High Performance Computing Cloud - a Platform-as-a-Service Perspective

A PROJECT REPORT
SUBMITTED IN PARTIAL FULFIMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
Master of Technology
IN
Computational Science

BY

Pratima Dhuldhule

Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2015

DEDICATED TO

My Family

Acknowledgements

I wish to thank my guides, Dr. J. Lakshmi and Prof. S. K. Nandy, who have been a source of guidance throughout the course of this project. Without their support this work would not have been possible.

I wish to thank Prof. Govindarajan, SERC chairman and all the professors from SERC and CSA, for their support and great courses that they offered. It was a great learning experience sitting through each and every one of the lectures.

It was also a great pleasure knowing my peers at IISc. I wish to thank my seniors, Aakriti, Nitisha and Mohit for their endless support and encouragement.

I wish to thank my labmates, Ishwar and Vaibhav for all the brainstorming sessions in the labs. It was great experiance learning with them.

I also wish to express my deepest appriciation for my friends - David, Sayani and Karishma for always being there and constant encouragement.

Finally, I wish to express my special gratitute to my parents and my dear sister for their constant love and understanding.

Abstract

HPC applications are widely used in academics, research laboratories and in industries for business and analytic. Most HPC applications require dedicated, available and highly customized resources and environments for computation since they exhibit intense resource utilization. These needs were traditionally provided by clusters and supercomputers which are difficult to setup, manage or operate. While majority of the HPC installations ensure good resource utilization, the reach of these is restricted to few who are members of a specific HPC community. Cloud computing is emerging as a latest computing technology. The on-demand nature of cloud has provoked interest to explore if cloud properties can be useful for HPC setups. This report is a work in that direction. The prevalent public clouds have accessibility to many and have been explored by the HPC community too. The biggest deterrent identified on these computing platforms for HPC workloads is the virtualization layer used by the cloud systems for resource provisioning.

In this report we propose a Platform-as-a-Service model to build an HPC cloud setup. The key goals for the architecture design is to include features like on-demand provisioning both for hardware as well as HPC runtime environment for the cloud user and at the same time ensure that the HPC applications do not suffer virtualization overheads. The architecture builds the required HPC platform by providing dedicated node or a group of nodes booted with the desired HPC environment without the virtualization layer. Technologies like Wake-on-LAN and network booting are used to achieve this goal. Once the usage of these resources is relinquished, the same nodes are re-deployed for another HPC platform. Thus this architecture merges cloud properties with HPC platforms for delivering effective performance. We show the results of benchmarks used to evaluate performance difference between a virtualized and non-virtualized environment for this observation.

Contents

Acknowledgements i

Abstract ii

Contents iii

List of Figures v

1 High Performance Computing Cloud 1

1.1 Introduction . . . 1
1.2 Related Work . . . 3
1.3 Motivation . . . 4
1.4 Summary . . . 5

2 HPC Cloud Architecture 6

2.1 Overall System Architecture . . . 6
2.1.1 Platform Manager . . . 7
2.1.2 Admission Control Unit . . . 8
2.1.3 Resource Information Manager . . . 9
2.1.4 Environment Setup Unit . . . 9
2.1.5 Monitor . . . 9
2.2 Working . . . 10
2.3 Failure Handling . . . 10
2.4 Summary . . . 11

3 Case Study 12

3.1 Experimental Setup . . . 12
3.2 Benchmarks . . . 13

3.2.1 NBench Benchmark . . . 13
3.2.2 Stream Benchmark . . . 14
3.2.3 IOR Benchmark . . . 14
3.2.4 Iperf Benchmark . . . 15
3.2.5 OSU micro Benchmark . . . 15
3.3 Results for Benchmarks . . . 16
3.3.1 CPU Performance . . . 16
3.3.2 Memory Performance . . . 16
3.3.3 Disk Performance . . . 17
3.3.4 Network Performance . . . 18
3.4 Summary . . . 21

4 Conclusion and Future Work **24**

Bibliography **25**

List of Figures

2.1 HPC Cloud Architecture 7
2.2 Admission Control Unit 8

3.1 NBench : CPU performance 16
3.2 Stream : Memory Bandwidth 17
3.3 Stream : Execution Time for Memory Operations 18
3.4 IOR : Disk Bandwidth 18
3.5 Iperf : Bandwidth Measurement for UDP 19
3.6 Iperf : Jitter during UDP Transmission 20
3.7 Iperf : Packet Loss during UDP Transmission 21
3.8 Iperf : Bandwidth Measurement for TCP 22
3.9 OSU micro benchmarks : Bandwidth for point-to-point communication 22
3.10 OSU micro benchmarks : Bi-directional bandwidth for point-to-point communication 23
3.11 OSU micro benchmarks : Latency for point-to-point communication 23

Chapter 1

High Performance Computing Cloud

High Performance Computing Cloud is gaining popularity among the HPC community. Properties of cloud suitable for HPC workload makes it a viable solution. In this chapter we discuss about the properties of HPC workload and cloud environment. We also discuss about the disadvantages of virtualization for HPC applications. The work done in this field and the motivation behind our chosen approach is discussed next.

1.1 Introduction

Cloud computing is growing as a popular computing technology. Cloud is being extensively used for enterprise or business tasks because there is no need to own an infrastructure, handle the services and maintain it. It provides on-demand resources and services on a metered basis. This is cost efficient as we pay for whatever we are using and once a task is done we give away the resources and stop paying for it. Also the availability, scalability, flexibility are some of the attractive features of cloud computing. Its availability and on-demand nature are some of the major reasons for its growth. The main idea behind cloud computing is enabling a virtual machine to run on any server. Since there are many customers and many servers, the management of the infrastructure must be highly automated. A customer can request the creation or removal of a virtual machine and without human intervention a virtual machine is started or stopped on one of the servers. To take advantage of the economic benefits, the cloud providers use multi-tenancy, where virtual machines from multiple customers share a server. This catering to different users at a time is possible due the hypervisors present in virtualization layer. At a physical level, the resources are shared between different users and applications. It caters to different variety of applications such as enterprise workload, social networking, data storage and research computations. The needs of these applications are also different. Resource

requirements are manly in the area of networking, disk IO, CPU etc.

Due to all the benefits of cloud, it seems an attractive option for HPC workloads. Feature like flexibility plays an important role as it is not seen in dedicated machines. If on a dedicated machine, different environment is required, a new cluster has to be set up with the required environment which can take upto several days. On the other hand cloud provides variety of environment on-demand easily. The property of elasticity in the cloud is also useful for HPC applications as an HPC application may require scaling in between the computation. Availability is also one of the property useful for HPC workloads.

Inspite of all the above advantages of cloud for HPC applications, there are some serious issues related with it. HPC applications are used in academia and laboratories for research, hence need guarantee of resources and timely results. They tend to consume more than 90% of the available CPU cycles, require low latency and high bandwidth inter-process communication, which is difficult to achieve in a virtualized environment due to I/O overhead and delay in CPU cycles [8]. Previous work shows that Cloud is one or two magnitude worse as compared to Infiniband, which is currently used in supercomputers for interconnect network [4] [12]. The virtualization of interconnect becomes a bottleneck for HPC applications and increases latency of communication intense applications. Thus the use of virtualization layer proves to be a major hindrance in performance of HPC applications [1].

