrastructure has taken a simplistic approach when it comes to VM
3339 and workload scheduling. Often, round-robin or greedy [256] scheduling algorithms
3340 are naively applied. With round robin scheduling, a simple list of host machines are
3341 used and iterated over as VM requests are made. This essentially scatters the VMs
3342 without regard to their locality, and over-subscription becomes commonplace regard-
3343 less of the number of requested VMs or their interconnection. A greedy algorithm can
3344 help keep spacial locality, but focuses specifically on over-subscription as well to help
3345 consolidate VM allocations. While this over-subscription aspect is advantageous for
3346 public cloud providers such as Amazon EC2, it becomes counterproductive for high
3347 performance virtual clusters.

3348 With high performance virtual clusters, VM instances that can gain near-native
3349 performance are needed. Furthermore, these VMs will be running tightly coupled
3350 applications, and, as such, must be allocated in a way in which communication latency
3351 is minimized and bandwidth is maximized for all VMs. This will help insure the
3352 entire virtual clusters can perform optimally. However, given off-the-shelf private
3353 cloud providers or, worse still, public cloud infrastructure, these requirements are at
3354 best opaque to the user, and at worst extremely suboptimal.

3355 One way in which high performance virtual clusters can operate efficiently is
3356 by defining specific instance types, or *flavors* within OpenStack, that define what
3357 resources a VM has allocated. Given previous research on the NUMA effects of
3358 VMs [257], these specialized flavors should be defined to fit within a NUMA socket.
3359 Furthermore, CPU pinning should be used to specifically keep the VM within the
3360 NUMA socket itself. Using Libvirt, a common API utilized in many cloud infrastruc-
3361 ture deployments (including OpenStack), we can specify CPU pinning directly in the

3362 XML configuration.

```

3363 <cputune>
3364   <vcpu pin vcpu="0" cpuset="0"/>
3365   <vcpu pin vcpu="1" cpuset="1"/>
3366   <vcpu pin vcpu="2" cpuset="4"/>
3367   <vcpu pin vcpu="3" cpuset="5"/>
3368   <vcpu pin vcpu="4" cpuset="6"/>
3369   <emulator pin cpuset="2"/>
3370 </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 7.6 Adding extra specs to a VM flavor in OpenStack

3371 With OpenStack, this NUMA configuration can be executed through the KVM
 3372 Nova plugin and a scheduling filter, which defines how to place VMs effectively. Fur-
 3373 thermore, the instance flavor can also specify the addition of `instance_type_extra_specs`
 3374 within Nova, whereby specialized hardware such as GPUs and InfiniBand intercon-
 3375 nects (as described in Chapters 5 and 6) can be passed through to the VMs directly.
 3376 Once defined, the same specialized high performance flavors in OpenStack simply have
 3377 to add the labels (such as 'gpu' or 'infiniband') to the flavor to gain the requested
 3378 hardware, as illustrated in Figure 7.6 with OpenStack Horizon's UI interface. The
 3379 implementation put forth in the OpenStack Havana build has since been upgraded by

3380 Intel with SR-IOV support and additional scheduling filter additions, and is available
3381 in the latest OpenStack releases [258].

```
3382 pci_passthrough_devices=[{"label":"gpu","address":"0000:08:00.0"},  
3383 {"label":"infiniband","address":"0000:21:00.0"}]  
3384 instance_type_extra_specs={'pci_passthrough:labels': ['gpu']}  
3385 instance_type_extra_specs={'pci_passthrough:labels': ['infiniband']}
```

3386 To provide a space for high performance virtual clusters, we need a scheduling
3387 mechanism within a cloud infrastructure to support the effective and proper place-
3388 ment beyond controlling for NUMA characteristics. While there are many effective
3389 scheduling algorithms for workload placement within an environment, a Proximity
3390 Scheduler [41], as defined in OpenStack, may work well. Specifically, one could either
3391 define or compute the underlying cloud infrastructure network topology. This could
3392 be as simple as a YAML file that defines an underlying InfiniBand interconnect 2-1
3393 Fat Tree topology, or a more complex solution that utilizes network performance met-
3394 rics to measure bandwidth and latency between disjoint nodes to build a weighted
3395 proximity graph. As it is possible that such network parameters could change due
3396 to other usage or to reconfiguration, a method of periodically monitoring and up-
3397 dating this proximity network using measurement tools such as PerfSonar [259] may
3398 also help keep an effective proximity metric between hosts. With a metric, one can
3399 then apply a proximity scheduler to handle high performance virtual cluster alloca-
3400 tion requests effectively and in a way that will be far more optimal than a simple
3401 round robin scheduling mechanism. The OpenStack private cloud IaaS framework
3402 is proposed for this effort, however, further development is needed to bring such a
3403 scheduling mechanism to fruition.

3404 The use of service level agreements (SLAs) within cloud infrastructure allocation

3405 is a well studied aspect [260]. It is also possible that certain SLAs could be in-
3406 corporated into a private cloud infrastructure such as OpenStack to simultaneously
3407 guarantee performance for virtual clusters with the above defined methods while con-
3408 currently offering "classical" VM workload scheduling for HTC, big data, or other
3409 cloud usage models. This would provide the same user experience yet support diverse
3410 workloads and performance expectations as defined by a given SLA. As it is likely
3411 such performance-tuned cloud infrastructure deployments as proposed in this disser-
3412 tation will likely still be used for traditional cloud workloads, it is advantageous to
3413 leverage SLAs in this way.

3414 **7.6 Chapter Summary**

3415 In summary, we expect the combination of transparent huge pages, an RDMA-capable
3416 interconnect multiplexed in hardware for use by both guest and hosts, a high per-
3417 formant hypervisor, and post-copy migration and cloning mechanisms, to enable a
3418 novel architecture for high performance virtual clusters. These mechanisms, if im-
3419 plemented and properly managed within a performance oriented scheduling system,
3420 could help change how cloud infrastructure supports HPC and big data applications,
3421 where performance, usability, and reconfigurability all are available. These migration
3422 and specialized environment support capabilities may also help enable new runtime
3423 systems for large scale scientific applications.

3424 **Chapter 8**

3425 **Conclusion**

3426 With the advent of virtualization and the availability of virtual machines through
3427 cloud infrastructure, a paradigm shift in distributed systems has occurred. Many
3428 services and applications once deployed on workstations, private servers, personal
3429 computers, and even some supercomputers, have migrated to a cloud infrastructure.
3430 The reasons for this change are vast; however, these reasons are not enough to support
3431 all computational challenges within such a virtualized infrastructure.

3432 The use of tightly coupled, distributed memory applications common in High Per-
3433 formance Computing communities has seen a number of problems and complications
3434 when deployed in virtualized infrastructure. While the reasons for this can be nu-
3435 merous, many challenges stem from the performance impact and overhead associated
3436 with virtualization, along with a lack of hardware necessary to support such con-
3437 current environments. This dissertation looks to evaluate virtualization's ability to
3438 support mid-tier scientific HPC applications, and what needs to take place to make
3439 high performance virtualization a reality.

3440 From the beginning, this dissertation proposes the advent of high performance
3441 virtual clusters to support a wide array of scientific computation, including mid-tier

3442 HPC applications, as well as a framework for building such an environment. This
3443 framework aims at identifying virtualization overhead and finding solutions and best
3444 practices with performant hypervisors, providing support for advanced accelerators
3445 and interconnects to enable new class of applications, and evaluating potential meth-
3446 ods using benchmarks and real-world applications. We propose to use the OpenStack
3447 IaaS project to encompass components together in a unified private cloud architec-
3448 ture.

3449 Chapter 2 studied the related research necessary for defining not only the context
3450 for virtualization and cloud computing, but also virtual clusters and their history
3451 through supercomputing. Chapter 3 looked to study the applicability of various hy-
3452 pervisors for supporting common HPC workloads through the use of benchmarks from
3453 a single-node aspect. This found challenges and some solutions to these workloads,
3454 and identified missing gaps that exist.

3455 Chapter 4 started the investigation of the utility of GPUs to support mid-tier
3456 scientific applications using the Xen hypervisor. This chapter provided a proof-of-
3457 concept that with proper configuration by utilizing the latest in hardware support,
3458 GPU passthrough was possible and a viable model for supporting CUDA-enabled
3459 applications, a fast-growing application set. Chapter 5 provides an in-depth com-
3460 parison of multiple hypervisors using the SHOC GPU benchmark suite, as well as
3461 GPU-enabled HPC applications. Here we discover our KVM implementation per-
3462 forms at near-native speeds and allows for effective GPU utilization.

3463 Chapter 6 takes the lessons learned with KVM in GPU passthrough and adds in
3464 SR-IOV InfiniBand support, a critical tool for supporting tightly coupled distributed
3465 memory applications, to build a small virtual cluster. This environment supports
3466 two class-leading Molecular Dynamics simulations, LAMMPS and HOOMD-blue, and
3467 shows how both applications can not only perform at near-native speeds, but also

3468 leverage the latest HPC technologies such as GPUDirect for efficient GPU-to-GPU
3469 communication.

