

2012

# Empirical Performance Analysis of High Performance Computing Benchmarks Across Variations in Cloud Computing

Sindhu Mani  
*University of North Florida*

---

## Suggested Citation

Mani, Sindhu, "Empirical Performance Analysis of High Performance Computing Benchmarks Across Variations in Cloud Computing" (2012). *UNF Graduate Theses and Dissertations*. 418.  
<https://digitalcommons.unf.edu/etd/418>

This Master's Thesis is brought to you for free and open access by the Student Scholarship at UNF Digital Commons. It has been accepted for inclusion in UNF Graduate Theses and Dissertations by an authorized administrator of UNF Digital Commons. For more information, please contact [Digital Projects](#).

© 2012 All Rights Reserved

Empirical Performance Analysis of High Performance Computing Benchmarks Across  
Variations in Cloud Computing

by

Sindhu Mani

A thesis submitted to the  
School of Computing  
in partial fulfillment of the requirements for the degree of

Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA  
SCHOOL OF COMPUTING

December 2012

Copyright © 2012 by Sindhu Mani

All rights reserved. Reproduction in whole or in part in any form requires the prior written permission of Sindhu Mani or designated representative.

The thesis “Empirical Performance Analysis of High Performance Computing Benchmarks Across Variations in Cloud Computing” submitted by Sindhu Mani in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been

Approved by the thesis committee:

Date

---

Dr. Sanjay P. Ahuja  
Thesis Advisor and Committee Chairperson

---

Dr. Roger Eggen  
Committee Chairperson

---

Dr. Saurabh Gupta  
Committee Chairperson

Accepted for the School of Computing:

---

Dr. Asai Asaithambi  
Director of the School

Accepted for the College of Computing, Engineering, and Construction:

---

Dr. Mark A. Tumeo  
Dean of the College

Accepted for the University:

---

Dr. Len Roberson  
Dean of the Graduate School

## ACKNOWLEDGEMENT

This thesis would not have been possible without the direction and support of my thesis advisor Dr. Sanjay P. Ahuja. I thank my committee members Dr. Saurabh Gupta and Dr. Roger Eggen for providing valuable suggestions. In addition, thank you to the support teams of various cloud computing tools provided for Amazon EC2 and Microsoft Windows Azure for patiently answering my questions when faced with several challenges. I also thank Dr. Asai Asaithambi and Dr. Karthikeyan Umapathy for being present at my thesis defense and providing suggestions. I would also like to thank James Littleton for providing valuable editorial suggestions on the thesis write up.

Finally, I thank my husband and my family for their continuous support, encouragement and love during the long process in achieving this important milestone. This thesis presented a great opportunity for me to learn about the currently evolving cutting edge cloud computing platforms.

## CONTENTS

List of Figures .....	x
List of Tables.....	xii
Abstract.....	xiv
Chapter 1: Introduction.....	1
1.1 Services in the Cloud .....	2
1.2 Cloud Architectures .....	3
1.2.1 Amazon EC2.....	3
1.2.2 Windows Azure.....	4
1.3 HPC in the Cloud.....	6
1.4 Examples of HPC Applications .....	6
1.5 Thesis Layout.....	7
Chapter 2: Literature review .....	8
2.1 Communication and Computational Performance.....	10
2.2 Memory Bandwidth .....	11
2.3 Input/Output Performance .....	12
Chapter 3: Research Methodology.....	15

3.1	STREAM Benchmark.....	15
3.2	Interleaved Or Random Benchmark .....	17
3.3	NAS Parallel Benchmarks .....	17
3.4	Amazon Web Service EC2 Platform.....	18
3.4.1	Master and Compute Nodes.....	19
3.4.2	EBS Volume and Instance Storage.....	20
3.5	Microsoft Windows Azure Platform .....	21
3.5.1	Web and Worker Roles.....	21
3.5.2	Head Node and Compute Node .....	22
3.5.3	Windows Azure Storage: .....	22
Chapter 4: Hardware and Software Specifications .....		24
4.1	Software Specifications .....	24
4.2	Benchmarks.....	25
4.3	Hardware Specifications .....	26
Chapter 5: Setting up, Configuring and Benchmarking EC2 .....		29
5.1	Pre-requisites on the Local Windows Development Machine .....	29
5.2	Installing StarCluster to Build the Cluster .....	29
5.3	Edit StarCluster Configuration File .....	30
5.3.1	Amazon Machine Image.....	30
5.3.2	Plugins - Message Passing Interface.....	31

5.3.3	Scaling.....	31
5.4	Starting the Cluster with MPICH2.....	32
5.4.1	AWS Management Console.....	32
5.5	Transfer the Benchmark Files to the Cluster .....	33
5.5.1	Network File System.....	33
5.6	Execute Benchmarks on the Master Node .....	34
5.6.1	STREAM Benchmark.....	34
5.6.2	Interleaved Or Random Benchmark .....	35
5.6.3	NAS Parallel Benchmarks .....	37
Chapter 6:	Setting up, Configuring and Benchmarking Windows Azure .....	43
6.1	Building Windows Binaries .....	43
6.2	Pre-requisites on the Local Windows Development Machine.....	44
6.3	Deploy Windows Azure HPC Scheduler via PowerShell.....	44
6.3.1	Service Configuration and Service Definition Files.....	45
6.4	Connect to HeadNode on the Cluster on Windows Azure.....	46
6.5	Windows Azure Firewall Configuration for MPI Communication .....	46
6.6	Create and Submit MPI Jobs for Executing Benchmarks.....	48
6.6.1	STREAM Benchmark.....	48
6.6.2	Interleaved Or Random Benchmark .....	49
6.6.3	NAS Parallel Benchmarks .....	50



Chapter 7: Analysis of Results .....	54
7.1 STREAM Benchmark .....	54
7.1.1 EC2 Standard Small Instance (m1.small) Versus Azure Small.....	55
7.1.2 EC2 Standard Medium Instance (m1.medium) Versus Azure Medium.....	59
7.1.3 EC2 High-CPU Medium instance (c1.medium) Versus Azure Medium .....	62
7.2 Interleaved Or Random Benchmark .....	65
7.2.1 EC2 Standard Small Instance (m1.small) Versus Azure Small.....	66
7.2.2 EC2 Standard Medium Instance (m1.medium) Versus Azure Medium.....	68
7.2.3 EC2 High-CPU Medium Instance (c1.medium) Versus Azure Medium .....	70
7.3 NAS Parallel Benchmarks (NPB-CG, FT, EP).....	71
7.3.1 EC2 Standard Small Instance (m1.small) Versus Azure Small.....	72
7.3.2 EC2 Standard Medium Instance (m1.medium) Versus Azure Medium.....	76
7.3.3 EC2 High-CPU Medium Instance (c1.medium) Versus. Azure Medium .....	80
Chapter 8: Conclusion.....	85
8.1 Future Research .....	88
References.....	90
Appendix A: Metrics.....	94
Appendix B: Configuration File and Benchmark Commands.....	96
Appendix C: Sample Results .....	104