The notion of virtualization was brought in mainly to utilize all the existing resources, as virtual machines share the overall resources instead of having equal share of it. But for HPC applications who are capable of utilizing major portion of the available resources do not have the issue of under-utilization. Past results [4] [1] [5] on evaluation of HPC applications on Cloud using IaaS platform with virtualization have been pessimistic. [4] tells that not all types of HPC applications are suitable for Cloud. The HPC applications with less intensive communication patterns, having less sensitivity towards interference and applications with performance needs that can be met at small to medium scale execution (in terms of number of cores) are suitable for Cloud under virtualized environment [5] [6].

This paper discusses about the idea of converging the benefits of Cloud like elasticity and flexibility with HPC platforms to create an environment for HPC workload. PaaS properties are explored for achieving this goal as PaaS has the ability to provide on-demand platform, abstracts and controls the underlying resources and gives choice of platform and guarantee of performance. Instead of improving the virtualization layer we propose a model where virtialization layer is completely removed. By elimination of virtualization layer and with PaaS model a framework is obtained for provisioning of dedicated platforms. On-demand environment is setup by booting a node with desired operating system and runtime libraries for building user-specific HPC

platform. The key contributions of this work are as follows:

- Provision of a non-virtualized platform to obtain contention free and dedicated use of resources.
- Design of a PaaS model to provide on-demand and isolated environment to meet HPC application requirements.

1.2 Related Work

Significant work has been done for achieving high performance in cloud. The merging of the two worlds is not new and has been studied from long time. Scientists have marched towards deploying their HPC workloads on cloud. So far, provision of dedicated resources for HPC applications is largely unexplored on the cloud setup.

There has been work done towards using latest technology for improving performance in various aspects in the cloud. A design of heterogeneous high performance IaaS using OpenStack to address issues regarding virtualization performance, heterogeneous hardware and GPUs, high speed interconnects, advanced scheduling and high performance storage is discussed in [14]. Several science clouds have been establish for scientific computations [11]. But no considerable work is done towards completely removing the virtualization layer to avoid the overhead. In [7], strategies have been discussed to remove the hypervisor while still keeping the notion of VMs. But there are architectural as well as implementation issues which are not properly addressed. [3] discusses about environment aware scheduling of HPC applications on cloud setup to improve performance.

It is been observed that the performance obtained on cloud is much below than the dedicated machines. The major performance hindrance is virtualization. Also security issues are also attached for the scientists while deploying their HPC workload on cloud [13]. Amazon has come up with a HPC cloud with special instances provided for compute intense applications. Different instance types and sizes are provided for deploying HPC workload at low cost. NASA has conducted a performance evaluation test of Amazon EC2 vs Pleiades, a supercomputer in NASA. There performance is tested using various benchmarks, and considerable drop in performance is shown in [9]. NASA has said that Amazon EC2 are not yet ready for HPC workloads but the technology sure is catching up fast. Similar tests have been carried out to evaluate the performance of HPC applications on current Cloud setups in [5] [4] and [1].

[12] discusses about the virtualization impact on the network performance. The unstable network characteristics are caused by virtualization and processor sharing on server hosts. The

unstable network performance degrades the performance of and bring new challenges to many applications.

[10] discusses about virtualization impact on HPC applications for IO and makes an effort in improving IO performance with the help of SR-IOV. It concludes that virtualized performance for HPC workloads is competitive with that of native if network-interrupt parameters are tuned to increase the responsiveness of the SR-IOV system.

In [2], linux containers and VMs are discussed. This study compares the performance of linux containers and virtual machines. Performance over different physical resources is measured and it concludes that the performance of linux containers is equal or better than VMs in most cases.

The studies carried out till now talk about improvement of HPC application on the existing architecture of Cloud. But the methodology presented in this report gives an architecture which addresses all the challenges of deploying HPC on Cloud. The architecture completely removes the virtualization layer and provides dedicated resources in terms of CPU cores, memory bandwidth, network bandwidth and storage.

1.3 Motivation

High Performance Computing applications consist of concurrent programs designed using multi-threaded and multi-process paradigm. The application consists of various parallel constructs like threads, local processes, distributed processes etc. with varying degree of parallelism. Traditionally, Clouds have been designed for running business and web applications, whose resource requirements are different from HPC applications. Unlike web applications, HPC applications typically require low latency and high bandwidth inter-processor communication to achieve best performance. In case of cloud, effect of virtualization result in interconnect becoming a bottleneck for HPC applications. Studies show that network performance of Cloud is one to two orders of magnitude worse as compared to Infiniband, which is commonly used as interconnect in Supercomputers. HPC applications tend to consume about 90% of the available resources. They require accurate and timely results. Furthermore, Supercomputers have operating systems and I/O subsystems specifically tailored to match HPC application demands.

Cloud is built on the notion of shared resources and virtualization. Due to this, applications running on cloud experience CPU jitter or delay in CPU cycles, I/O overhead and high latency. Also as the workload is deployed on an unknown machine there are security concerns associated with it. The use of virtualization makes the system prone to attacks. While using the cloud environment user is provided with predefined size of instances. User cannot demand for instance

size of his or her own choice if the requirement is more as instance types are limited.

HPC applications do not give expected performance on the cloud environment. Thus there is a need of a strategy which builds isolated platforms on-demand. To achive this goal Platform-as-a-Service is a viable choice. PaaS model provides facilities like scalability, manageability, elasticity etc. PaaS model has the ability to provide on-demand platform. It guarantees the quality of performance. The user only manages the application and the data associated with it. User need not worry about the underlying infrastructure details. PaaS abstracts and controls the resources and gives choice of platform. Also PaaS supports the feature of development team collaboration. This property of PaaS is useful for scientist across different geographical locations to work on a common project. Thus PaaS model has the properties benificial for building a on-demand solution for HPC applications on cloud.

1.4 Summary

In this chapter, we discussed about HPC Cloud and its significance in recent research. We also discussed the work done in this field to improve the performance of HPC applications on cloud platform. We then discussed the motivation behind coming up with a methodology for providing isolated and non- virtualized platform and how PaaS model of cloud acts beneficial for provisioning of required platform. In the next section we present the proposed architecture which merges cloud properties with HPC needs.

Chapter 2

HPC Cloud Architecture

In this chapter we present the proposed solution for HPC applications in cloud environment. As seen in the previous chapter, previous work done shows that virtualization becomes a bottleneck. This chapter describes the proposed architecture in detail. Also working and error handling is discussed in this chapter.