3470 Chapter 7 is an introspective look at other advancements that can be made in
3471 virtualization to support high performance virtual clusters. Specifically, this chap-
3472 ter details the utility of virtual clusters backed with hugepages, added support for
3473 specialized live migration techniques leveraging high speed RDMA-capable intercon-
3474 nects, VM cloning for fast deployment of virtual clusters themselves, and scheduling
3475 considerations for integration of high performance virtual clusters in OpenStack.

3476 **8.1 Impact**

3477 This dissertation has illustrated how virtualization can be used to support HPC ap-
3478 plications using virtual clusters. While the example applications used herein are in
3479 relation to Molecular Dynamics, it is anticipated that this work is also equally appli-
3480 cable to other fields including Astronomy, High Energy Physics, Bioinformatics, and
3481 computational chemistry, to name a few. It is also possible that such applications can
3482 scale with future infrastructure deployments, however further study will be necessary
3483 to confirm these assumptions.

3484 The model for PCI passthrough may also be able to impact other hardware. First,
3485 this could include the Intel Xeon Phi (coprocessor models, not the new Knights Land-
3486 ing CPU), or some emerging FPGA implementations like the Stillwater Knowledge
3487 Processing Unit (KPU), a distributed data flow processor. The PCI-Express has been
3488 upgraded in spec 4.0 to support larger I/O devices and accelerators with large power
3489 deliver on-bus, leading to the assumption that, for at least commodity x86 systems,
3490 there could be an increase in device utilization. The caveat to this will be if the PCIE
3491 bus is abandoned or superseeded by other methods, such as System-on-Chip designs

3492 or Nvidia's NVLink effort. While the HPC accelerator usage could very well wane in
3493 the wake of novel many core architectures such as Knights Landing, such movements
3494 have still yet to take place within the HPC community.

3495 As some of the advances described in the dissertation have already made their way
3496 to the OpenStack cloud platform. With that, it may be possible to build such a cloud
3497 infrastructure to run high performance virtual clusters at a larger scale. Applying this
3498 infrastructure, along with high level experiment management and support services,
3499 could lead to a new national scale cyberinfrastructure deployment. In time and with
3500 further development, this could be deployed within the NSF-funded XSEDE project.

3501 Bibliography

- 3502 [1] B. Alexander, “Web 2.0: A New Wave of Innovation for Teaching and Learn-
3503 ing?” *Learning*, vol. 41, no. 2, pp. 32–44, 2006.
- 3504 [2] R. Buyya, C. S. Yeo, and S. Venugopal, “Market-oriented cloud computing:
3505 Vision, hype, and reality for delivering it services as computing utilities,” in
3506 *Proceedings of the 10th IEEE International Conference on High Performance*
3507 *Computing and Communications (HPCC-08, IEEE CS Press, Los Alamitos,*
3508 *CA, USA)*, 2008, pp. 5–13.
- 3509 [3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Comput-
3510 ing 360-Degree Compared,” in *Grid Computing Environments Workshop, 2008.*
3511 *GCE’08*, 2008, pp. 1–10.
- 3512 [4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski,
3513 G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “Above
3514 the clouds: A berkeley view of cloud computing,” University of
3515 California at Berkeley, Tech. Rep., February 2009. [Online]. Available:
3516 <http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html>
- 3517 [5] T. Kuhn, *The structure of scientific revolutions*. University of Chicago press
3518 Chicago, 1970.

- 3519 [6] T. Sterling, *Beowulf cluster computing with Linux*. The MIT Press, 2001.
- 3520 [7] T. Hoare and R. Milner, “Grand challenges for computing research,” *The Computer Journal*, vol. 48, no. 1, pp. 49–52, 2005.
- 3521
3522 [8] J. Dongarra, H. Meuer, and E. Strohmaier, “Top 500 supercomputers,” website, November 2013.
- 3523
3524 [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.
- 3525
3526
3527 [10] L.-W. Wang, “A survey of codes and algorithms used in nersc material science allocations,” *Lawrence Berkeley National Laboratory*, 2006.
- 3528
3529 [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.
- 3530
3531
3532
3533 [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].
3534
3535
3536 Available: <http://doi.acm.org/10.1145/1851476.1851535>
- 3537
3538 [13] A. S. Bland, J. Wells, O. E. Messer, O. Hernandez, and J. Rogers, “Titan: Early experience with the cray xk6 at oak ridge national laboratory,” *Cray User Group*, 2012.
- 3539
3540

- 3541 [14] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazle-
3542 wood, S. Lathrop, D. Lifka, G. D. Peterson *et al.*, “Xsede: accelerating scientific
3543 discovery,” *Computing in Science & Engineering*, vol. 16, no. 5, pp. 62–74, 2014.
- 3544 [15] S. A. Goff, M. Vaughn, S. McKay, E. Lyons, A. E. Stapleton, D. Gessler,
3545 N. Matasci, L. Wang, M. Hanlon, A. Lenards *et al.*, “The iplant collaborative:
3546 cyberinfrastructure for plant biology,” *Frontiers in plant science*, vol. 2, p. 34,
3547 2011.
- 3548 [16] K. Antypas, “Nersc-6 workload analysis and benchmark selection process,”
3549 *Lawrence Berkeley National Laboratory*, 2008.
- 3550 [17] J. S. Vetter, R. Glassbrook, J. Dongarra, K. Schwan, B. Loftis, S. McNally,
3551 J. Meredith, J. Rogers, P. Roth, K. Spafford *et al.*, “Keeneland: Bringing het-
3552 erogeneous gpu computing to the computational science community,” *Comput-
3553 ing in Science and Engineering*, vol. 13, no. 5, pp. 90–95, 2011.
- 3554 [18] P. Pacheco, *Parallel Programming with MPI*, ser. ISBN. Morgan Kaufmann,
3555 October 1996, no. 978-1-55860-339-4.
- 3556 [19] G. C. Fox, S. W. Otto, and A. J. Hey, “Matrix algorithms on a hypercube i:
3557 Matrix multiplication,” *Parallel computing*, vol. 4, no. 1, pp. 17–31, 1987.
- 3558 [20] D. Agrawal, S. Das, and A. El Abbadi, “Big data and cloud computing: cur-
3559 rent state and future opportunities,” in *Proceedings of the 14th International
3560 Conference on Extending Database Technology*. ACM, 2011, pp. 530–533.
- 3561 [21] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large
3562 clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

- 3563 [22] S. Kamburugamuve, G. Fox, D. Leake, and J. Qiu, "Survey of apache big data
3564 stack," Ph.D. dissertation, Ph. D. Qualifying Exam, Dept. Inf. Comput., Indiana Univ., Bloomington, IN, 2013.
3565
- 3566 [23] M. Chen, S. Mao, and Y. Liu, "Big data: a survey," *Mobile Networks and*
3567 *Applications*, vol. 19, no. 2, pp. 171–209, 2014.
- 3568 [24] G. Bell, T. Hey, and A. Szalay, "Beyond the data deluge," *Science*, vol. 323,
3569 no. 5919, pp. 1297–1298, 2009.
- 3570 [25] D. A. Reed and J. Dongarra, "Exascale computing and big data," *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.
- 3571
- 3572 [26] S. Jha, J. Qiu, A. Luckow, P. K. Mantha, and G. C. Fox, "A tale of two
3573 data-intensive paradigms: Applications, abstractions, and architectures," in
3574 *Proceedings of the 3rd International Congress on Big Data*, 2014.
- 3575 [27] J. Qiu, S. Jha, A. Luckow, and G. C. Fox, "Towards hpc-abds: An initial high-
3576 performance big data stack," *Building Robust Big Data Ecosystem ISO/IEC*
3577 *JTC 1 Study Group on Big Data*, pp. 18–21, 2014.
- 3578 [28] M. W. ur Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and
3579 D. K. D. Panda, "High-performance rdma-based design of hadoop mapreduce
3580 over infiniband," in *Parallel and Distributed Processing Symposium Workshops*
3581 *PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 1908–
3582 1917.
- 3583 [29] S. Ekanayake, S. Kamburugamuve, and G. Fox, "Spidal: High performance data
3584 analytics with java and mpi on large multicore hpc clusters," in *Proceedings of*
3585 *24th High Performance Computing Symposium (HPC 2016)*, 2016.

- 3586 [30] F. Tian and K. Chen, “Towards optimal resource provisioning for running
3587 mapreduce programs in public clouds,” in *Cloud Computing (CLOUD), 2011*
3588 *IEEE International Conference on*. IEEE, 2011, pp. 155–162.
- 3589 [31] J. Lange, K. Pedretti, T. Hudson, P. Dinda, Z. Cui, L. Xia, P. Bridges, A. Gocke,
3590 S. Jaconette, M. Levenhagen *et al.*, “Palacios and kitten: New high performance
3591 operating systems for scalable virtualized and native supercomputing,” in *Par-*
3592 *allel & Distributed Processing (IPDPS), 2010 IEEE International Symposium*
3593 *on*. IEEE, 2010, pp. 1–12.