Appendix D: EC2 Screenshots.....	110
Appendix E: Azure Screenshots.....	116
Vita.....	121

## LIST OF FIGURES

Figure 1: Connecting to EC2 Cluster From Client Machine Installed With StarCluster..	19
Figure 2: Microsoft Windows Azure Roles .....	23
Figure 3: STREAM Copy - m1.small and Azure Small Instance Average .....	56
Figure 4: STREAM Scale - m1.small and Azure Small Instance Average .....	56
Figure 5: STREAM Add - m1.small and Azure Small Instance Average .....	58
Figure 6: STREAM Triad - m1.small and Azure Small Instance Average .....	58
Figure 7: STREAM Copy - m1.medium and Azure Medium Instance .....	61
Figure 8: STREAM Scale - m1.medium and Azure Medium Instance .....	61
Figure 9: STREAM Add- m1.medium and Azure Medium Instance .....	61
Figure 10: STREAM Triad - m1.medium and Azure Medium Instance.....	62
Figure 11: STREAM Copy - c1.medium and Azure Medium Instance.....	64
Figure 12: STREAM Scale - c1.medium and Azure Medium Instance.....	64
Figure 13: STREAM Add - c1.medium and Azure Medium Instance.....	65
Figure 14: STREAM Triad - c1.medium and Azure Medium Instance.....	65
Figure 15: IOR - m1.small and Azure Small Instance .....	67
Figure 16: IOR - m1.medium and Azure Medium Instance .....	69
Figure 17: IOR - c1.medium and Azure Medium Instance.....	70
Figure 18: CG - m1.small and Azure Small Instance .....	73
Figure 19: FT - m1.small and Azure Small Instance .....	74

Figure 20: EP - m1.small and Azure Small Instance .....	75
Figure 21: CG - m1.medium and Azure Medium Instance.....	76
Figure 22: FT - m1.medium and Azure Medium Instance.....	78
Figure 23: EP - m1.medium and Azure Medium Instance.....	79
Figure 24: CG - c1.medium and Azure Medium Instance .....	81
Figure 25: FT - c1.medium and Azure Medium Instance .....	82
Figure 26: EP - c1.medium and Azure Medium Instance .....	83

## LIST OF TABLES

Table 1: Major Contributions.....	9
Table 2: Hardware Specifications of EC2 and Azure Instance Types.....	27
Table 3: STREAM - m1.small and Azure Small Instance Average MB/S.....	57
Table 4: STREAM - m1.small and Azure Small Instance Average Time in S.....	57
Table 5: STREAM - m1.medium and Azure Medium Instance Average MB/S .....	60
Table 6: STREAM - m1.medium and Azure Medium Instance Average Time in S .....	60
Table 7: STREAM - c1.medium and Azure Medium Instance Average MB/S .....	63
Table 8: STREAM - c1.medium and Azure Medium Instance Average Time in S .....	63
Table 9: IOR - m1.small and Azure Small Instance Average MiB/S .....	67
Table 10: IOR - m1.medium and Azure Medium Instance Average MiB/S .....	69
Table 11: IOR - c1.medium and Azure Medium Instance Average MiB/S.....	71
Table 12: CG - m1.small and Azure Small Instance .....	73
Table 13: FT - m1.small and Azure Small Instance .....	74
Table 14: EP - m1.small and Azure Small Instance .....	75
Table 15: CG - m1.medium and Azure Medium Instance .....	77
Table 16: FT - m1.medium and Azure Medium Instance .....	78
Table 17: EP - m1.medium and Azure Medium Instance .....	79
Table 18: CG - c1.medium and Azure Medium Instance.....	81
Table 19: FT - c1.medium and Azure Medium Instance.....	82

Table 20: EP - c1.medium and Azure Medium Instance..... 83

## ABSTRACT

High Performance Computing (HPC) applications are data-intensive scientific software requiring significant CPU and data storage capabilities. Researchers have examined the performance of Amazon Elastic Compute Cloud (EC2) environment across several HPC benchmarks; however, an extensive HPC benchmark study and a comparison between Amazon EC2 and Windows Azure (Microsoft's cloud computing platform), with metrics such as memory bandwidth, Input/Output (I/O) performance, and communication computational performance, are largely absent. The purpose of this study is to perform an exhaustive HPC benchmark comparison on EC2 and Windows Azure platforms.

We implement existing benchmarks to evaluate and analyze performance of two public clouds spanning both IaaS and PaaS types. We use Amazon EC2 and Windows Azure as platforms for hosting HPC benchmarks with variations such as instance types, number of nodes, hardware and software. This is accomplished by running benchmarks including STREAM, IOR and NPB benchmarks on these platforms on varied number of nodes for small and medium instance types. These benchmarks measure the memory bandwidth, I/O performance, communication and computational performance. Benchmarking cloud platforms provides useful objective measures of their worthiness for HPC applications in addition to assessing their consistency and predictability in supporting them.

## Chapter 1

### INTRODUCTION

The increasing levels of research and IT investment in cloud computing indicate that cloud computing is fast emerging as the next generation technology for computational needs. The “cloud” refers to a combination of both hardware and software applications available over the Internet as services. The cloud also provides applications as services to store, retrieve, and share data from systems connected to the Internet. In other words, the applications themselves need not be installed on the client machine.

Large data centers used to build this “cloud” are designed to support highly scalable applications. These data centers usually consist of several thousand interconnected computing devices capable of handling remote requests to run large and small applications. The companies housing these data centers (Google, Amazon, Sun Microsystems, and Microsoft, to name a few) actually bear the costs associated with them in addition to providing software updates. This type of service is called Public Cloud [Gillam10]. On the other hand, if the service is solely used within an organization and not shared with people outside of the organization it is called Private Cloud [Velte10]. There is also a third kind, a combination of public and private cloud. It is referred to as Hybrid cloud [Velte10].



Choosing which one to deploy purely depends on the organization's needs. Two of the most important concerns in a cloud-based environment are security and performance. Performance has been particularly a topic of interest for researchers as it heavily impacts their applications that require high CPU and data storage capacities.

## 1.1 Services in the Cloud

Before getting into HPC it is important to understand what type of services are currently out there in cloud computing and how they fit into the above-mentioned models. These are categorized as Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [SunMicrosystems09].

Organizations provide SaaS on demand. An example of a software application that is offered as a service is Google Apps that manages pictures, email service, or calendar. Another example is salesforce.com, which provides software solutions for sales and marketing on the cloud. SaaS can be provided to individuals as well as organizations as needed.

PaaS provides developers a platform to build and deploy software applications. The support is provided in the form of OS, development environment and middleware. APIs (Application Program Interfaces) are provided so that developers can interact with the environment to connect and deploy their applications. In addition, PaaS also provides tools to maintain these applications. Google App Engine is an example of PaaS that

provides an infrastructure and environment for application developers.