2.1 Overall System Architecture

There has been work done towards improving the performance of HPC applications on cloud platforms. The work done till date leans towards minimization of the virtualization overhead and building strategies for isolation on existing cloud setups. But the problem of latency caused for HPC workload remains. There is a need of such a strategy which gives performance comparable to the supercomputers and dedicated machines used for HPC workloads. In this work, we present a strategy which brings together the benefits of cloud and the necessity of HPC applications. In this section an architecture is presented which delivers the required platform to the user. The main components of the architecture, as shown in Figure 2.1, are Platform Manager (PM), Admission Control Unit (ACU), Resource Information Manager (RIM), Environment Setup Unit (ESU) and Monitor. User interacts with the system through PM which acts as front end, with the help of which user sends a request for a platform. ACU accepts the request from PM and performs the task of provisioning of requested platform to the user if resources are available. RIM is responsible for maintaining the record of available resources, frequently updating it and providing it to the ACU. The setting up of environment and platform is done by the Environment Setup Unit. After a machine is given to a user Monitor maintains the state of the machines and responds back to the ACU. We call the machine or group of machines created for the user as compute cluster. Each component of the architecture is explained in

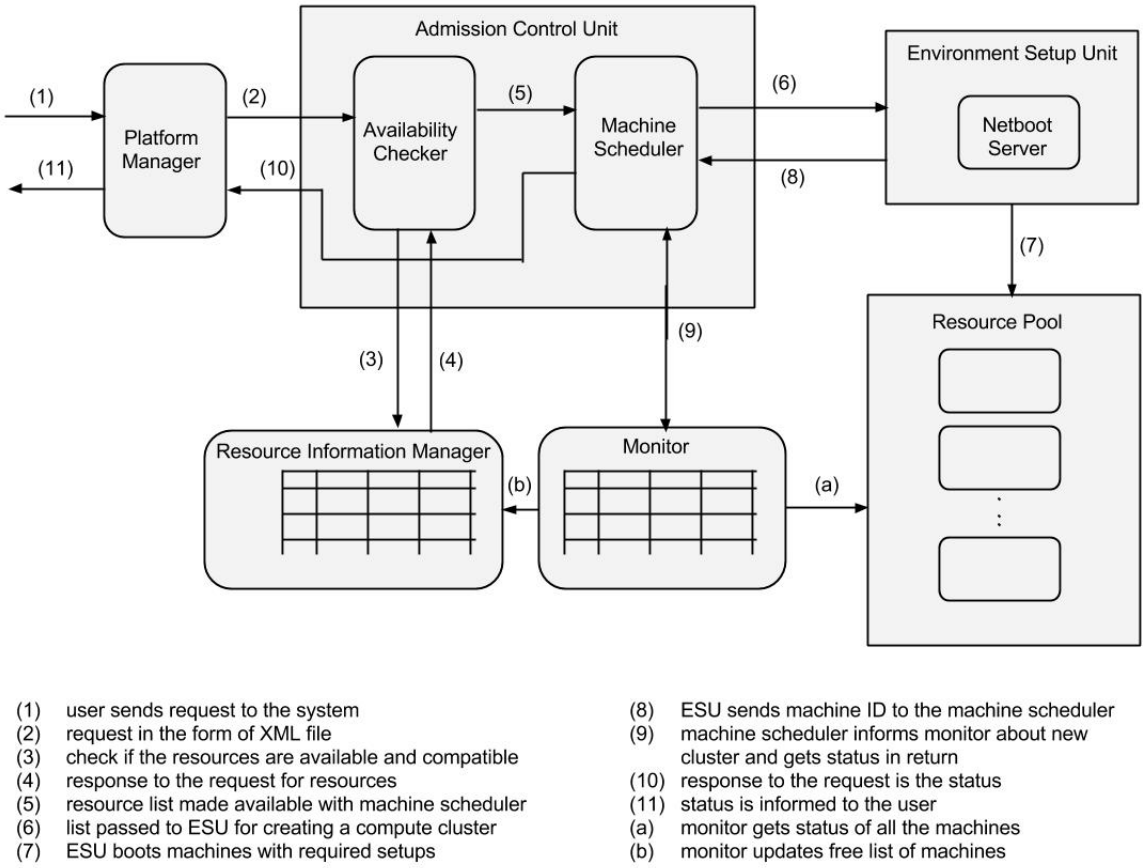

Figure 2.1: HPC Cloud Architecture

detail below.

2.1.1 Platform Manager

The PM in the architecture acts as an interface between the user and the system. It provides user the ability to create, manage and remove the machines allocated to it. Before specifying resource requirement, user needs to specify the type of request, that is, start a new compute cluster (fresh request), scale an existing compute cluster (scalability request) and stop existing compute cluster (termination request). Then the user specifies resource requirement which is divided into two parts. One is the Host requirement and other is Environment requirement. The host requirement part consist of type of processor, number of CPU cores, RAM size etc. The environment requirement part consist of type of OS, runtime environment etc. Once this information is obtained, PM forwards it to the ACU in the form of a XML file. PM is the

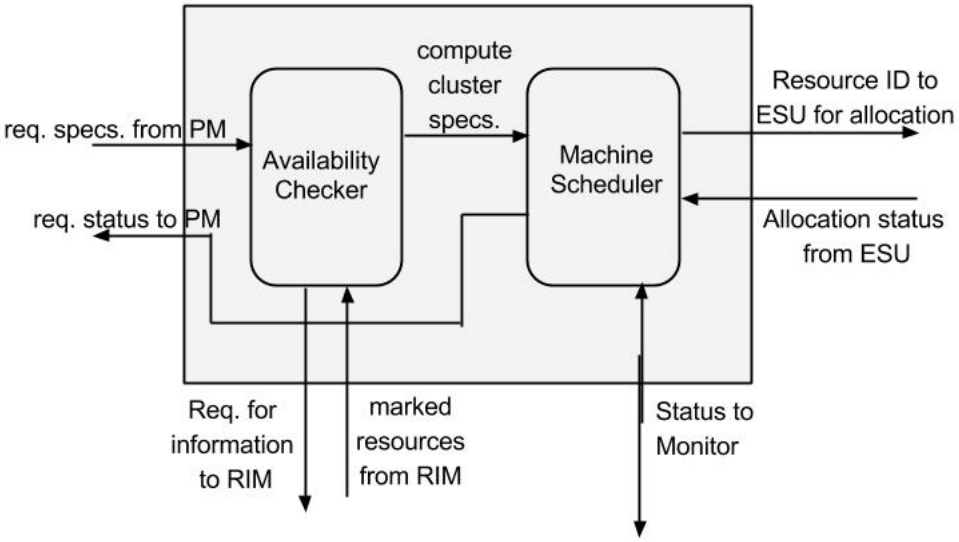

Figure 2.2: Admission Control Unit

only way to communicate with the user hence the system responds back to the user with the status of request. Also a machine ID is generated to represent the compute cluster and to ensure isolation. Any further communication related to the platform, like scaling or stopping the machine, is done through this interface.

2.1.2 Admission Control Unit

The Admission Control Unit, shown in Figure 2.2, is responsible for managing the user requests and sending instruction to other units for provisioning of a platform. As the request comes in the form of a XML file from the PM, the Availability Checker passes this information to RIM and asks to check for the required resources. If the resources are available and compatible with the required environment, RIM sends back the marked resources to ACU. Further the Availability Checker Unit forwards these marked resources and runtime environment requirement to the Machine Scheduler. Machine Scheduler is responsible for giving away the control of compute cluster to the user. It does this by communicating environment requirements to the ESU to obtain a platform of user specifications. Once ESU boots the required machines it sends back a unique machine ID for the user which is used for future communication related to the platform. Once a desired machine with required platform is obtained, Machine Scheduler sends status and machine ID to the PM informing about the successful allocation of the platform. It also informs the Monitor so that it makes an entry in its table to keep track of running systems.

2.1.3 Resource Information Manager

The Resource Information Unit is responsible for maintaining information about the available resources. RIM maintains a list of all free machines in a table populated with information like RAM size, number of CPU cores, size of disk, NICs and runtime environment. These entries are maintained by the Monitor in the table. RIM gets requests from the ACU to check for required resources. RIM matches the request with the available resource. The reason to check the availability is that there is no notion of virtualization present, the resources become finite in number. Hence availability of resources is checked. Apart from availability RIM also checks for compatibility of required runtime environment and hardware as some hardware might not support all the libraries. If they match then those resources are marked and removed from the free list. The information about these marked resources like MAC IDs is passed back to the Availability Checker in the ACU.