- 3594 [32] I. Foster, T. Freeman, K. Keahy, D. Scheftner, B. Sotomayer, and X. Zhang,
3595 “Virtual clusters for grid communities,” *Cluster Computing and the Grid, IEEE*
3596 *International Symposium on*, vol. 0, pp. 513–520, 2006.
- 3597 [33] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and
3598 S. Tuecke, “A resource management architecture for metacomputing systems,”
3599 in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer,
3600 1998, pp. 62–82.
- 3601 [34] D. Kondo, B. Javadi, P. Malecot, F. Cappello, and D. P. Anderson, “Cost-
3602 benefit analysis of cloud computing versus desktop grids,” in *Parallel & Dis-*
3603 *tributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*.
3604 IEEE, 2009, pp. 1–12.
- 3605 [35] T. Bell, B. Bompastor, S. Bukowiec, J. C. Leon, M. Denis, J. van Eldik, M. F.
3606 Lobo, L. F. Alvarez, D. F. Rodriguez, A. Marino *et al.*, “Scaling the cern
3607 openstack cloud,” in *Journal of Physics: Conference Series*, vol. 664, no. 2.
3608 IOP Publishing, 2015, p. 022003.

- 3609 [36] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, “Mapreduce in the clouds for
3610 science,” in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE*
3611 *Second International Conference on.* IEEE, 2010, pp. 565–572.
- 3612 [37] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema,
3613 “A performance analysis of ec2 cloud computing services for scientific comput-
3614 ing,” in *International Conference on Cloud Computing.* Springer, 2009, pp.
3615 115–131.
- 3616 [38] J. Dongarra *et al.*, “The international exascale software project roadmap,”
3617 *International Journal of High Performance Computing Applications*, p.
3618 1094342010391989, 2011.
- 3619 [39] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau,
3620 P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. K. and Dean Klein,
3621 R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S.
3622 Williams, and K. Yelick, “Exascale computing study: Technology challenges
3623 in achieving exascale systems,” *Defense Advanced Research Projects Agency*
3624 *Information Processing Techniques Office (DARPA IPTO), Tech. Rep*, vol. 15,
3625 2008.
- 3626 [40] J. Shalf, S. Dosanjh, and J. Morrison, “Exascale computing technology chal-
3627 lenges,” in *International Conference on High Performance Computing for Com-
3628 putational Science.* Springer, 2010, pp. 1–25.
- 3629 [41] J. Suh, “Proximity scheduler in openstack,” Webpage. [Online]. Available:
3630 <https://wiki.openstack.org/wiki/ProximityScheduler>
- 3631 [42] S. Kamburugamuve, S. Ekanayake, M. Pathirage, and G. Fox, “Towards high
3632 performance processing of streaming data in large data centers,” in *HPBDC*

- 3633 *2016 IEEE International Workshop on High-Performance Big Data Comput-*
3634 *ing in conjunction with The 30th IEEE International Parallel and Distributed*
3635 *Processing Symposium (IPDPS 2016), Chicago, Illinois USA, Friday, 2016.*
- 3636 [43] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, and
3637 G. Pike, “Design of the FutureGrid Experiment Management Framework,”
3638 in *Proceedings of Gateway Computing Environments 2010 at Supercomputing*
3639 *2010.* New Orleans, LA: IEEE, Nov 2010. [Online]. Available: <http://grids.ucs.indiana.edu/ptliupages/publications/vonLaszewski-10-FG-exp-GCE10.pdf>
- 3641 [44] S. Drake and O. development team, “Heat: OpenStack Orchestration,”
3642 Webpage. [Online]. Available: <https://wiki.openstack.org/wiki/Heat>
- 3643 [45] G. Von Laszewski, F. Wang, H. Lee, H. Chen, and G. C. Fox, “Accessing
3644 multiple clouds with cloudmesh,” in *Proceedings of the 2014 ACM international*
3645 *workshop on Software-defined ecosystems.* ACM, 2014, pp. 21–28.
- 3646 [46] “The magellan project,” Webpage. [Online]. Available: <http://magellan.alcf.anl.gov/>
- 3648 [47] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon, “The Magellan Report on
3649 Cloud Computing for Science,” U.S. Department of Energy Office of Science
3650 Office of Advanced Scientific Computing Research (ASCR), Tech. Rep., Dec.
3651 2011.
- 3652 [48] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf,
3653 H. Wasserman, and N. Wright, “Performance Analysis of High Performance
3654 Computing Applications on the Amazon Web Services Cloud,” in *2nd IEEE*
3655 *International Conference on Cloud Computing Technology and Science.* IEEE,
3656 2010, pp. 159–168.

- 3657 [49] D. Merkel, “Docker: lightweight linux containers for consistent development
3658 and deployment,” *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.
- 3659 [50] D. M. Jacobsen and R. S. Canon, “Contain this, unleashing docker for hpc,”
3660 *Proceedings of the Cray User Group*, 2015.
- 3661 [51] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A.
3662 De Rose, “Performance evaluation of container-based virtualization for high
3663 performance computing environments,” in *2013 21st Euromicro International
3664 Conference on Parallel, Distributed, and Network-Based Processing*. IEEE,
3665 2013, pp. 233–240.
- 3666 [52] K. Hwang, J. Dongarra, and G. C. Fox, *Distributed and cloud computing: from
3667 parallel processing to the internet of things*. Morgan Kaufmann, 2013.
- 3668 [53] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neuge-
3669 bauer, I. Pratt, and A. Warfield, “Xen and the art of virtualization,” in *Pro-
3670 ceedings of the 19th ACM Symposium on Operating Systems Principles*, New
3671 York, U. S. A., Oct. 2003, pp. 164–177.
- 3672 [54] “Vmware esx server,” [Online], <http://www.vmware.com/de/products/vi/esx/>.
- 3673 [55] J. Duato, A. J. Pena, F. Silla, J. C. Fernández, R. Mayo, and E. S. Quintana-
3674 Orti, “Enabling CUDA acceleration within virtual machines using rCUDA,” in
3675 *High Performance Computing (HiPC), 2011 18th International Conference on*.
3676 IEEE, 2011, pp. 1–10.
- 3677 [56] L. Shi, H. Chen, J. Sun, and K. Li, “vCUDA: GPU-accelerated high-
3678 performance computing in virtual machines,” *Computers, IEEE Transactions
3679 on*, vol. 61, no. 6, pp. 804–816, 2012.

- 3680 [57] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, “kvm: the Linux
3681 virtual machine monitor,” in *Proceedings of the Linux Symposium*, vol. 1, 2007,
3682 pp. 225–230.
- 3683 [58] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee,
3684 D. Patterson, A. Rabkin, I. Stoica *et al.*, “A view of cloud computing,” *Com-*
3685 *munications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.
- 3686 [59] I. Foster, C. Kesselman, and S. Tuecke, “The Anatomy of the Grid: Enabling
3687 Scalable Virtual Organizations,” *Intl. J. Supercomputer Applications*, vol. 15,
3688 no. 3, 2001.
- 3689 [60] I. Foster, C. Kesselman *et al.*, “The Physiology of the Grid:An Open Grid
3690 Services Architecture for Distributed Systems Integration,” Argonne National
3691 Laboratory, Chicago, Tech. Rep., Jan. 2002.
- 3692 [61] D. DiNucci, “Fragmented future,” *AllBusiness-Champions of Small Business*,
3693 1999. [Online]. Available: <http://www.cdinucci.com/Darcy2/articles/Print/Printarticle7.html>
- 3695 [62] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard,
3696 S. Pogliani, K. Riemer, S. Struble, P. Takacs-Nagy, I. Trickovic, and S. Zimek,
3697 “Web Services Choreography Interface,” Jun. 2002, version 1.0. [Online].
3698 Available: <http://wwws.sun.com/software/xml/developers/wsci/index.html>
- 3699 [63] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architec-*
3700 *ture Best Practices (The Coad Series)*. Prentice Hall PTR Upper Saddle
3701 River, NJ, USA, 2004.
- 3702 [64] L. Wang, G. von Laszewski, A. J. Younge, X. He, M. Kunze, and J. Tao,
3703 “Cloud Computing: a Perspective Study,” *New Generation Computing*, vol. 28,

- 3704 pp. 63–69, Mar 2010. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/10-cloudcomputing-NGC/vonLaszewski-10-NGC.pdf>
- 3705
- 3706 [65] G. von Laszewski, A. J. Younge, X. He, K. Mahinthakumar, and
3707 L. Wang, “Experiment and Workflow Management Using Cyberaide
3708 Shell,” in *Proceedings of the 4th International Workshop on Workflow
3709 Systems in e-Science (WSES 09) with 9th IEEE/ACM International
3710 Symposium on Cluster Computing and the Grid (CCGrid 09)*. IEEE,
3711 May 2009. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/09-gridshell-ccgrid/vonLaszewski-ccgrid09-final.pdf>
- 3712
- 3713 [66] G. von Laszewski, F. Wang, A. J. Younge, X. He, Z. Guo, and M. Pierce,
3714 “Cyberaide JavaScript: A JavaScript Commodity Grid Kit,” in *Proceedings
3715 of the Grid Computing Environments 2007 at Supercomputing 2008*. Austin,
3716 TX: IEEE, Nov 2008. [Online]. Available: <http://cyberaide.googlecode.com/svn/trunk/papers/08-javascript/vonLaszewski-08-javascript.pdf>
- 3717
- 3718 [67] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, “Comparison of multiple
3719 cloud frameworks,” in *Cloud Computing (CLOUD), 2012 IEEE 5th Interna-
3720 tional Conference on*, June 2012, pp. 734–741.