Finally, IaaS provides the storage, data center spaces, servers and other networking devices such as routers and the provisioning computer clusters as needed. The primary purpose of IaaS is to handle the workload for computational needs. Amazon Elastic Compute Cloud (EC2) platform is an example of IaaS.

The cloud provides the architecture of hardware and software for computational needs for both organizations and consumers. Many consumer services focus on web services that rely on relatively less intensive tasks and hence performance is not necessarily an issue in these situations.

## 1.2 Cloud Architectures

While quite a few cloud architectures exist, this thesis focuses on benchmarking Amazon EC2's (Elastic Compute Cloud) and Windows Azure with STREAM, IOR and NPB benchmarks. These two platforms are of IaaS and PaaS types and hence present a great opportunity for performance comparison.

### 1.2.1 Amazon EC2

While Amazon EC2 provides the web services to its instances, its S3, also referred to as Simple Storage Service, provides a storage service. Together they provide the compute

cluster and storage capacity needed for cloud servicing. These clusters can be created and destroyed per demand [Evangelinos08].

Primarily, EC2 is built on Linux and Xen [Sun Microsystems09, Evangelinos08].

However, it supports wide range of Operating Systems including Red Hat Linux, Windows Server, Amazon Linux Amazon Machine Image (AMI), Oracle Enterprise Linux, and OpenSolaris. EC2 provides infrastructure for scalable compute capacity in the cloud. Amazon provides it as Infrastructure as a Service (IaaS). The applications it supports can be highly scalable, which is one of the requirements for the HPC applications in the cloud.

Amazon uses a variety of measures to provide a consistent and predictable amount of CPU capacity (GHz, clock speed). This is for the developers to compare the CPU capacities among different instances types. For this purpose, Amazon has defined an Amazon EC2 Compute Unit. The amount of CPU for a particular instance is expressed in terms of these EC2 Compute Units. According to Amazon.com, “One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor” [AWS12A].

### 1.2.2 Windows Azure

Windows Azure is Microsoft’s application platform for public cloud and is offered as PaaS. This platform can also be used for parallel processing which is the basis of High-

Performance computing (HPC). On Windows Azure, this means running many role instances simultaneously, all working in parallel to perform tasks. Windows Azure provides the HPC Scheduler for distributing their work across the instances. The HPC Scheduler can be used with so-called embarrassingly parallel applications and with HPC applications built to use the industry-standard Message Passing Interface (MPI) [WindowsAzure12].

There are various roles provided by Azure that make up the complete application. They are Web Roles and Compute Roles. For each role, the desired Virtual Machine (VM) size that the instances of that role should use is indicated. The various VM sizes available are Extra Small, Small, Medium, Large and Extra Large.

#### 1.2.2.1 Windows HPC Server 2008 R2

Windows HPC Server 2008 R2 is an Operating System provisioned on the Head and compute nodes on Windows Azure platform. It supports both 32-bit and 64-bit programs. It provides powerful virtualization capabilities and supports MS-MPI (Microsoft-Message Passing Interface) for scalable applications. MS-MPI is Microsoft's implementation of MPICH. MPI is an essential feature for computing in clusters. It is installed with Windows HPC Pack 2008 R2 that has utilities to submit and monitor HPC MPI (Message Passing Interface) jobs [Microsoft12].

### 1.3 HPC in the Cloud

The challenge for the cloud is when the computational needs of applications are increased manifold such as the needs of scientific applications warranting supercomputer capabilities. Examples could be building 3D models from large amount of data for scientific research and development and grid computing. Today HPC systems use supercomputers and computer-clusters to solve advanced problems. These computer-clusters involve network of systems with parallel programming capabilities in multiple disciplines such as system software, architecture and computational techniques.

The traditional HPC technologies provide the tools to build HPC systems. Adequate hardware and software services may have to be provisioned in the cloud in order to handle its high performance needs as the applications running on these systems may require hundreds of thousands of CPU-hours [Hazelhurst08].

### 1.4 Examples of HPC Applications

As mentioned above, HPC applications are mostly of scientific nature in areas of mathematics, weather, and life sciences, and extensive data processing occurs in such applications. Examples include solving sparse real and complex linear equations, scientific prototyping and extensive data processing, and solving complex algebraic equations; weather forecasting models including 3D models; and recognize protein signatures.

## 1.5 Thesis Layout

This thesis evaluates and analyzes performance of two public clouds spanning both IaaS and PaaS types. EC2 and Windows Azure are used as platforms for hosting HPC benchmarks and executing them with variations such as instance types, number of nodes, hardware and software. The metrics used to analyze these public clouds are memory bandwidth, I/O performance, and Computational and Communication performance.

The rest of this thesis is structured as follows. Following this Introduction will be the chapter on Literature Review in which the works of other researchers in the area of HPC in the cloud are surveyed. The Literature Review chapter also helped identify the benchmarks for this thesis. Following the Literature Review chapter is the chapter on Research Methodology. This chapter describes in detail the methodology used in this study and the types of benchmarks used. Chapters detailing the setup of the cloud environments and execution of the HPC benchmarks in these environments are presented next. Finally, the chapter on the Analysis of Results presents a detailed analysis of the results obtained.

## Chapter 2

### LITERATURE REVIEW

This chapter provides a survey of the works of other researchers who have investigated the performance of HPC benchmarks in cloud environments. HPC benchmarks were originally designed to assess the performance of traditional supercomputers and distributed computing systems. In this thesis, these benchmarks are used for the same purpose to compare the performance of two public clouds, Amazon EC2 and Windows Azure platforms for HPC applications.

Previous performance studies have used some standard HPC benchmarks and metrics such as memory bandwidth, input/output capabilities, communication and computational performances. In this chapter, we highlight works related to the standard HPC benchmarks along with the metrics used for benchmarking cloud platforms. The following table summarizes the relevant information.

Study	Platform Investigated	Benchmarks Used	Major Conclusions
Computation & Communication Performance	EC2	NPB	Executed on 1 node. Cluster is not used. M1.medium instance type is not benchmarked.
Memory Bandwidth	EC2	STREAM	Executed on 1 node. Cluster is not used. M1.medium instance type is not benchmarked.
I/O Performance	EC2	IOR	Executed on 1 node. Cluster is not used. M1.medium instance type is not benchmarked.
Computation & Communication Performance Memory Bandwidth I/O Performance	Windows Azure	NPB STREAM IOR	These benchmarks are not investigated on Windows Cloud Environment.

Table 1: Major Contributions



## 2.1 Communication and Computational Performance

Amedro *et al.* launched the MPI NAS Parallel benchmarks on four different architectures: Private cluster, Amazon small instance, High-CPU Medium instance and High-CPU X-Large instance [Amedro10]. The throughput and latency for both the small and medium instances reflected moderate EC2 I/O performance whereas XLarge instance had high EC2 I/O performance. However, there is a large gap for latency when compared to the private cloud.