2.1.4 Environment Setup Unit

The Environment Setup Unit is responsible for setting up the required environment for the user platform. The Machine Scheduler from the ACU sends MAC ids of the marked machines and the runtime environment to setup on those machines. The ESU has a netboot server which acts as NFS, DHCP and FTP server. All the OS images are stored here. All the machines in the resource pool are in low power mode and hence the server in the ESU sends a magic packet to bring up those machines i.e. a notion of Wake-on-LAN is used here. Once the desired machines are up the netboot server configures them for booting over network. Once the machines are booted with desired OS, required libraries are installed in the selected machines. After environment setup is done, ESU creates an ID associated with the compute cluster for provisioning to the user. This ID is unique and represents a single compute cluster. As the user interacts with the machines using this unique ID, total isolation of resources is achieved. This ID is given to the Machine Scheduler to notify it about the completion of platform creation.

2.1.5 Monitor

Monitor is responsible for keeping track of all the compute clusters and maintaining their state. It maintains the information in a table with fields such as User ID, compute cluster ID, MAC IDs of all machines present in the compute cluster and IP addresses associated with it and finally the status of the compute cluster. The User ID column need not be unique as a single user can own multiple compute clusters. The compute cluster ID field is unique one as each group is given different ID for isolation purpose. Monitor gets information about the new compute cluster from the Machine Scheduler to make an entry in the table. Until the machines are

freed, Monitor keeps track of all the activities and maintains the current status of the compute cluster. Monitor is also responsible for updating the free list of machines maintained by the RIM. Once a machine or group of machines is freed by a user, Monitor adds those machines in the free list again.

2.2 Working

The architecture presented in this report provides a platform with specified environment on user demand. The main component, ESU, is responsible for creating a compute cluster. It sets up a platform over the network which gives flexibility to the user to choose various options for the runtime environment. The management of the user machine is done through the user interface, PM which allows user to start, scale or shutdown the machine. The user monitors the system though this interface only. All the status updates from the Monitor are displayed to the user for monitoring of activity and for the statistics of resource usage. During a scale up request, the user sends requirements to ACU specifying that it is a scale up request and provides the ID associated with the machine and required amount of resources to scale. Now again the process goes as before like a fresh request but the only difference now is that rather than associating a new ID to the machine, the old ID sent by the user is given. Hence it becomes a single cluster associated with same ID.

For scaling down the user specifies the number of nodes that are no longer required for the computation task. Hence they are freed and corresponding changes are made in the table managed by the Monitor. When user wants to give away the resources, request to terminate the use is sent and corresponding machines are freed and kept ready for re-deploying. The respective changes are made with the Monitor. In this manner, the architecture provides the functionality of flexibility and elasticity similar to cloud for HPC workload.

2.3 Failure Handling

A machine in the resource pool can fail or may not respond at times. Failure can occur in two cases. Once when a machine is not responding to the ESU during a netboot operation and second when a machine is given to the user. In the first scenario, failure is less likely to happen because the monitor keeps a free machine list with RIM. During updating the free machine list monitor has to talk to the machine and then it gets added. Hence only those machines are kept which respond to the monitor. But still if the machine fails and does not respond during booting, the ESU send back an error message to the Machine Scheduler conveying failure of machines. The Machine Scheduler collects the information about the required resources from the Monitor and again requests the Availability Checker to check for the availability of those

resources. The Availability Checker again gives the required resources if available.

In the later case scenario, that is when the user owns the machine, the failure is detected by the Monitor. Monitor informs the Machine Scheduler about the failure and gives the information about failed machines. Machine Scheduler asks the Availability Checker for machines with same configurations. Once the machine MAC ids are obtained they are booted with required environment setup. Thus new set of nodes are booted up and are provided to the user.

2.4 Summary

In this section we discussed the proposed architecture designed to achieve better performance for HPC workloads in cloud setup. Each component of the architecture was discussed in detail. Also the working of the framework during a start, scaling and termination request was mentioned. A machine or set of machines might fail to respond or crash. Hence failure handling during such scenarios was discussed.

Chapter 3

Case Study

The proposed architecture in this report builds a cluster without virtualization of any resources. The resource allocated by the architecture are dedicated to a particular user and no other user intervenes in the working of it. Hence it can be said that the compute cluster provided by this architecture will be same as the non-virtualized dedicated machines used for HPC workloads. Hence the performance measurement on non-virtualized machines will justify the performance of compute cluster and we claim that compute cluster will give better performance than virtualized environment. To support our claim, we have conducted experiments to measure performance difference between virtualized and non-virtualized environment. Experiments were carried out to measure usage of physical resources like CPU, Memory, Disk and Network.

3.1 Experimental Setup

The experiments were conducted on a virtualized and a non-virtualized environment. The host machine had quad-core intel core i7 Processor with 8GB RAM and 1TB SATA2 disk. For setting up virtualized environment KVM was used as a hypervisor. All the VMs used in the experiments had single vCPU, 1 GB RAM and 30 GB disk space. All the VMs were pinned to separate CPU cores to obtain isolation and to avoid delay caused by CPU scheduling. The corresponding non-virtualized machine was booted with single CPU core and 1 GB RAM. All the machines used for the experiments were booted with Linux. The number of VMs were increased gradually and the experimental results were conducted on VMs running simultaneously. For experiments running on multiple VMs, only those results were taken into consideration whose completion time differ by negligible amount. Average result values was calculated for experiments running on multiple VMs. The benchmarks considered depict the HPC workload behavior in terms of CPU, memory, disk IO and networking. Stream benchmark was used to measure memory

bandwidth and corresponding execution time. NBench benchmark was used to measure CPU speed. IOR (Interleaved or Random) benchmark was used to measure parallel file system IO performance. And Iperf benchmark was used to measure bandwidth, packet loss and jitter during UDP and TCP data streams.

3.2 Benchmarks

3.2.1 NBench Benchmark

The NBench algorithm suite consists of ten different tasks:

- Numeric sort - Sorts an array of long integers.
- String sort - Sorts an array of strings of arbitrary length.
- Bitfield - Executes a variety of bit manipulation functions.
- Emulated floating-point - A small software floating-point package.
- Fourier coefficients - A numerical analysis routine for calculating series approximations of waveforms.
- Assignment algorithm - A well-known task allocation algorithm.
- Huffman compression - A well-known text and graphics compression algorithm.
- IDEA encryption - A relatively new block cipher algorithm.
- Neural Net - A small but functional back-propagation network simulator.
- LU Decomposition - A robust algorithm for solving linear equations.

A run of the benchmark suite consists essentially of two phases for each of the tests. First, a calibration loop is run to determine the size of the problem the system can handle in a reasonable time, in order to adapt to the ever faster computer hardware available. Second, the actual test is run repeatedly several times to obtain a statistically meaningful result.

Originally, NBench produced two overall index figures: Integer index and Floating-point index. The Integer index is the geometric mean of those tests that involve only integer processing-numeric sort, string sort, bitfield, emulated floating-point, assignment, Huffman, and IDEA-while the Floating-point index is the geometric mean of those tests that require the floating-point coprocessor-Fourier, neural net, and LU decomposition. The index figures where

relative scores to get a general feel for the performance of the machine under test as compared to a baseline system based on a 90 MHz Pentium Intel CPU.