- 3721
- 3722 [68] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing
systems,” *INC, IMS and IDC*, pp. 44–51, 2009.
- 3723
- 3724 [69] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Virtual infrastruc-
3725 ture management in private and hybrid clouds,” *IEEE Internet Computing*,
vol. 13, no. 5, pp. 14–22, 2009.
- 3726
- 3727 [70] S. A. Baset, “Cloud slas: present and future,” *ACM SIGOPS Operating Systems
Review*, vol. 46, no. 2, pp. 57–66, 2012.

- 3728 [71] “Amazon Elastic Compute Cloud.” [Online]. Available: <http://aws.amazon.com/ec2/>
- 3729
- 3730 [72] S. Krishnan and J. L. U. Gonzalez, “Google compute engine,” in *Building Your*
3731 *Next Big Thing with Google Cloud Platform*. Springer, 2015, pp. 53–81.
- 3732 [73] “Nimbus Project,” <http://www.nimbusproject.org>. Last access Mar. 2011.
- 3733 [74] K. Keahey, I. Foster, T. Freeman, and X. Zhang, “Virtual workspaces: Achiev-
3734 ing quality of service and quality of life in the grid,” *Scientific Programming*
3735 *Journal*, vol. 13, no. 4, pp. 265–276, 2005.
- 3736 [75] “Amazon web services s3 rest api,” <http://awsdocs.s3.amazonaws.com/S3/latest/s3-api.pdf>. Last Access Mar. 2011.
- 3737
- 3738 [76] “Jets3t project,” <http://bitbucket.org/jmurty/jets3t/wiki/Home>. Last Access
3739 Mar. 2011.
- 3740 [77] “Boto project,” <http://code.google.com/p/boto>. Last Access Mar. 2011.
- 3741 [78] “S3tools project,” <http://s3tools.org/s3cmd>. Last Access Mar. 2011.
- 3742 [79] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and
3743 D. Zagorodnov, “The Eucalyptus Open-source Cloud-computing System,” *Pro-*
3744 *ceedings of Cloud Computing and Its Applications*, 2008.
- 3745 [80] “Eucalyptus open-source cloud computing infrastructure,” an Overview, Euca-
3746 lypts Systems, Inc. 2009.
- 3747 [81] “Eucalyptus,” <http://www.eucalyptus.com>, Last Access Mar. 2011.

- 3748 [82] I. H. Shaon, “Eucalyptus and its components,” Webpage.
3749 [Online]. Available: <https://mdshaonimran.wordpress.com/2011/11/26/eucalyptus-and-its-components>
- 3750
- 3751 [83] “Openstack. cloud software,” <http://openstack.org>, Last Access Mar. 2011.
- 3752 [84] O. Sefraoui, M. Aissaoui, and M. Eleuldj, “Openstack: toward an open-source
3753 solution for cloud computing,” *International Journal of Computer Applications*,
3754 vol. 55, no. 3, 2012.
- 3755 [85] “Opennebula project,” <http://www.opennebula.org>. Last access Mar. 2011.
- 3756 [86] I. M. Llorente, R. Moreno-Vozmediano, and R. S. Montero, “Cloud computing
3757 for on-demand grid resource provisioning,” *Advances in Parallel Computing*,
3758 vol. 18, no. 5, pp. 177–191, 2009.
- 3759 [87] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Capacity leasing
3760 in cloud systems using the opennebula engine,” *Cloud Computing and its
3761 Applications*, 2008.
- 3762 [88] “Libvirt API webpage,” <http://libvirt.org>. Last access Mar. 2011.
- 3763 [89] “Elastichosts webpage,” <http://www.elastichosts.com>. Last Access Mar. 2011.
- 3764 [90] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Mat-
3765 sunaga, M. Tsugawa, J. Zhang, M. Zhao *et al.*, “From virtualized resources
3766 to virtual computing grids: the in-vigo system,” *Future Generation Computer
3767 Systems*, vol. 21, no. 6, pp. 896–909, 2005.
- 3768 [91] J. Chase, D. Irwin, L. Grit, J. Moore, and S. Sprenkle, “Dynamic virtual clus-
3769 ters in a grid site manager,” in *12th IEEE International Symposium on High
3770 Performance Distributed Computing, 2003. Proceedings*, 2003, pp. 90–100.

- 3771 [92] I. VMware, “VMware vCloud Air,” Webpage. [Online]. Available: <http://vcloud.vmware.com>
- 3772
- 3773 [93] CERN, “LHC Computing Grid Project,” Web Page, Dec. 2003. [Online].
3774 Available: <http://lcg.web.cern.ch/LCG/>
- 3775 [94] J. Luo, A. Song, Y. Zhu, X. Wang, T. Ma, Z. Wu, Y. Xu, and L. Ge, “Grid sup-
3776 porting platform for AMS data processing,” *Lecture notes in computer science*,
3777 vol. 3759, p. 276, 2005.
- 3778 [95] “CMS,” Web Page. [Online]. Available: <http://cms.cern.ch/>
- 3779 [96] J. Diaz, A. J. Younge, G. von Laszewski, F. Wang, and G. C. Fox, “Grappling
3780 Cloud Infrastructure Services with a Generic Image Repository,” in *Proceedings*
3781 of *Cloud Computing and Its Applications (CCA 2011)*, Argonne, IL, Mar 2011.
- 3782 [97] I. Foster, “The anatomy of the grid: Enabling scalable virtual organizations,”
3783 in *Euro-Par 2001 Parallel Processing: 7th International Euro-Par Conference*.
3784 Springer, 2001, pp. 1–5. [Online]. Available: www.globus.org/alliance/publications/papers/anatomy.pdf
- 3785
- 3786 [98] K. Keahey and T. Freeman, “Contextualization: Providing one-click virtual
3787 clusters,” in *eScience, 2008. eScience’08. IEEE Fourth International Confer-
3788 ence on*. IEEE, 2008, pp. 301–308.
- 3789 [99] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos,
3790 W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni, R. P. Wagner, N. Wilkins-
3791 Diehr, and M. L. Norman, “Gateways to discovery: Cyberinfrastructure
3792 for the long tail of science,” in *Proceedings of the 2014 Annual Conference
3793 on Extreme Science and Engineering Discovery Environment*, ser. XSEDE

- 3794 '14. New York, NY, USA: ACM, 2014, pp. 39:1–39:8. [Online]. Available:
3795 <http://doi.acm.org/10.1145/2616498.2616540>
- 3796 [100] D. Bernstein, “Containers and cloud: From lxc to docker to kubernetes,” *IEEE
3797 Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.
- 3798 [101] F. Berman, “From TeraGrid to knowledge grid,” *Communications of the ACM*,
3799 vol. 44, no. 11, pp. 27–28, 2001.
- 3800 [102] C. Catlett, “The philosophy of TeraGrid: building an open, extensible, dis-
3801 tributed TeraScale facility,” in *2nd IEEE/ACM International Symposium on
3802 Cluster Computing and the Grid, 2002*, 2002, pp. 8–8.
- 3803 [103] H. H. Goldstine and A. Goldstine, “The electronic numerical integrator and
3804 computer (eniac),” *Mathematical Tables and Other Aids to Computation*, vol. 2,
3805 no. 15, pp. 97–110, 1946.
- 3806 [104] J. E. Thornton, “Design of a computer - the control data 6600,” 1970.
- 3807 [105] R. M. Russell, “The cray-1 computer system,” *Communications of the ACM*,
3808 vol. 21, no. 1, pp. 63–72, 1978.
- 3809 [106] C. L. Seitz, “The cosmic cube,” *Communications of the ACM*, vol. 28, no. 1,
3810 pp. 22–33, 1985.
- 3811 [107] W. D. Hillis, *The connection machine*. MIT press, 1989.
- 3812 [108] X. Zhang, C. Yang, F. Liu, Y. Liu, and Y. Lu, “Optimizing and scaling hpcg
3813 on tianhe-2: early experience,” in *International Conference on Algorithms and
3814 Architectures for Parallel Processing*. Springer, 2014, pp. 28–41.

- 3815 [109] A. Geist, *PVM: Parallel virtual machine: a users' guide and tutorial for net-*
3816 *worked parallel computing.* MIT press, 1994.
- 3817 [110] B. Carpenter, V. Getov, G. Judd, A. Skjellum, and G. C. Fox, “Mpj: Mpi-like
3818 message passing for java,” 2000.