Amedro's research team also conducted tests to determine the performance in mflops of the following three NAS Parallel benchmarks. For up to 32 processes, one process is run per machine and then the number of processes is increased [Amedro10].

Embarrassingly Parallel (EP) Benchmark: In EP problem there is no communication between processes, hence it proves to be a test for pure computational speed. The XLarge running at 2.3 GHz and eight cores has almost the same speed compared to the private cluster.

Conjugate Gradient (CG) Benchmark: This benchmark is a test for communication performance. It computes Conjugate Gradient involving large number of small messages. In both EP and CG benchmarks, the private cluster performance is much higher than Amazon EC2 instances.

Fourier Transform (FT) Benchmark: The FT benchmark is to test for both computational and communication speed involving large data transfers. In the benchmark the performance difference between private, XLarge and medium instances are narrow.

The experiments conducted by Amedro *et al.* show that EC2 does not offer good performance for communication intensive applications, compared to local cluster. However, CPU intensive applications do not present significant performance hit. The study also concluded that when dealing with a complex application mixing communications and computations, it would be interesting to have a part on a cloud and another on a cluster depending on the application characteristics.

Evangelinos *et al.* employed the serial version of the NAS Parallel Benchmarks (NPB v.3.3) using the workstation class (W) and smallest of parallel classes (A) to test the computational performance on a wide set of model applications and kernels [Evangelinos08]. The results showed that the geometric mean of (BT, CG, FT, IS, LU, MG, SP, UA-excluding DC) as well as the value of the EP tested was between 2.2 and 2.4 times faster on the High-CPU instances.

## 2.2 Memory Bandwidth

With the CPU processing speeds increasing more quickly than computer memory speeds, the high performance computing systems will be especially limited in performance by memory bandwidth rather than by the computational performance of the CPU. The ratio

of CPU speed to memory speed is growing rapidly in high performance systems. The CPU speed of the fastest available microprocessor is increasing by 80% per year where as the memory speed is increasing by only 7% every year [McCalpin95B].

The STREAM benchmark is a benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels [McCalpin95A]. This benchmark is specifically used for measuring memory bandwidth of very large datasets such as in scientific computing. Both serial and MPI versions of the benchmark are available.

Evangelinos *et al.* tested the memory bandwidth of EC2 instance using the STREAM benchmark [Evangelinos08]. The results showed high bandwidth for the standard instance type. The High-CPU medium instance delivered bandwidth better than what one would expect from two cores sharing the same socket's pins to main memory.

### 2.3 Input/Output Performance

Input/Output (I/O) is very fundamental to HPC applications to store output for later analysis, to store the state of an application in case of failure, and to implement algorithms that process large amount of data. Typically, HPC applications have parallel file systems that greatly increase their scalability and capacity.

Interleaved or Random (IOR) is an I/O benchmark that is useful for characterizing the performance of parallel/cluster file systems. In particular, it can perform parallel reads and writes to/from either a single file, or multiple files, using MPIIO. The IOR software is used for benchmarking parallel file systems also using POSIX or HDF5 interfaces [Ghoshal11].

IOR benchmark leverages the scalability of MPI to easily and accurately calculate the aggregate bandwidth of unlimited number of client machines. In addition, IOR can utilize the POSIX, MPI-IO, and HDF5 I/O interfaces. The downside is that it is quite limited in its capabilities, focusing on reading and writing a file from beginning to end in a sequential manner [Ghoshal11].

Evangelinos *et al.* tested the I/O subsystem performance on the IOR benchmark in POSIX mode and tested large read and write requests on both the local /tmp disk and the remote home directory on standard small instance [Evangelinos08]. The results showed that there is an appreciable difference between the write and read performance of the standard and the High-CPU instances to/from local disk. In addition, the results showed that while the read performance from local disk appears to be close between the two instance types (standard and high CPU instance), most measurements were in the range of 800MB/s for the standard one.

Ghoshal *et al.* presented results on benchmarking the I/O performance over different clouds and HPC platforms to identify the major bottlenecks in the existing infrastructure

[Ghoshal11]. This work also compared the I/O performance using IOR benchmarks on two cloud platforms - Amazon and the Magellan cloud test bed. For evaluation purposes and in order to understand the effects of buffering caches, the study measured both buffered I/O and direct I/O.

In this thesis, we extend the previous research by conducting an empirical performance analysis of two public clouds of IaaS and PaaS types. Our methodology, results, and an analysis of results are presented in the subsequent chapters.

## Chapter 3

### RESEARCH METHODOLOGY

This thesis compares benchmark results on cluster of nodes for two public cloud-computing platforms that span both Infrastructure as a Service (IaaS) and Platform as a Service (PaaS). The Amazon Web Service EC2 and Windows Azure cloud computing platforms were used for this purpose. The methodology involved implementation of existing benchmarks STREAM, IOR and NAS Parallel Benchmarks (NPB) on both the cloud platforms with variations such as number of nodes (1, 2, 4, 6, and 8), small and medium instance types in the cluster that have comparable hardware and software specifications. At the conclusion of the literature review, we decided to include a new EC2 medium instance type (m1.medium) in the study.

#### 3.1 STREAM Benchmark

As indicated in the literature review, STREAM benchmark primarily measures the sustainable memory bandwidth. MPI version of STREAM is run on EC2's Standard small instance (m1.small), High-CPU medium instance (c1.medium) and standard medium (m1.medium) instance to measure their memory bandwidths. In each case, the number of EC2 instances is varied as 1, 2, 4, 6, and 8 nodes. It is also run on small and medium instances of Windows Azure platform. The benchmark comes in several

versions but MPI version is executed since it provides the parallel processing capabilities required in a cluster. STREAM is run on each core of a node using the MPI programming paradigm [McCalpin95A].

The sustained memory bandwidth is measured for four computational kernels:

- Copy: Copy measures transfer rates in the absence of arithmetic.

$$a(i) = b(i), \text{ where } a \text{ and } b \text{ are arrays}$$

- Scale: Adds a simple arithmetic operation

$$a(i) = q * b(i), \text{ where } a \text{ and } b \text{ are arrays and } q \text{ is a constant.}$$

- Add: Adds a third operand to allow multiple load/store ports on vector machines to be tested.

$$a(i) = b(i) + c(i), \text{ where } a, b, \text{ and } c \text{ are arrays.}$$

- Triad: Allows chained/overlapped/fused multiply/add operations.

$$a(i) = b(i) + q * c(i), \text{ where } a, b, \text{ and } c \text{ are arrays and } q \text{ is a constant.}$$

The STREAM benchmark generally expects the array size to be at least four times the size of the sum of all the last-level caches or 1 million elements whichever is larger during execution [McCalpin95C].

For each vector kernel, a memory bandwidth rate, average time, minimum time, and maximum time are measured for each choice of thread count. On all modern systems, the rate of execution is determined by the access to memory rather than the peak FLOP rate (i.e., the clock rate). The size of the arrays,  $n$ , can be varied to get sensible timings

[McCalpin95A]. If  $n$  is very large then the program will be accessing main memory. If it is small enough, then data may fit into cache, leading to an increased bandwidth for multiple iterations.