The Linux/Unix port has a second baseline machine, it is an AMD K6/233 with 32 MB RAM and 512 KB L2-cache running Linux 2.0.32 and using GNU gcc version 2.7.2.3 and libc-5.4.38. The original integer index was split into an integer-operation and a memory-operation index, reflecting the realization that memory management is important in CPU design. The original tests have been left alone, however, the geometric mean of the tests numeric sort, floating-point emulation, IDEA, and Huffman now constitutes the integer-arithmetic focused benchmark index, while the geometric mean of the tests string sort, bitfield, and assignment makes up the new memory index. The floating point index has been left alone, it is still the geometric mean of fourier, neural net, and LU decomposition.

3.2.2 Stream Benchmark

The Stream benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels. Computer cpus are getting faster much more quickly than computer memory systems. As this progresses, more and more programs will be limited in performance by the memory bandwidth of the system, rather than by the computational performance of the cpu. As an extreme example, several current high-end machines run simple arithmetic kernels for out-of-cache operands at 4-5% of their rated peak speeds, that means that they are spending 95-96% of their time idle and waiting for cache misses to be satisfied. The Stream benchmark is specifically designed to work with datasets much larger than the available cache on any given system, so that the results are more indicative of the performance of very large, vector style applications.

3.2.3 IOR Benchmark

IOR is designed to measure parallel file system I/O performance at both the POSIX and MPI-IO level. IOR stands InterleavedOrRandom, which has very little to do with how the program works currently. This parallel program performs writes and reads to/from files under several sets of conditions and reports the resulting throughput rates. IOR performs get a time stamp START, then has all participating tasks open a shared or independent file, transfer data, close the file(s), and then get a STOP time. A stat() or MPI-File-get-size() is performed on the file(s) and compared against the aggregate amount of data transferred. If this value does not match, a warning is issued and the amount of data transferred as calculated from write(), e.g., return codes is used. The calculated bandwidth is the amount of data transferred divided by the elapsed STOP-minus-START time. IOR also gets time stamps to report the open, transfer,

and close times. Each of these times is based on the earliest start time for any task and the latest stop time for any task. Without using barriers between these operations, the sum of the open, transfer, and close times may not equal the elapsed time from the first open to the last close.

3.2.4 Iperf Benchmark

Iperf is a commonly used network testing tool that can create Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) data streams and measure the throughput of a network that is carrying them. Iperf is a tool for network performance measurement written in C. It is a compatible reimplementation of the ttcp program that was developed by the Distributed Applications Support Team (DAST) at the National Laboratory for Applied Network Research (NLANR). Iperf allows the user to set various parameters that can be used for testing a network, or alternatively for optimizing or tuning a network. Iperf has a client and server functionality, and can measure the throughput between the two ends, either unidirectionally or bi-directionally. It is open-source software and runs on various platforms including Linux, Unix and Windows.

UDP: When used for testing UDP capacity, Iperf allows the user to specify the datagram size and provides results for the datagram throughput and the packet loss.

TCP: When used for testing TCP capacity, Iperf measures the throughput of the payload. Iperf uses 1024 1024 for megabytes and 1000 1000 for megabits.

Typical Iperf output contains a time-stamped report of the amount of data transferred and the throughput measured. Iperf is significant as it is a cross-platform tool that can be run over any network and output standardized performance measurements. Thus it can be used for comparison of both wired and wireless networking equipment and technologies.

3.2.5 OSU micro Benchmark

The Ohio MicroBenchmark suite is a collection of independent MPI message passing performance microbenchmarks developed and written at The Ohio State University. It includes traditional benchmarks and performance measures such as latency, bandwidth and host overhead and can be used for both traditional and GPU-enhanced nodes. Following tests were used for the purpose of this project.

- osu-latency : Single-pair Ping-pong Latency
- osu-bw : Ping-pong Bandwidth
- osu-bibw : Bi-directional Bandwidth

3.3 Results for Benchmarks

3.3.1 CPU Performance

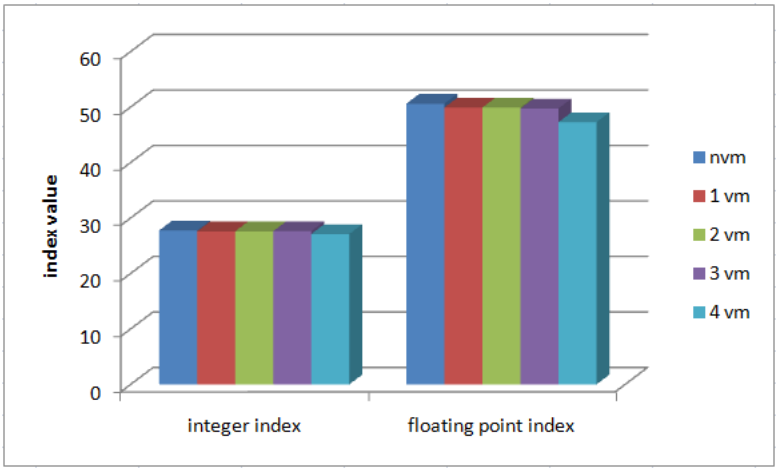

Figure 3.1: NBench : CPU performance

For measuring CPU performance NBench benchmark was used. NBench algorithm consist ten tasks which are a mixture of integer and floating point operations. Due to this mix, this benchmark resembles the HPC compute intense workload. The iterations per cycle are calculated for each algorithm. The integer index and the floating point index plotted in the graph are nothing but geometric mean of tests that require integer and floating point processing respectively. The benchmark was run on non-virtualized and virtualized machines and the number of VMs was increased gradually. It is clear from the graph shown in figure 3.1, that CPU performance is not affected by the virtualization layer and the performance is comparable with the non-virtualized machine. Thus for pure compute intensive jobs virtualization layer does not act as an overhead.

3.3.2 Memory Performance

Stream benchmark is a popular benchmark in HPC community. It measures the memory bandwidth and the corresponding rate for four simple vector kernels. During the experiment, the data handled by Stream was kept more than the size of cache to eliminate any caching effect. The four vector kernels run multiple times and average bandwidth is calculated for those runs. The graph in figure 3.2 clearly shows that the bandwidth of non-virtualized machine is greater than the virtual machine. Further as we increase the number of VMs, the individual bandwidth for each VM decreases. The reason behind this performance drop is the overhead caused by