- 3819 [111] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres,
3820 V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, “Open mpi: Goals,
3821 concept, and design of a next generation mpi implementation,” in *European
3822 Parallel Virtual Machine/Message Passing Interface Users Group Meeting.*
3823 Springer, 2004, pp. 97–104.
- 3824 [112] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming
3825 with the message-passing interface.* MIT press, 1999, vol. 1.
- 3826 [113] M. J. Koop, T. Jones, and D. K. Panda, “Mvapich-aptus: Scalable high-
3827 performance multi-transport mpi over infiniband,” in *IEEE International Sym-
3828 posium on Parallel and Distributed Processing, 2008. IPDPS 2008.* IEEE,
3829 2008, pp. 1–12.
- 3830 [114] R. Rabenseifner, G. Hager, and G. Jost, “Hybrid mpi/openmp parallel program-
3831 ming on clusters of multi-core smp nodes,” in *2009 17th Euromicro international
3832 conference on parallel, distributed and network-based processing.* IEEE, 2009,
3833 pp. 427–436.
- 3834 [115] D. A. Jacobsen, J. C. Thibault, and I. Senocak, “An mpi-cuda implementation
3835 for massively parallel incompressible flow computations on multi-gpu clusters,”
3836 in *48th AIAA aerospace sciences meeting and exhibit*, vol. 16, 2010, p. 2.

- 3837 [116] J. J. Dongarra, P. Luszczek, and A. Petitet, “The linpack benchmark: past,
3838 present and future,” *Concurrency and Computation: practice and experience*,
3839 vol. 15, no. 9, pp. 803–820, 2003.
- 3840 [117] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, “Introducing the
3841 graph 500,” *Cray Users Group (CUG)*, 2010.
- 3842 [118] M. A. Heroux and J. Dongarra, “Toward a new metric for ranking high perfor-
3843 mance computing systems,” *Sandia Report, SAND2013-4744*, vol. 312, 2013.
- 3844 [119] R. Brightwell, R. Oldfield, A. B. Maccabe, and D. E. Bernholdt, “Hobbes:
3845 Composition and virtualization as the foundations of an extreme-scale os/r,”
3846 in *Proceedings of the 3rd International Workshop on Runtime and Operating*
3847 *Systems for Supercomputers*. ACM, 2013, p. 2.
- 3848 [120] S. Perarnau, R. Gupta, and P. Beckman, “Argo: An exascale operating system
3849 and runtime,” in *Poster Proceedings from IEEE/ACM Supercomputing 2015*,
3850 2015.
- 3851 [121] T. Sterling, D. Kogler, M. Anderson, and M. Brodowicz, “SLOWER: A perfor-
3852 mance model for Exascale computing,” *Supercomputing Frontiers and Innova-*
3853 *tions*, vol. 1, pp. 42–57, Sep 2014.
- 3854 [122] E. Ciurana, *Developing with Google App Engine*. Springer, 2009.
- 3855 [123] D. Chappell, “Introducing windows azure,” Microsoft, Inc, Tech. Rep., 2009.
- 3856 [124] K. Keahey, R. Figueiredo, J. Fortes, T. Freeman, and M. Tsugawa, “Science
3857 clouds: Early experiences in cloud computing for scientific applications,” *Cloud*
3858 *Computing and Applications*, vol. 2008, 2008.

- 3859 [125] R. Creasy, “The origin of the VM/370 time-sharing system,” *IBM Journal of*
3860 *Research and Development*, vol. 25, no. 5, pp. 483–490, 1981.
- 3861 [126] “Amazon elastic compute cloud,” [Online], <http://aws.amazon.com/ec2/>.
- 3862 [127] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron, “Virtual
3863 Workspaces in the Grid,” *Lecture Notes in Computer Science*, vol. 3648,
3864 pp. 421–431, 2005. [Online]. Available: http://workspace.globus.org/papers/VW_EuroPar05.pdf
- 3866 [128] J. Fontan, T. Vazquez, L. Gonzalez, R. S. Montero, and I. M. Llorente, “Open-
3867 NEbula: The Open Source Virtual Machine Manager for Cluster Computing,”
3868 in *Open Source Grid and Cluster Software Conference*, San Francisco, CA, USA,
3869 May 2008.
- 3870 [129] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Mat-
3871 sunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu, “From virtualized
3872 resources to virtual computing Grids: the In-VIGO system,” *Future Generation
3873 Comp. Syst.*, vol. 21, no. 6, pp. 896–909, 2005.
- 3874 [130] Rackspace, “Openstack,” WebPage, Jan 2011. [Online]. Available: <http://www.openstack.org/>
- 3876 [131] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. Shin, “Performance evaluation
3877 of virtualization technologies for server consolidation,” HP Laboratories, Tech.
3878 Rep., 2007.
- 3879 [132] J. Watson, “Virtualbox: bits and bytes masquerading as machines,” *Linux
3880 Journal*, vol. 2008, no. 166, p. 1, 2008.

- 3881 [133] D. Leinenbach and T. Santen, “Verifying the Microsoft Hyper-V Hypervisor
3882 with VCC,” *FM 2009: Formal Methods*, pp. 806–809, 2009.
- 3883 [134] I. Parallels, “An introduction to os virtualization and paral-
3884 lels virtuozzo containers,” Parallels, Inc, Tech. Rep., 2010. [On-
3885 line]. Available: http://www.parallels.com/r/pdf/wp/pvc/Parallels_Virtuozzo_Containers_WP_an_introduction_to_os_EN.pdf
- 3887 [135] D. Bartholomew, “Qemu: a multihost, multitarget emulator,” *Linux Journal*,
3888 vol. 2006, no. 145, p. 3, 2006.
- 3889 [136] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamil-
3890 ton, M. McCabe, and J. Owens, “Quantifying the performance isolation prop-
3891 erties of virtualization systems,” in *Proceedings of the 2007 workshop on Ex-*
3892 *perimental computer science*, ser. ExpCS ’07. New York, NY, USA: ACM,
3893 2007.
- 3894 [137] Oracle, “Performance evaluation of oracle vm server virtualization software,”
3895 Oracle, Whitepaper, 2008. [Online]. Available: <http://www.oracle.com/us/technologies/virtualization/oraclevm/026997.pdf>
- 3897 [138] K. Adams and O. Agesen, “A comparison of software and hardware techniques
3898 for x86 virtualization,” in *Proceedings of the 12th international conference on*
3899 *Architectural support for programming languages and operating systems*. ACM,
3900 2006, pp. 2–13, vMware.
- 3901 [139] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, “An analy-
3902 sis of performance interference effects in virtual environments,” in *Performance*
3903 *Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Sym-*
3904 *posium on*. IEEE, 2007, pp. 200–209.

- 3905 [140] S. Rixner, “Network virtualization: Breaking the performance barrier,” *Queue*,
3906 vol. 6, no. 1, p. 36, 2008.
- 3907 [141] S. Nanda and T. Chiueh, “A survey of virtualization technologies,” Tech. Rep.,
3908 2005.
- 3909 [142] A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan, “Performance im-
3910 plications of virtualizing multicore cluster machines,” in *Proceedings of the 2nd*
3911 *workshop on System-level virtualization for high performance computing*. ACM,
3912 2008, pp. 1–8.
- 3913 [143] R. Harper and K. Rister, “Kvm limits arbitrary or architectural?” IBM Linux
3914 Technology Center, Jun 2009.
- 3915 [144] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, and A. Brinkmann, “Non-
3916 intrusive virtualization management using libvirt,” in *Design, Automation Test*
3917 *in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 574 –579.
- 3918 [145] “FutureGrid,” Web Page, 2009. [Online]. Available: <http://www.futuregrid.org>
- 3919 [146] J. J. Dujmovic and I. Dujmovic, “Evolution and evaluation of spec bench-
3920 marks,” *SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 3, pp. 2–9, 1998.
- 3921 [147] P. Luszczek, D. Bailey, J. Dongarra, J. Kepner, R. Lucas, R. Rabenseifner,
3922 and D. Takahashi, “The HPC Challenge (HPCC) benchmark suite,” in *SC06*
3923 *Conference Tutorial*. Citeseer, 2006.
- 3924 [148] J. Dongarra and P. Luszczek, “Reducing the time to tune parallel dense linear
3925 algebra routines with partial execution and performance modelling,” University
3926 of Tennessee Computer Science Technical Report, Tech. Rep., 2010.

- 3927 [149] K. Dixit, "The SPEC benchmarks," *Parallel Computing*, vol. 17, no. 10-11, pp.
3928 1195–1209, 1991.
- 3929 [150] SPEC, "Standard performance evaluation corporation," Webpage, Jan 2011.
3930 [Online]. Available: <http://www.spec.org/>
- 3931 [151] R. Henschel and A. J. Younge, "First quarter 2011 spec omp results," Webpage,
3932 Mar 2011. [Online]. Available: <http://www.spec.org/omp/results/res2011q1/>
- 3933 [152] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. Prentice Hall, 2002,
3934 vol. 2.
- 3935 [153] Amazon, "Elastic Compute Cloud." [Online]. Available: <http://aws.amazon.com/ec2/>
- 3936
- 3937 [154] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large
3938 clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.