### 3.2 Interleaved Or Random Benchmark

While there has been research done for I/O performance in general, there is a limited understanding of its behavior in the cloud environments particularly from a cluster perspective. Understanding the I/O performance is critical to understanding the performance of HPC applications in the cloud [Ghoshal11]. Hence, we have chosen in this thesis to evaluate the I/O performance of Amazon EC2's standard small instance, High-CPU medium instance and standard medium instance with respect the handling of IOR benchmarks and compare it to Windows Azure's small and medium instances. The number of instances is varied as 1, 2, 4, 6, and 8 nodes and I/O performance is measured on both EC2 and Azure platforms. The amount of CPU that is allocated to a particular instance is expressed in terms of these EC2 Compute Units.

### 3.3 NAS Parallel Benchmarks

NAS Parallel Benchmarks (NPB3.3) measure the communication and computational performance of parallel machines such as clusters of nodes. All the NAS benchmarks communicate via MPI [Wong99], and the NPB suite consists of EP, CG and FT benchmarks and several others. For the purposes of this thesis, these benchmarks were



run on Amazon EC2's High-CPU Medium Instance (c1.medium), standard small instance (m1.small) and standard medium instance (m1.medium) and Windows Azure's small and medium instance types. CG and FT benchmarks could only be run on number of nodes in powers of two. EP is run with 1, 2, 4, 6, and 8 nodes.

- EP: Embarrassingly Parallel benchmark is used to test the computational speed of the nodes.
- CG: Conjugate Gradient is used to test the communication performance.
- FT: Fourier Transform benchmark is used to test both the computational and communication performances involving large data transfers.

### 3.4 Amazon Web Service EC2 Platform

In order to accomplish the EC2 cluster provisioning, StarCluster [StarCluster12A, 12B], an open-source command line utility developed at MIT is installed on the local development machine. StarCluster is a cluster-computing toolkit capable of configuring, creating, managing and terminating the cluster of VMs on Amazon EC2 instances on demand [StarCluster12A]. It is released under LGPL license [StarCluster12A].

StarCluster is capable of enabling MPICH2 communication between the nodes in a cluster in addition to creating and submitting MPI jobs to the cluster. Amazon provides AWS Management Console to monitor the status of the instance nodes, Elastic Block Storage (EBS) volumes, creating AMIs and other several other features for the Amazon's cloud related utilities.

### 3.4.1 Master and Compute Nodes

An EC2 node is a VM that has a hardware configuration that includes local instance storage and memory. The hardware configurations are different depending on the type of instance. When a cluster is built in EC2 platform, a master node and several compute nodes are created. The cluster is provisioned with an Operating System, EBS Storage, Network File System (NFS), MPICH2 and necessary compilers to execute the benchmarks.

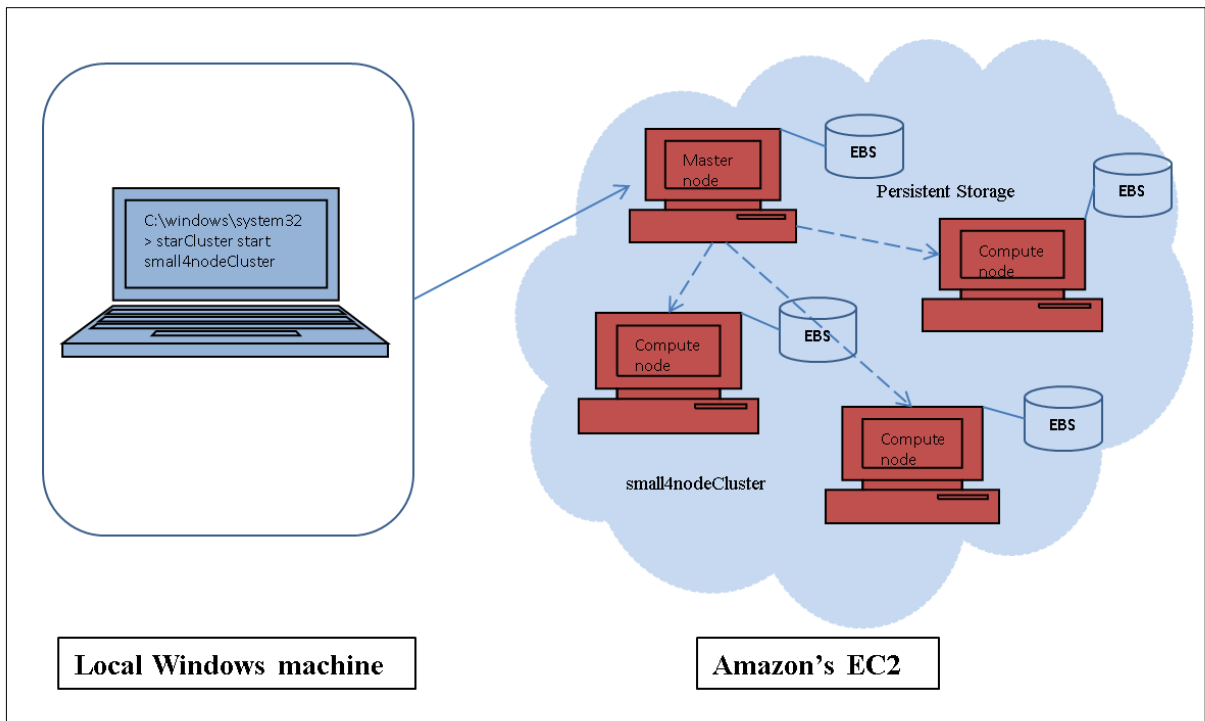


Figure 1: Connecting to EC2 Cluster From Client Machine Installed With StarCluster [StartCluster12A]

Each cluster in EC2 is configured with a master node and one or more compute nodes depending on the size of the cluster. The file and folder structure in each node is exactly

same. For the MPI jobs to be executed, MPI communication and password-less login are established between the master and compute nodes. Most of the functions including the submission of MPI jobs (benchmarks) for execution occur from the master node as shown in Figure 1. Once the execution begins, the master node coordinates the job execution with compute nodes in the cluster and uses the NFS shared EBS volume to store data. The cluster size is controlled using the StarCluster Configuration File during the cluster creation.

### 3.4.2 EBS Volume and Instance Storage

Each node in the cluster is attached with a default Elastic Block Store (EBS) volume in EC2. It persists regardless of the life of the instance. Persistent storage means data in the volume are not lost or deleted if the cluster is stopped. They range from 1 GB to 1 TB and can be mounted as devices to any EC2 instances. By default, Amazon attaches a 10 GiB (1GiB  $\approx$  1.074GB) EBS volume to each node of an instance. This volume is attached to the instance in addition to local instance storage for an instance. For example, a c1.medium instance comes with a default 10 GiB of EBS volume storage and a 350 GB of local instance storage. EBS volume provides highly available, highly reliable block storage. These are placed in a specific availability zone and can be attached to instances in the same region [AWS12B]. The local instance storage on the other hand is not persistent; it is ephemeral. Any data stored is deleted or removed automatically if the cluster is stopped. Termination of a cluster however has the same effect on the data in both EBS and Instance storage.