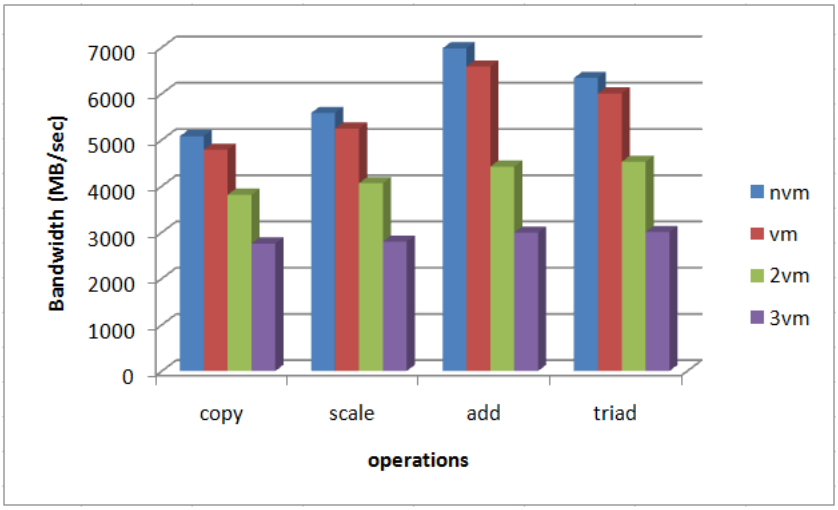

Figure 3.2: Stream : Memory Bandwidth

the virtualization layer in translating the memory address instruction. As the instruction goes through an extra address translation of the hypervisor, bandwidth of the VM suffers. The other graph in figure 3.3 shows execution rate for completing a task for each machine. As the bandwidth decreases, the corresponding execution time raises. This effect is due to overhead caused by virtualization layer as VMs need extra time for going through extra abstraction layer. Hence virtualization layer causes memory bandwidth to suffer and does not provide expected performance guarantee.

3.3.3 Disk Performance

IOR is designed to measure parallel file system I/O performance at both the POSIX and MPI-IO level. This parallel program performs writes and reads to/from files under several sets of conditions and reports the resulting throughput rates. The api used here for conducting experiments is MPI-IO with number of clients set to 16. To avoid the possibility of caching, the file size was kept more than the free memory on each client hence the file size used here is 8 GB. While performing the operation, each participating task opens a shared file, transfers the data and closes the file. The time elapsed for transferring the data and total amount of data transferred is calculated from which bandwidth is obtained. As seen from the graph shown in figure 3.4, there is slight drop in write bandwidth while 30% drop in read bandwidth for a single VM case. As the number of VMs are increased the write and read bandwidth starts to degrade further. This drop in bandwidth is caused due to additional level of abstraction. Due to this layer, VM has to go through an extra address translation at hypervisor level as well as host OS level. Due to this delay the bandwidth of the VM suffers. Thus virtualization of disk

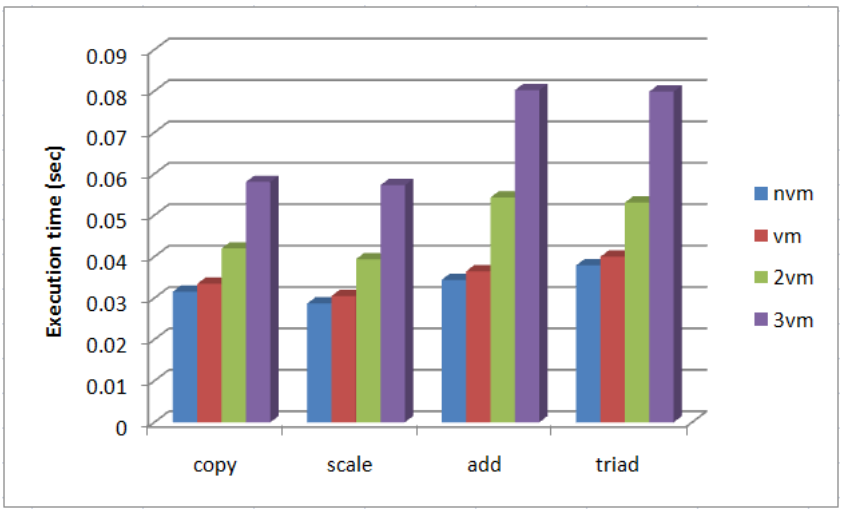

Figure 3.3: Stream : Execution Time for Memory Operations

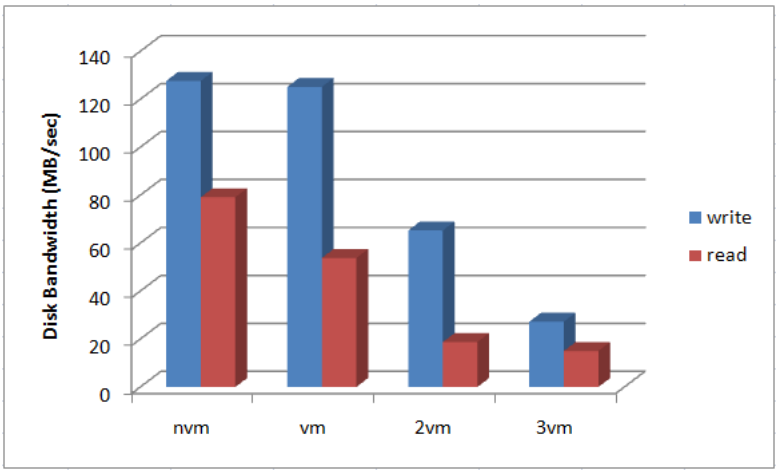

Figure 3.4: IOR : Disk Bandwidth

leads to degradation of IO performance.

3.3.4 Network Performance

Iperf benchmark creates Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) data streams and measure the throughput of a network that is carrying them. For conducting this experiment, server and client were setup on different machines present in the same network. For virtualization case, server was setup on a VM while client was setup on different machine in a non-virtualized environment. The number of server was increased gradually. Each VM was running as a server while clients were connected to each server separately. To get accurate results, experiment was conducted during a time when network usage was mini-

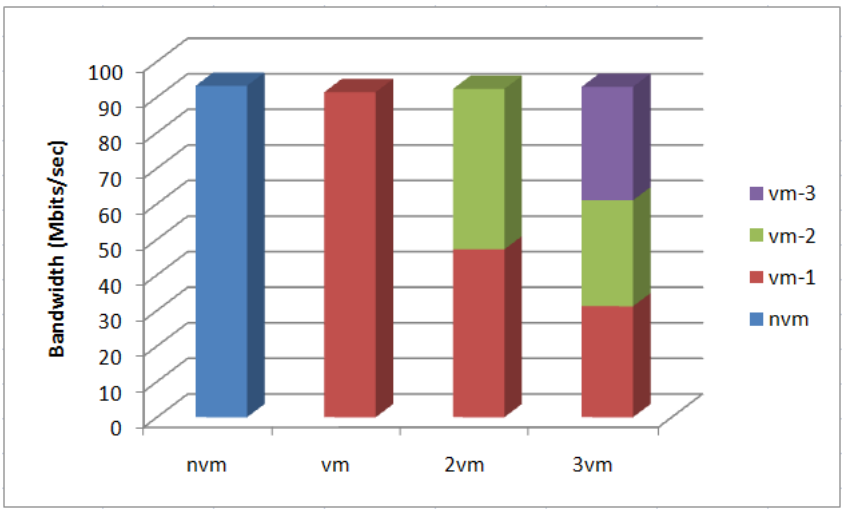

Figure 3.5: Iperf : Bandwidth Measurement for UDP

mum. Network performance was evaluated in terms of bandwidth, jitter and packet loss during communication. Two separate tests for UDP and TCP were conducted. UDP transmission gives information about bandwidth, jitter and packet loss. As seen from the graph in figure 3.5, the total bandwidth during UDP transmission remains same while individual VM bandwidth is decreased as we increase the number of VMs. It can be seen clearly that the bandwidth is uniformly distributed among all the VMs. The graph in figure 3.6 shows jitter i.e. delay for response, which increases with number of VMs. As the VMs share the same NIC, the communication overhead increases with increased number of VMs. During the UDP transmission, packet loss is seen in the network. This packet loss also differs from machine to machine. Packet loss can be seen in the third graph from figure 3.7, where there is nearly 50% packet loss when 2 VMs are running simultaneously and 70% packet loss when 3 VMs run simultaneously.