- 3939 [155] "Windows azure platform," [Online], <http://www.microsoft.com/azure/>.
- 3940 [156] G. von Laszewski, G. C. Fox, F. Wang, A. J. Younge, A. Kulshrestha, G. G.
3941 Pike, W. Smith, J. Vockler, R. J. Figueiredo, J. Fortes *et al.*, "Design of the
3942 futuregrid experiment management framework," in *Gateway Computing Envi-
3943 ronments Workshop (GCE), 2010*. IEEE, 2010, pp. 1–10.
- 3944 [157] OpenStack, "Openstack compute administration manual," 2013.
- 3945 [158] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and
3946 D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in
3947 *Proceedings of Cloud Computing and Its Applications*, October 2008. [Online].
3948 Available: <http://eucalyptus.cs.ucsb.edu/wiki/Presentations>

- 3949 [159] C. Nvidia, “Programming guide,” 2008.
- 3950 [160] J. E. Stone, D. Gohara, and G. Shi, “OpenCL: A parallel programming standard
3951 for heterogeneous computing systems,” *Computing in science & engineering*,
3952 vol. 12, no. 3, p. 66, 2010.
- 3953 [161] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron,
3954 “A performance study of general-purpose applications on graphics processors
3955 using CUDA,” *Journal of Parallel and Distributed Computing*, vol. 68,
3956 no. 10, pp. 1370 – 1380, 2008, k-means implementation on CUDA
3957 with 72x speedup. Compares to 4-threaded CPU version with 30x
3958 speedup. [Online]. Available: <http://www.sciencedirect.com/science/article/B6WKJ-4SVV8GS-2/2/f7a1dccceb63cbbfd25774c6628d8412>
- 3959
- 3960 [162] Z. Liu and W. Ma, “Exploiting computing power on graphics processing unit,”
3961 vol. 2, Dec. 2008, pp. 1062–1065.
- 3962 [163] S. Hong and H. Kim, “An integrated GPU power and performance model,”
3963 in *ACM SIGARCH Computer Architecture News*, vol. 38. ACM, 2010, pp.
3964 280–289.
- 3965 [164] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau,
3966 P. Franzon, W. Harrod, K. Hill *et al.*, “Exascale computing study: Tech-
3967 nology challenges in achieving exascale systems,” 2008.
- 3968 [165] S. Crago, K. Dunn, P. Eads, L. Hochstein, D.-I. Kang, M. Kang, D. Mod-
3969 ium, K. Singh, J. Suh, and J. P. Walters, “Heterogeneous cloud computing,” in
3970 *Proceedings of the Workshop on Parallel Programming on Accelerator Clusters*
3971 (*PPAC*). *Cluster Computing (CLUSTER), 2011 IEEE International Confer-
3972 ence on*. IEEE, 2011, pp. 378–385.

- 3973 [166] J. Sanders and E. Kandrot, *CUDA by example: an introduction to general-*
3974 *purpose GPU programming.* Addison-Wesley Professional, 2010.
- 3975 [167] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E.
3976 Stone, J. C. Phillips, and W.-m. Hwu, “GPU clusters for high-performance
3977 computing,” in *Cluster Computing and Workshops, 2009. CLUSTER’09. IEEE*
3978 *International Conference on.* IEEE, 2009, pp. 1–8.
- 3979 [168] K. J. Barker, K. Davis, A. Hoisie, D. J. Kerbyson, M. Lang, S. Pakin, and J. C.
3980 Sancho, “Entering the petaflop era: the architecture and performance of Road-
3981 runner,” in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing.*
3982 IEEE Press, 2008, p. 1.
- 3983 [169] S. Craven and P. Athanas, “Examining the viability of FPGA supercomputing,”
3984 *EURASIP Journal on Embedded systems*, vol. 2007, no. 1, pp. 13–13, 2007.
- 3985 [170] G. Shainer, T. Liu, J. Layton, and J. Mora, “Scheduling strategies for HPC as a
3986 service (HPCaaS),” in *Cluster Computing and Workshops, 2009. CLUSTER’09.*
3987 *IEEE International Conference on.* IEEE, 2009, pp. 1–6.
- 3988 [171] A. J. Younge, R. Henschel, J. T. Brown, G. von Laszewski, J. Qiu, and G. C.
3989 Fox, “Analysis of Virtualization Technologies for High Performance Computing
3990 Environments,” in *Proceedings of the 4th International Conference on Cloud*
3991 *Computing (CLOUD 2011).* Washington, DC: IEEE, July 2011.
- 3992 [172] W. Wade, “How NVIDIA and Citrix are driving the future of virtualized visual
3993 computing,” [Online], <http://blogs.nvidia.com/blog/2013/05/22/synergy/>.
- 3994 [173] S. Long, “Virtual machine graphics acceleration deployment guide,” VMWare,
3995 Tech. Rep., 2013.

- 3996 [174] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and
3997 J. T. Klosowski, “Chromium: a stream-processing framework for interactive
3998 rendering on clusters,” in *ACM Transactions on Graphics (TOG)*, vol. 21.
3999 ACM, 2002, pp. 693–702.
- 4000 [175] G. Giunta, R. Montella, G. Agrillo, and G. Coviello, “A GPGPU
4001 transparent virtualization component for high performance computing clouds,”
4002 in *Euro-Par 2010 - Parallel Processing*, ser. Lecture Notes in Computer
4003 Science, P. DAmbra, M. Guarracino, and D. Talia, Eds. Springer
4004 Berlin Heidelberg, 2010, vol. 6271, pp. 379–391. [Online]. Available:
4005 http://dx.doi.org/10.1007/978-3-642-15277-1_37
- 4006 [176] K. Diab, M. Rafique, and M. Hefeeda, “Dynamic sharing of GPUs in cloud
4007 systems,” in *Parallel and Distributed Processing Symposium Workshops PhD
4008 Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 947–954.
- 4009 [177] C.-T. Yang, H.-Y. Wang, and Y.-T. Liu, “Using pci pass-through for gpu vir-
4010 tualization with cuda,” in *Network and Parallel Computing*. Springer, 2012,
4011 pp. 445–452.
- 4012 [178] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford,
4013 V. Tipparaju, and J. S. Vetter, “The scalable heterogeneous computing (SHOC)
4014 benchmark suite,” in *Proceedings of the 3rd Workshop on General-Purpose
4015 Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.
- 4016 [179] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen, “Direct numerical
4017 simulation of turbulent combustion: fundamental insights towards predictive
4018 models,” in *Journal of Physics: Conference Series*, vol. 16. IOP Publishing,
4019 2005, p. 65.

- 4020 [180] F. Silla, “rCUDA: share and aggregate GPUs in your cluster,” Nov. 2012, mel-
4021 lanox Booth Presaentation.
- 4022 [181] H. Jo, J. Jeong, M. Lee, and D. H. Choi, “Exploiting GPUs in virtual machine
4023 for biocloud,” *BioMed research international*, vol. 2013, 2013.
- 4024 [182] J. Duato, A. Pena, F. Silla, R. Mayo, and E. Quintana-Orti, “rCUDA: Re-
4025 ducing the number of gpu-based accelerators in high performance clusters,”
4026 in *High Performance Computing and Simulation (HPCS), 2010 International
4027 Conference on*, June 2010, pp. 224–231.
- 4028 [183] B. L. Jacob and T. N. Mudge, “A look at several memory management units,
4029 TLB-refill mechanisms, and page table organizations,” in *Proceedings of the
4030 Eighth International Conference on Architectural Support for Programming
4031 Languages and Operating Systems*. ACM, 1998, pp. 295–306.
- 4032 [184] B.-A. Yassour, M. Ben-Yehuda, and O. Wasserman, “Direct device assignment
4033 for untrusted fully-virtualized virtual machines,” IBM Research, Tech. Rep.,
4034 2008.
- 4035 [185] M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, and
4036 L. Van Doorn, “The price of safety: Evaluating IOMMU performance,” in
4037 *Ottawa Linux Symposium*, 2007.
- 4038 [186] J. Liu, “Evaluating standard-based self-virtualizing devices: A performance
4039 study on 10 GbE NICs with SR-IOV support,” in *Parallel Distributed Processing
4040 (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1–12.
- 4041 [187] V. Jujjuri, E. Van Hensbergen, A. Liguori, and B. Pulavarty, “VirtFS-a virtu-
4042 alization aware file system passthrough,” in *Ottawa Linux Symposium*, 2010.

- 4043 [188] C. Yang, H. Wang, W. Ou, Y. Liu, and C. Hsu, “On implementation of GPU
4044 virtualization using PCI pass-through,” in *4th IEEE International Conference*
4045 on *Cloud Computing Technology and Science Proceedings (CloudCom)*, 2012.
- 4046 [189] M. Dowty and J. Sugerman, “GPU virtualization on VMware’s hosted I/O
4047 architecture,” *ACM SIGOPS Operating Systems Review*, vol. 43, no. 3, pp. 73
4048 – 82, 2009.