### 3.5 Microsoft Windows Azure Platform

Windows Azure is Microsoft's platform for cloud services. Currently it is offered with Platform-as-a-Service capabilities. Hence, it supports organizations that would like to run Windows applications [Marquand10]. Microsoft is continuously making important updates to this platform introducing new instances and Operating Systems support.

Windows Azure cloud platform provides compute instances and a shared storage account to store data from these instances.

#### 3.5.1 Web and Worker Roles

A compute node in the Windows Azure environment is a virtual server and is categorized into web roles and worker roles. A web role offers support for front-end portion of an application and consumes http requests via IIS [Marquand10]. A worker role is similar to a web role but does not take the http request. A cluster when built is configured according to the needs and the type of application being run on it. For running the HPC benchmarks there is no need to select a web role compute unit as there is no front end involved. So, the cluster is built with worker roles. Windows Azure loaded Windows HPC Server 2008 R2 as the OS on these compute nodes.

### 3.5.2 Head Node and Compute Node

A cluster in Windows Azure cloud platform always has a head node and one or more compute nodes. Both these node types are worker roles. Windows HPC Server 2008 R2 comes with HPC Pack and support for MPI which is MS-MPI (Microsoft's implementation of MPI). MPI is necessary for compute nodes to support the MPI jobs such as HPC benchmarks and to communicate with each other for parallel code processing. The head node passes on all the necessary parameters and instructions to the compute nodes to execute an MPI job. Once the job is complete, the results are sent back to the head node.

### 3.5.3 Windows Azure Storage:

Windows Azure's storage feature is accomplished with SQL Azure and Windows Azure storage account. The process of building the cluster allows the user to create an Azure Storage account for the cluster. This account is shared across all the compute nodes of a cluster. These components are illustrated in the Figure 2 below.

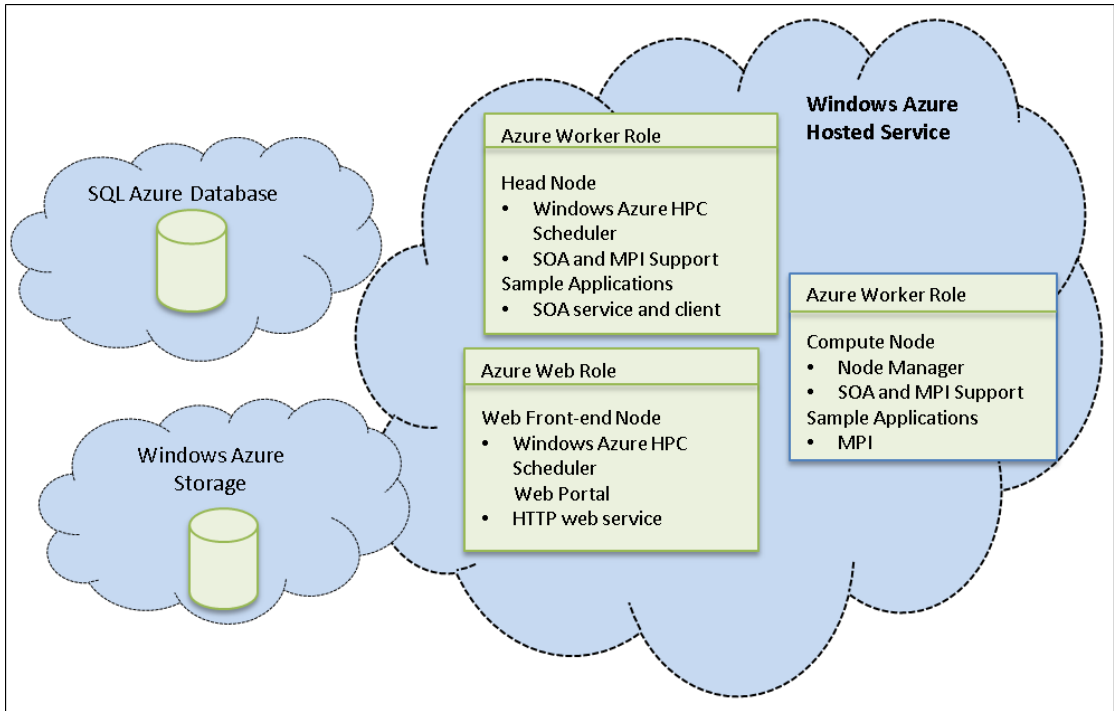


Figure 2: Microsoft Windows Azure Roles [MSDN12B]

## Chapter 4

### HARDWARE AND SOFTWARE SPECIFICATIONS

The benchmarks are executed in Linux and Windows environments installed with MPI implementations of C and Fortran compilers. These compilers are necessary for compiling and running MPI versions of the benchmarks. We describe below the Software and Hardware specifications used.

#### 4.1 Software Specifications

StarCluster Amazon Machine Image (AMI) is used to build a Cluster on EC2 loaded with Linux Ubuntu 11.10 operating system. This AMI enables several components necessary to run MPI jobs in a cluster. It is loaded with MPICC, MPIF77 and MPIF90 compilers. Since the benchmarks are written in C and Fortran languages, appropriate compilers are used.

On Windows Azure cluster Windows HPC Server 2008 R2 operating system is loaded. It is loaded with Microsoft implementations of MPICC, MPIF77 & MPIF90 compilers. These compilers run the windows binaries created for the benchmarks.

These windows binaries are created in a VM loaded with HPC Linux guest operating system on 64-bit Windows 7 Home Premium host machine. The VM is created via VirtualBox. HPC Linux OS comes with tools such as PToolsWin and x86\_64-w64-mingw32-gcc cross compiler.

Other software used to accomplish files transfer between the guest OS and the host OS is WinSCP. WinSCP is also used to connect and transfer files between the cloud platforms and local windows machine. Puttygen is employed to create private key used to connect to master node on EC2.

## 4.2 Benchmarks

The URLs used for downloading the MPI versions of STREAM, IOR and NPI benchmarks are available in Appendix A.

### STREAM Benchmark:

STREAM is an HPC benchmark that measures the sustainable memory bandwidth and is written in C and Fortran languages for single and multi-processors.



Interleaved or Random Benchmark:

IOR is an HPC benchmark that measures the input/output performance of HPC systems and is written in Fortran.

NAS Parallel Benchmarks:

NAS Parallel benchmarks are HPC benchmarks written in Fortran that measure computation and communication performance of the HPC systems.

#### 4.3 Hardware Specifications

The hardware on EC2 instance types includes a RAM of 1.7 GB on Standard small and High-CPU Medium instances. The EBS volume (persistent storage) is 10 GiB on all the three instance types. The I/O performance is Moderate on all three EC2 instance types. The differences between these three instance types are highlighted in Table 2 below.