The TCP transmission gives information about the bandwidth alone. TCP waits for an acknowledgement for certain amount of time and if it doesnt get one it considers that the packet was lost and again transmits the packet. Hence it ensures that all packet reach destination. Hence there is no notion of packet loss in TCP transmission. Figure 3.8 shows bandwidth of a network during TCP transmission. Same experimental setup as UDP was repeated for conducting this experiment. It can be seen from the graph that the total bandwidth used remains same while individual bandwidth for each VM differ. Also it is observed from the graph that the individual bandwidth of VMs is not as uniform as in case of UDP. This is due to the re-transmissions caused when a packet or its acknowledgement is lost. When a re-transmission is caused the host resets its TCP window and size of buffer to minimum default size. And thus the throughput suffers due to this. Thus sharing of NIC causes each VM to wait

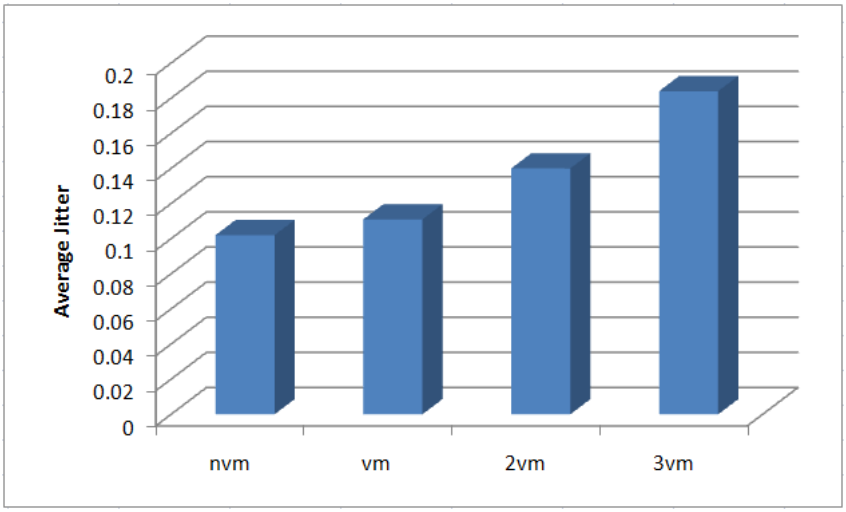

Figure 3.6: Iperf : Jitter during UDP Transmission

for its share of packets and this delay causes re-transmission of packets. Thus virtualization of NIC is a bottleneck for VMs. we

We have also conducted experiments with the MPI version of OSU micro benchmarks to check the communication bandwidth and communication latency amoung the MPI processes. The algorithms checked the bandwidth, bi-directional bandwidth and latency for point-to-point communication. Measures uni-directional bandwidth from one to another MPI rank. Several Isends are started followed by a Waitall. The receving side uses matching Irecvs with Waitall. One iteration ends when the sending side gets the receive of all messages acknowledged. The graph in figure 3.9 shows the bandwidth measured for different message sizes. It is observed that we get a peak performance for message size of approximately 64 KB. Also as the message size increases bandwidth for multiple VM case decreases.

Figure 3.10 shows results for bi-directional bandwidth for point-to-point communication. This benchmark works as the uni-directional bandwidth benchmark only that both sides issue first Irecvs followed by Isends and Waitalls. Here also a peak performance is observed for message size of 64 KB. The VM performance keeps decreasing as the message size is increased.

Figure 3.11 shows the graph for latency during point-to-point communication. Latency denotes the time taken to transfer a message of a certain size from one MPI rank to another. It was observed that the latency for multiple VM case drastically increases with increased message size.

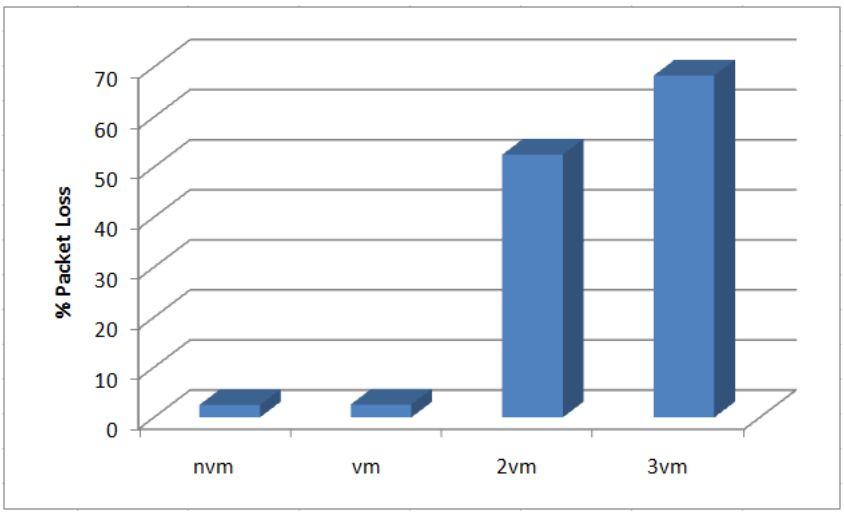

Figure 3.7: Iperf : Packet Loss during UDP Transmission

3.4 Summary

In this chapter we studied about the different benchmarks used and how they are related to the HPC workload. We also presented the results obtained for each of these benchmark. It was clearly seen that virtualization layer causes performance degradation. The comparision between virtualized and non-virtualized platforms depicts the comparision between the cloud and the proposed solution setup.