- 4049 [190] “FutureGrid,” Web page. [Online]. Available: <http://portal.futuregrid.org>
- 4050 [191] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spaf-
4051 ford, V. Tipparaju, and J. S. Vetter, “The Scalable Heterogeneous Computing
4052 (SHOC) benchmark suite,” in *Proceedings of the 3rd Workshop on General-*
4053 *Purpose Computation on Graphics Processing Units*, ser. GPGPU ’10. ACM,
4054 2010, pp. 63–74.
- 4055 [192] “LAMMPS molecular dynamics simulator,” <http://lammps.sandia.gov/>, [On-
4056 line; accessed Jan. 2, 2014].
- 4057 [193] A. Athanasopoulos, A. Dimou, V. Mezaris, and I. Kompatsiaris, “GPU ac-
4058 leartion for support vector machines,” in *Proc 12th International Workshop*
4059 on *Image Analysis for Multimedia Interactive Services (WIAMIS 2001)*, April
4060 2011.
- 4061 [194] “LULESH shock hydrodynamics application,” <https://codesign.llnl.gov/lulesh.php>, [Online; accessed Jan. 2, 2014].
- 4063 [195] S. Plimpton, “Fast parallel algorithms for short-range molecular dynamics,”
4064 *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995.

- 4065 [196] W. M. Brown, “GPU acceleration in LAMMPS,” <http://lammps.sandia.gov/workshops/Aug11/Brown/brown11.pdf>, [Online; accessed Jan. 2, 2014].
- 4066
- 4067 [197] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,”
4068 *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, pp. 27:1–27:27, May 2011.
- 4069 [198] “Hydrodynamics Challenge Problem,” <https://codesign.llnl.gov/pdfs/spec-7.pdf>, [Online; accessed Jan. 2, 2014].
- 4070
- 4071 [199] A. Arcangeli, “Transparent hugepage support,” <http://www.linux-kvm.org/wiki/images/9/9e/2010-forum-thp.pdf>, 2010, [Online; accessed Jan. 29, 2014].
- 4072
- 4073 [200] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, “SR-IOV
4074 support for virtualization on infiniband clusters: Early experience,” in *Cluster
4075 Computing and the Grid, IEEE International Symposium on*. IEEE Computer
4076 Society, 2013, pp. 385–392.
- 4077 [201] R. L. Moore, C. Baru, D. Baxter, G. C. Fox, A. Majumdar, P. Papadopoulos,
4078 W. Pfeiffer, R. S. Sinkovits, S. Strande, M. Tatineni *et al.*, “Gateways to
4079 discovery: Cyberinfrastructure for the long tail of science,” in *Proceedings of
4080 the 2014 Annual Conference on Extreme Science and Engineering Discovery
4081 Environment*. ACM, 2014, p. 39.
- 4082 [202] P. Luszczek, E. Meek, S. Moore, D. Terpstra, V. M. Weaver, and J. Dongarra,
4083 “Evaluation of the hpc challenge benchmarks in virtualized environments,” in
4084 *Proceedings of the 2011 International Conference on Parallel Processing - Volume 2*, ser. Euro-Par’11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 436–
4085 445.
- 4086

- 4087 [203] N. Huber, M. von Quast, M. Hauck, and S. Kounev, “Evaluating and model-
4088 ing virtualization performance overhead for cloud environments.” in *CLOSER*,
4089 2011, pp. 563–573.
- 4090 [204] J. P. Walters, A. J. Younge, D.-I. Kang, K.-T. Yao, M. Kang, S. P. Crago,
4091 and G. C. Fox, “GPU-Passthrough Performance: A Comparison of KVM, Xen,
4092 VMWare ESXi, and LXC for CUDA and OpenCL Applications,” in *Proceedings
4093 of the 7th IEEE International Conference on Cloud Computing (CLOUD 2014)*.
4094 Anchorage, AK: IEEE, 2014.
- 4095 [205] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, “Sr-iov
4096 support for virtualization on infiniband clusters: Early experience,” in *Cluster,
4097 Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International
4098 Symposium on*. IEEE, 2013, pp. 385–392.
- 4099 [206] M. Musleh, V. Pai, J. P. Walters, A. J. Younge, and S. P. Crago, “Bridging
4100 the Virtualization Performance Gap for HPC using SR-IOV for InfiniBand,”
4101 in *Proceedings of the 7th IEEE International Conference on Cloud Computing
4102 (CLOUD 2014)*. Anchorage, AK: IEEE, 2014.
- 4103 [207] “Aws high performance computing,” <http://aws.amazon.com/hpc/>, last Access
4104 Nov. 2014.
- 4105 [208] “Openstack flavors,” [http://docs.openstack.org/openstack-ops/content/
4106 flavors.html](http://docs.openstack.org/openstack-ops/content/flavors.html), last Access Nov. 2014.
- 4107 [209] K. Pepple, *Deploying OpenStack*. O’Reilly Media, Inc., 2011.
- 4108 [210] S. Hazelhurst, “Scientific computing using virtual high-performance computing:
4109 a case study using the amazon elastic computing cloud,” in *Proceedings of the*

- 4110 2008 annual research conference of the South African Institute of Computer
4111 Scientists and Information Technologists on IT research in developing countries:
4112 riding the wave of technology. ACM, 2008, pp. 94–103.
- 4113 [211] E. Amazon, “Amazon elastic compute cloud (amazon ec2),” *Amazon Elastic*
4114 *Compute Cloud (Amazon EC2)*, 2010.
- 4115 [212] R. Jennings, *Cloud Computing with the Windows Azure Platform*. John Wiley
4116 & Sons, 2010.
- 4117 [213] “Google cloud platform,” <https://cloud.google.com/>, last Access Nov. 2014.
- 4118 [214] L. Ramakrishnan, R. S. Canon, K. Muriki, I. Sakrejda, and N. J. Wright,
4119 “Evaluating interconnect and virtualization performance forhigh performance
4120 computing,” *SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 2, pp. 55–60,
4121 Oct. 2012. [Online]. Available: <http://doi.acm.org/10.1145/2381056.2381071>
- 4122 [215] M. Righini, “Enabling intel virtualization technology features and benefits,”
4123 Intel Corporation, Tech. Rep., 2010.
- 4124 [216] AMD, “AMD i/o virtualization technology (IOMMU) specification,” AMD Cor-
4125 poration, Tech. Rep., 2009.
- 4126 [217] A. Limited, “Arm system memory management unit architecture specification,”
4127 ARM Limited, Tech. Rep., 2013.
- 4128 [218] A. J. Younge, J. P. Walters, S. Crago, and G. C. Fox, “Evaluating GPU
4129 Passthrough in Xen for High Performance Cloud Computing,” in *High-*
4130 *Performance Grid and Cloud Computing Workshop at the 28th IEEE Inter-*
4131 *national Parallel and Distributed Processing Symposium*, IEEE. Pheonix, AZ:
4132 IEEE, 05/2014 2014.

- 4133 [219] L. Vu, H. Sivaraman, and R. Bidarkar, “Gpu virtualization for high
4134 performance general purpose computing on the esx hypervisor,” in *Proceedings
4135 of the High Performance Computing Symposium*, ser. HPC ’14. San Diego,
4136 CA, USA: Society for Computer Simulation International, 2014, pp. 2:1–2:8.
4137 [Online]. Available: <http://dl.acm.org/citation.cfm?id=2663510.2663512>
- 4138 [220] J. Liu, “Evaluating standard-based self-virtualizing devices: A performance
4139 study on 10 gbe nics with sr-iov support,” in *Parallel Distributed Processing
4140 (IPDPS), 2010 IEEE International Symposium on*, April 2010, pp. 1–12.
- 4141 [221] T. P. P. D. L. Ruivo, G. B. Altayo, G. Garzoglio, S. Timm, H. Kim, S.-Y.
4142 Noh, and I. Raicu, “Exploring infiniband hardware virtualization in opennebula
4143 towards efficient high-performance computing.” in *CCGRID*, 2014, pp. 943–948.
- 4144 [222] “NVIDIA GPUDirect,” <https://developer.nvidia.com/gpudirect>, [Online; ac-
4145 cessed Nov. 24, 2014].
- 4146 [223] G. Shainer, A. Ayoub, P. Lui, T. Liu, M. Kagan, C. R. Trott, G. Scantlen, and
4147 P. S. Crozier, “The development of mellanox/nvidia gpudirect over infinibanda
4148 new model for gpu to gpu communications,” *Computer Science-Research and
4149 Development*, vol. 26, no. 3-4, pp. 267–273, 2011.
- 4150 [224] “Getting Xen working for Intel(R) Xeon Phi(tm)
4151 Coprocessor,” <https://software.intel.com/en-us/articles/getting-xen-working-for-intelr-xeon-phitm-coprocessor>, [Online; ac-
4152 cessed Nov. 24, 2014].
- 4153 [225] “Mellanox Neutron Plugin,” <https://wiki.openstack.org/wiki/Mellanox-Neutron>, [Online; accessed Nov. 24, 2014].

- 4156 [226] S. Plimpton, P. Crozier, and A. Thompson, “Lammps-large-scale
4157 atomic/molecular massively parallel simulator,” *Sandia National Laboratories*, 2007.