			EC2 High-CPU Medium	Windows Azure Small	Windows Azure Medium
API Name	m1.small	m1.medium	c1.medium	-	-
Processor <sup>1</sup>	1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)	2 EC2 Compute Unit (1 virtual core with 2 EC2 Compute Unit)	5 EC2 Compute Unit (2 virtual core with 2.5 EC2 Compute Unit)	Quad-Core AMD Opteron (tm) Processor 2372 EE 2.10 GHz	AMD Opteron (tm) Processor 4171 HE 2.09 GHz (2 cores)
Local Instance Storage	160 GB	410 GB	350 GB	225 GB	490 GB
RAM	1.7 GB	3.75 GB	1.7 GB	1.75 GB	3.50 GB

Table 2: Hardware Specifications of EC2 and Azure Instance Types

---

<sup>1</sup> EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

The EC2 High-CPU medium instance (c1.medium) has proportionally more CPU resources (see processor details above for c1.medium and m1.medium) than memory (RAM) and is well suited for compute-intensive applications such as NPB benchmarks, whereas the EC2 standard instance (m1.medium) is well suited for most applications. On the other hand, Azure provides only a single medium instance type, which seems to be comparable with the EC2 standard instance m1.medium based on the RAM configuration. The experiments hence used the same Windows Azure medium instance wherever applicable. Both the Windows Azure instances have Windows HPC Server 2008 R2 (64-bit) operating System.

## Chapter 5

### SETTING UP, CONFIGURING AND BENCHMARKING EC2

The benchmarking of EC2 involves several carefully executed complex steps including building the cluster. Each step is explained in the subsections below. Chapter 6 details the procedure for Windows Azure.

#### 5.1 Pre-requisites on the Local Windows Development Machine

Before getting started to executing the benchmarks in the EC2 cloud platform the local Windows Development machine is installed with some pre-requisite software StarCluster. Python 2.7 is installed on the local machine first as it is required for StarCluster installation on Windows platform. The installer is available at [www.python.org](http://www.python.org). Python 2.7 is again dependent on Setuptools 0.6rc11 and Pycrypto 2.3 to be installed first [StartCluster12E].

#### 5.2 Installing StarCluster to Build the Cluster

After the necessary pre-requisite software installation is complete, StarCluster is installed using the following command in the command window.

```
C:\> easy_install StarCluster
```

### 5.3 Edit StarCluster Configuration File

StarCluster uses a configuration file that has all the necessary information and instructs StarCluster to create and start a new or existing cluster. It has sections and fields within the sections for EC2's AWS Subscription, Private Key, instance type for master node and compute node, Amazon Machine Image (AMI), NFS, EC2 region. A plugins section is added to the configuration file to enable MPICH2 on all the nodes in the cluster. MPI communication is an important component for the nodes to communicate with each other in a cluster. The CLUSTER\_SIZE parameter is edited to build a cluster of size (1, 2, 4, 6, and 8 nodes) for an instance type before benchmarking. An example StarCluster configuration file used for creating an eight-node m1.small cluster is available in Appendix B under StarCluster Configuration. Some of the important sections and fields of the configuration file are discussed below.

#### 5.3.1 Amazon Machine Image

An AMI in EC2 is a pre-configured Operating System and virtual application software that is used for building VMs in EC2 for parallel and distributed computing. Several public AMIs that already have the necessary software stacked up are available to be used. StarCluster uses some public AMIs that are both 32-bit and 64-bit. StarCluster AMI, ami-999d49f0 (x86\_64) is used for m1.small, m1.medium and c1.medium instance types

for cluster creation for node sizes 1, 2, 4, 6, 8. This AMI constitutes Ubuntu 11.10 operating system [StarCluster12C] and is loaded with necessary compilers as part of compatible MPICH2.

### 5.3.2 Plugins - Message Passing Interface

MPICH2 is a freely available high-performance and portable implementation of MPI (Message Passing Interface) [MPICH12]. MPI is a standard communication method used on distributed computing systems including clusters. StarCluster Configuration File has a PLUGIN section that configures MPICH2 on each node of an EC2 cluster.

MPICH2 configures and installs the mpicc, mpif77 and mpif90 compilers on all the nodes in the cluster. Mpicc compiler is for benchmarks written in C language and mpif77 and mpif90 compile the benchmarks in Fortran language. These compilers compile the STREAM, IOR and NPB benchmarks in MPI mode. MPICH2 also installs mpiexec, which is a command to run an executable created from mpicc compilation in a distributed computer network or cluster.

### 5.3.3 Scaling

Although EC2 provides auto-scaling features, for the purposes of this thesis, the nodes are incremented using the StarCluster Configuration File. CLUSTER\_SIZE parameter is

assigned the values of 1, 2, 4, 6, and 8 to build the appropriate cluster size before executing the benchmarks. Example: CLUSTER\_SIZE = 4

#### 5.4 Starting the Cluster with MPICH2

Following is the command used in the command prompt to start a new EC2 cluster with small instance type nodes:

```
C:\>starcluster start m1.small-AMI-cluster
```

This command creates and provisions the cluster with the configuration specified in the configuration file created in section 5.3 including NFS sharing across all nodes. In addition, the cluster is configured with password-less login so that nodes can communicate without any login issues. An example of successful start of an eight-node cluster is shown in Appendix D under EC2 screenshots.

##### 5.4.1 AWS Management Console

Once the cluster is created and is ready, the user can logon to Amazon AWS Management Console to ensure the nodes are in fact in 'Running' State as shown in Appendix C under EC2 screenshots. Other states of a cluster in EC2 are 'Stopped' and 'Terminated'.

Amazon AWS Subscription is required to monitor the instances on AWS Management

Console. Issuing appropriate StarCluster command from the local machine can restart a stopped cluster.

Each node in the cluster is assigned a unique name, Public DNS and Instance ID. This public DNS and the private key created using Puttygen are used to connect and login to a particular node.

### 5.5 Transfer the Benchmark Files to the Cluster

Secure Shell (SSH) tool WinSCP is installed on the local machine and is used to connect to the master node of the cluster using the private key to transfer the necessary benchmark files before execution. Example screenshots of file transfer to the master node of m1.medium instance using WinSCP are shown in Appendix D under EC2 screenshots. After successful connection, the benchmark files are transferred from the local system to /home/ec2-user of the master node.

#### 5.5.1 Network File System

NFS is a protocol used by UNIX/Linux computer systems to share the disk space in a cluster/network. When provisioning the EC2 cluster it is important to enable the disk sharing across all the nodes. StarCluster by default configures /home folder of the master node and NFS shares it with other nodes in the cluster [StarCluster12D]. Any benchmark related files copied or transferred to /home/ec2-user on the master node is copied



instantly to all the nodes in the cluster in the directory structure automatically since they are NFS shared. This is confirmed by connecting and logging into each node using PuTTY.