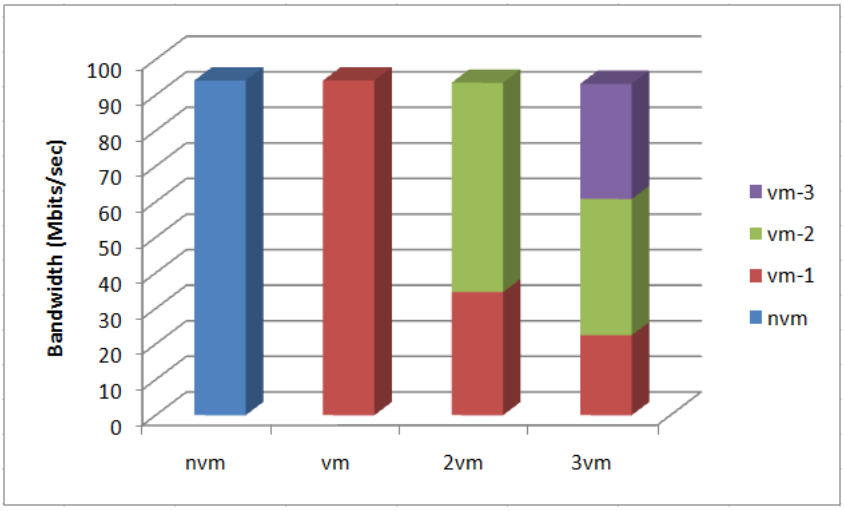

Figure 3.8: Iperf : Bandwidth Measurement for TCP

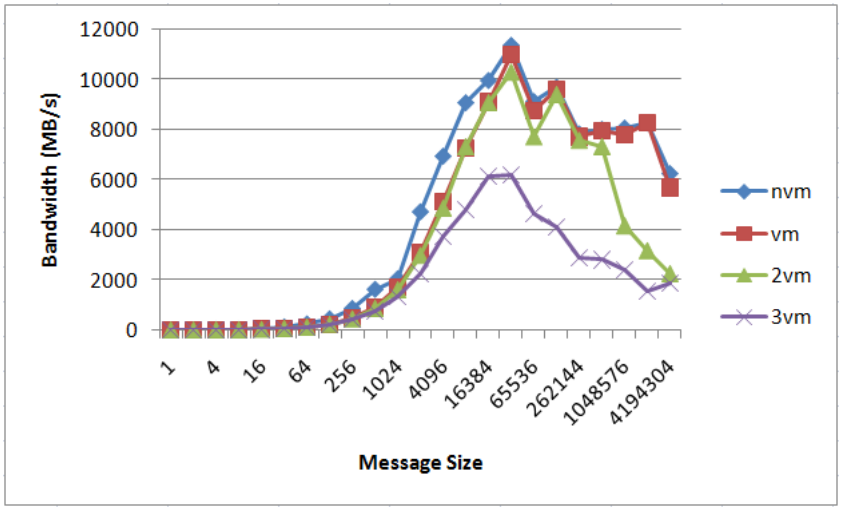

Figure 3.9: OSU micro benchmarks : Bandwidth for point-to-point communication

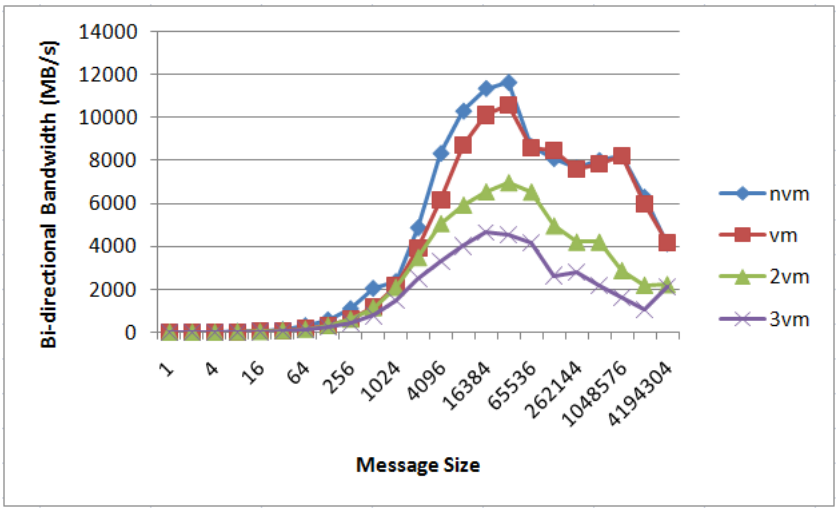

Figure 3.10: OSU micro benchmarks : Bi-directional bandwidth for point-to-point communication

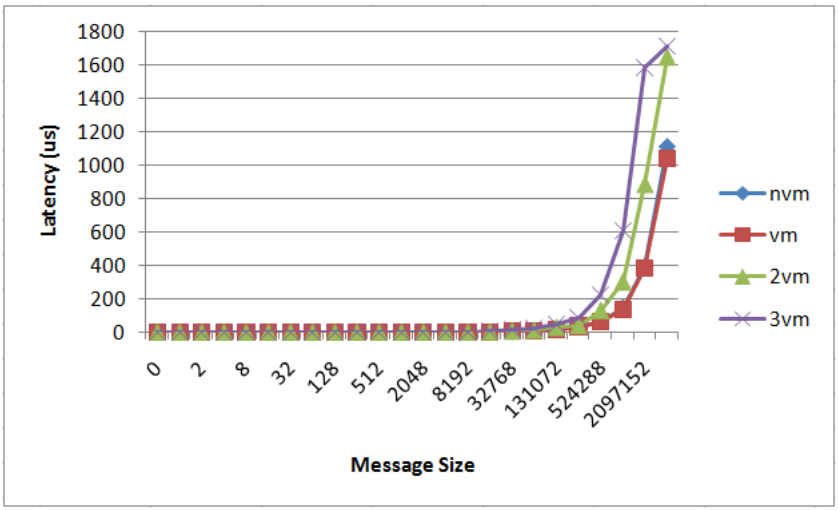

Figure 3.11: OSU micro benchmarks : Latency for point-to-point communication

Chapter 4

Conclusion and Future Work

In this report we studied different performance tradeoffs of HPC applications. We evaluated performance across different physical resources and studied the effect of virtualization on each of them and found that HPC applications do not give fair performance on virtualized environment. We presented an architecture to overcome the performance degradation caused by the virtualization layer.

The current architecture has the functionality of starting, scaling and terminating the compute cluster. The future work will consist of improving the current architecture for better functionality and adding more features to it such as pausing/resuming and saving the state of compute cluster to make it a full fledged framework which acts as a cloud setup for HPC applications.

Bibliography

[1] Constantinos Evangelinos and C Hill. Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazons ec2. *ratio*, 2(2.40):2–34, 2008. 2, 3

[2] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. *technology*, 28:32, 2014. 4

[3] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732–749, 2011. 3

[4] Abhishek Gupta and Dejan Milojicic. Evaluation of hpc applications on cloud. In *Open Cirrus Summit (OCS), 2011 Sixth*, pages 22–26. IEEE, 2011. 2, 3

[5] Abhishek Gupta, Paolo Faraboschi, Filippo Gioachin, Laxmikant V Kale, Richard Kaufmann, B-S Lee, Verdi March, Dejan Milojicic, and Chun Hui Suen. Evaluating and improving the performance and scheduling of hpc applications in cloud. 2014. 2, 3

[6] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 159–168. IEEE, 2010. 2

[7] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B Lee. Nohype: virtualized cloud infrastructure without the virtualization. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 350–361. ACM, 2010. 3

[8] Viktor Mauch, Marcel Kunze, and Marius Hillenbrand. High performance cloud computing. *Future Generation Computer Systems*, 29(6):1408–1416, 2013. 2

[9] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini, and Rupak Biswas. Performance evaluation of amazon ec2 for nasa hpc applications. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, pages 41–50. ACM, 2012. 3

[10] Malek Musleh, Vijay Pai, John Paul Walters, Andrew Younge, and Stephen Crago. Bridging the virtualization performance gap for hpc using sr-iov for infiniband. In *Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on*, pages 627–635. IEEE, 2014. 4

[11] Satish Srirama, Oleg Batrashev, and Eero Vainikko. Scicloud: scientific computing on the cloud. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 579–580. IEEE Computer Society, 2010. 3

[12] Guohui Wang and TS Eugene Ng. The impact of virtualization on network performance of amazon ec2 data center. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010. 2, 3

[13] Katherine Yelick, Susan Coghlan, Brent Draney, Richard S Canon, et al. The magellan report on cloud computing for science. *US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR)*, 2011. 3

[14] Andrew J Younge, John Paul Walters, Jinwoo Suh, Dong-In D Kang, Youngsam Park, Stephen P Crago, and Geoffrey C Fox. Towards a high performance virtualized iaas deployment. 3