4158
- 4159 [227] J. Anderson, A. Keys, C. Phillips, T. Dac Nguyen, and S. Glotzer, “Hoomd-
4160 blue, general-purpose many-body dynamics on the gpu,” in *APS Meeting Ab-*
4161 *stracts*, vol. 1, 2010, p. 18008.
- 4162 [228] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer,
4163 D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams *et al.*, “The land-
4164 scape of parallel computing research: A view from berkeley,” Technical Report
4165 UCB/EECS-2006-183, EECS Department, University of California, Berkeley,
4166 Tech. Rep., 2006.
- 4167 [229] G. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo,
4168 S. Smallen, W. Smith, and A. Grimshaw, “Futuregrida reconfigurable testbed
4169 for cloud, hpc and grid computing,” *Contemporary High Performance Comput-*
4170 *ing: From Petascale toward Exascale, Computational Science. Chapman and*
4171 *Hall/CRC*, 2013.
- 4172 [230] K. Keahey, J. Mambretti, D. K. Panda, P. Rad, W. Smith, and
4173 D. Stanzione, “Nsf chameleon cloud,” website, November 2014. [Online].
4174 Available: <http://www.chameleoncloud.org/>
- 4175 [231] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger, G. Regnier,
4176 R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert, “Intel virtu-
4177 alization technology for directed i/o.” *Intel technology journal*, vol. 10, no. 3,
4178 2006.

- 4179 [232] F. Silla, “rcuda: towards energy-efficiency in gpu computing by leveraging low-
4180 power processors and infiniband interconnects,” in *HPC Advisory Council Spain
4181 Conference*, 2013.
- 4182 [233] J. P. Walters, “Achieving near-native gpu performance in the cloud,” in *GPU
4183 Technology Conference*, 2015.
- 4184 [234] Y. Luo, “Network i/o virtualization for cloud computing,” *IT Professional Mag-
4185 azine*, vol. 12, no. 5, p. 36, 2010.
- 4186 [235] P. Kutch, “Pci-sig sr-iov primer: An introduction to sr-iov technology,” *Intel
4187 application note*, pp. 321 211–002, 2011.
- 4188 [236] M. Rosenblum and T. Garfinkel, “Virtual machine monitors: Current technol-
4189 ogy and future trends,” *Computer*, vol. 38, no. 5, pp. 39–47, 2005.
- 4190 [237] R. Bhargava, B. Serebrin, F. Spadini, and S. Manne, “Accelerating two-
4191 dimensional page walks for virtualized systems,” in *ACM SIGARCH Computer
4192 Architecture News*, vol. 36, no. 1. ACM, 2008, pp. 26–35.
- 4193 [238] B. Pham, J. Vesely, G. H. Loh, and A. Bhattacharjee, “Using tlb speculation to
4194 overcome page splintering in virtual machines,” Rutgers University Technical
4195 Report DCS-TR-713, Tech. Rep., 2015.
- 4196 [239] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and
4197 A. Warfield, “Live migration of virtual machines,” in *Proceedings of the 2nd
4198 conference on Symposium on Networked Systems Design & Implementa-
4199 tion-Volume 2*. USENIX Association, 2005, pp. 273–286.
- 4200 [240] M. R. Hines and K. Gopalan, “Post-copy based live virtual machine migration
4201 using adaptive pre-paging and dynamic self-balloonning,” in *Proceedings of the*

- 4202 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution
4203 environments. ACM, 2009, pp. 51–60.
- 4204 [241] P. Lu, A. Barbalace, and B. Ravindran, “Hsg-lm: Hybrid-copy speculative
4205 guest os live migration without hypervisor,” in *Proceedings of the*
4206 *6th International Systems and Storage Conference*, ser. SYSTOR ’13.
4207 New York, NY, USA: ACM, 2013, pp. 2:1–2:11. [Online]. Available:
4208 <http://doi.acm.org/10.1145/2485732.2485736>
- 4209 [242] J. Chu and V. Kashyap, “Transmission of IP over InfiniBand (IPoIB),” IETF,
4210 Tech. Rep., 2006.
- 4211 [243] W. Yu, N. S. Rao, P. Wyckoff, and J. S. Vetter, “Performance of rdma-capable
4212 storage protocols on wide-area network,” in *2008 3rd Petascale Data Storage*
4213 *Workshop*. IEEE, 2008, pp. 1–5.
- 4214 [244] W. Huang, Q. Gao, J. Liu, and D. K. Panda, “High performance virtual machine
4215 migration with rdma over modern interconnects,” in *2007 IEEE International*
4216 *Conference on Cluster Computing*. IEEE, 2007, pp. 11–20.
- 4217 [245] D. Gilbert, “Post copy live migration,” Webpage, 2015. [Online]. Available:
4218 <http://wiki.qemu.org/Features/PostCopyLiveMigration>
- 4219 [246] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer,
4220 K. D. Underwood, and R. C. Zak, “Intel omni-path architecture: Enabling
4221 scalable, high performance fabrics,” in *2015 IEEE 23rd Annual Symposium on*
4222 *High-Performance Interconnects*, Aug 2015, pp. 1–9.
- 4223 [247] M. Beck and M. Kagan, “Performance evaluation of the rdma over ethernet
4224 (roce) standard in enterprise data centers infrastructure,” in *Proceedings of*

- 4225 *the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching.*
4226 International Teletraffic Congress, 2011, pp. 9–15.
- 4227 [248] E. Kissel and M. Swany, “Photon: Remote memory access middleware for high-
4228 performance runtime systems,” in *2016 IEEE International Parallel and Dis-*
4229 *tributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 1736–
4230 1743.
- 4231 [249] J. Lofstead, I. Jimenez, and C. Maltzahn, “Consistency and fault tolerance con-
4232 siderations for the next iteration of the doe fast forward storage and io project,”
4233 in *2014 43rd International Conference on Parallel Processing Workshops*, Sept
4234 2014, pp. 61–69.
- 4235 [250] C. G. Wright Jr, “Trinity burst buffer-architecture and design,” Los Alamos
4236 National Laboratory (LANL), Tech. Rep., 2015.
- 4237 [251] F. B. Schmuck and R. L. Haskin, “Gpfss: A shared-disk file system for large
4238 computing clusters.” in *FAST*, vol. 2, 2002, pp. 231–244.
- 4239 [252] M. J. Rashti and A. Afsahi, “10-gigabit iwarps ethernet: comparative perfor-
4240 mance analysis with infiniband and myrinet-10g,” in *2007 IEEE International*
4241 *Parallel and Distributed Processing Symposium*. IEEE, 2007, pp. 1–8.
- 4242 [253] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, “Image processing on a
4243 simd/spmd architecture: Opsila,” in *Pattern Recognition, 1988., 9th Interna-*
4244 *tional Conference on*, Nov 1988, pp. 430–433 vol.1.
- 4245 [254] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble,
4246 E. De Lara, M. Brudno, and M. Satyanarayanan, “Snowflock: rapid virtual
4247 machine cloning for cloud computing,” in *Proceedings of the 4th ACM European*
4248 *conference on Computer systems*. ACM, 2009, pp. 1–12.

- 4249 [255] H. A. Lagar-Cavilla, J. A. Whitney, R. Bryant, P. Patchin, M. Brudno,
4250 E. de Lara, S. M. Rumble, M. Satyanarayanan, and A. Scannell, “Snowflock:
4251 Virtual machine cloning as a first-class cloud primitive,” *ACM Transactions on
4252 Computer Systems (TOCS)*, vol. 29, no. 1, p. 2, 2011.
- 4253 [256] A. J. Younge, G. von Laszewski, L. Wang, and G. C. Fox, “Providing a Green
4254 Framework for Cloud Based Data Centers,” in *The Handbook of Energy-Aware
4255 Green Computing*, I. Ahmad and S. Ranka, Eds. Chapman and Hall/CRC
4256 Press, 2011, ch. 17, in press.
- 4257 [257] D. P. Berrange, “Openstack performance optimization,” in *KVM Forum 2014*,
4258 2014.
- 4259 [258] Y. Jiang and Y. He, “Openstack pci passthrough,” Webpage, Intel, Inc, 2016.
4260 [Online]. Available: https://wiki.openstack.org/wiki/Pci_passthrough
- 4261 [259] A. Hanemann, J. W. Boote, E. L. Boyd, J. Durand, L. Kudarimoti, R. Łapacz,
4262 D. M. Swany, S. Trocha, and J. Zurawski, “Perfsonar: A service oriented archi-
4263 tecture for multi-domain network monitoring,” in *International Conference on
4264 Service-Oriented Computing*. Springer, 2005, pp. 241–254.
- 4265 [260] D. Serrano, S. Bouchenak, Y. Kouki, T. Ledoux, J. Lejeune, J. Sopena,
4266 L. Arantes, and P. Sens, “Towards qos-oriented sla guarantees for online
4267 cloud services,” in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th
4268 IEEE/ACM International Symposium on*. IEEE, 2013, pp. 50–57.