## 5.6 Execute Benchmarks on the Master Node

Because of NFS sharing, the benchmark files are copied to /home/ec2-user folder on each node of the cluster from the master node automatically. It is necessary that the benchmark files are available at the same location on each node so they are executed in parallel. Sections 5.6.1, 5.6.2, and 5.6.3 detail the execution procedures used for STREAM, IOR and NPB benchmarks.

### 5.6.1 STREAM Benchmark

The benchmark consists of two files, stream\_mpi.c and mysecond.c. These files are transferred to the master node's /home/ec2-user folder. PuTTY is used to connect and login as root into the master node with authentication using a private key. The following command is executed to compile the stream\_mpi.c file:

```
root@master:/home/ec2-user/STREAM-MPI# mpicc -DPARALLEL_MPI
-O3 -o stream_mpi stream_mpi.c
```

This command builds a UNIX/Linux executable file `stream_mpi` that can be run in a parallel computing environment. Following command is then executed to run the executable on all the nodes of the cluster. This example shows the execution on four-node of `c1.medium` instance type. This is ensured by specifying the name of the nodes in the `-host` argument of `mpiexec` command:

```
root@master:/home/ec2-user/STREAM-MPI# mpiexec -host
master, node001, node002, node003 ./stream_mpi >
output/c1.m_n4.1.txt
```

The benchmark execution is now complete and output is redirected to a text file in a folder named `Output`. Before the execution of the benchmark, the configuration file's `CLUSTER_SIZE` parameter is updated to 1, 2, 4, 6, and 8 to build the cluster of that size. In addition, `MASTER_INSTANCE_TYPE` and `NODE_INSTANCE_TYPE` parameters are updated to `m1.small`, `c1.medium` `m1.medium` instance types as necessary. The output for all the executions are redirected to text files for later analysis. Sample result of the execution is copied to Appendix C. The analysis of the STREAM benchmark is in chapter 7.

## 5.6.2 Interleaved Or Random Benchmark

Execution of the IOR benchmark involved several steps. The first step is to ensure the cluster of required number of nodes is in place for an instance type. Instance types used

are m1.small, c1.medium and m1.medium. Number of nodes used on each of these instance types is 1, 2, 4, 6, and 8. The necessary benchmark files are transferred to the master node of the cluster. NFS shares them with all the nodes in the cluster automatically.

One of the important steps for building the IOR executable in UNIX/Linux environment is the `make` command. The sample output of this command when compiled on a four-node cluster of c1.medium instance type is copied to Appendix B under Benchmarks Commands. This executable is created using the POSIX interface.

```
root@master:/mnt/ec2-user/IOR/src/C# make
```

Following is a sample command that is run to execute the IOR executable in parallel on a four-node cluster of c1.medium instance type:

```
root@master:/mnt/ec2-user/IOR/src/C# mpiexec -host master,  
node001, node002, node003 ./IOR -b 1g -t 4m >  
output/c1.m_n4.1.txt.
```

Buffering plays a very important part in IOR benchmarking. The first time when the benchmark is executed, the data from the testfile (1 GB) is buffered and hence the results for Read is higher.

The benchmark execution is now complete and output is redirected to a text file in a folder named Output. Before the execution of the benchmark, the configuration file's CLUSTER\_SIZE parameter is updated to 1, 2, 4, 6, and 8 to build the cluster of that size. In addition, MASTER\_INSTANCE\_TYPE and NODE\_INSTANCE\_TYPE parameters are updated to m1.small, c1.medium m1.medium instance types as necessary. The output from each execution is redirected to text files for later analysis. Sample result of the execution is copied to Appendix C under Sample Results. The analysis of the IOR benchmark is in chapter 7.

### 5.6.3 NAS Parallel Benchmarks

We consider three NPB 3.3 benchmarks: Conjugate Gradient (CG), Fourier Transform (FT), and Embarrassingly Parallel (EP) benchmark.

#### 5.6.3.1 Conjugate Gradient Benchmark

CG is written in Fortran and comes in different classes. CG is compiled for Class A on the master node of the cluster. Following is a sample command that is run to build UNIX/Linux CG executable for four-node on c1.medium instance type.

```
root@master:/home/ec2-user/NPB3.3/NPB3.3-MPI# make cg
NPROCS=4 CLASS=A
```

The output for this command is shown in Appendix B under Benchmark Commands.

This command creates an executable `cg.A.4` that can be successfully run on a cluster of four nodes. Unlike other benchmarks, NPB benchmarks must be compiled specifically to build executable for a cluster size. For example, when the executable file `cg.A.4` is run on a three-node cluster, a run time error is received indicating the number of processes (nodes) is not matching with the executable. So, in order to build an executable file for a six-node cluster the above command is run with `NPROCS=6` in command line argument that created `cg.A.6`.

Once the compilations are complete, all these are executed using `mpiexec` command and the results are redirected to text files for later analysis. Below is the sample command that is executed on a four-node cluster of `c1.medium` cluster for class A.

```
root@master:/home/ec2-user/NPB3.3-MPI# mpiexec -host
master, node001, node002, node003 bin/cg.A.4 >
output/cg.A.4_3.txt
```

The configuration file's `CLUSTER_SIZE` parameter is updated to 1, 2, 4, and 8 to build the cluster of that size before the above steps. CG can only be compiled in cluster size that is a power of two. In addition `MASTER_INSTANCE_TYPE` and `NODE_INSTANCE_TYPE` parameters are updated to `m1.small`, `c1.medium` and

m1.medium instance types as necessary. Sample result of the execution is copied to Appendix C. The analysis of the CG benchmark is in chapter 7.

### 5.6.3.2 Fourier Transform Benchmark

FT is written in Fortran language and comes in different classes. FT is compiled for Class A on the master node of the cluster. A sample `make` command output that is run to build UNIX/Linux CG executable for four-node on c1.medium instance type is shown in Appendix B under Benchmark Commands.

```
root@master:/home/ec2-user/NPB3.3-MPI# make FT NPROCS=4  
CLASS=A
```

This command creates an executable `ft.A.4` that can be successfully run on a cluster of four nodes. Unlike other benchmarks, NPB benchmarks must be compiled specifically to build executable for a cluster size. For example, when the executable file `ft.A.4` is run on a three-node cluster, a run time error is received indicating the number of processes (nodes) is not matching with the executable. So, in order to build an executable file for an eight-node cluster the above command is run with `NPROCS=8` in command line argument that created `ft.A.8`.

Once the compilations are complete, all these are executed using `mpiexec` command and the results are redirected to text files for later analysis. Below is the sample command that is executed on a four-node cluster of `c1.medium` cluster for class A:

```
root@master:/home/ec2-user/NPB3.3-MPI# mpiexec -host
master, node001, node002, node003 bin/ft.A.4 >
output/ft.A.4_3.txt
```

The configuration file's `CLUSTER_SIZE` parameter is updated to 1, 2, 4, and 8 to build the cluster of that size before the above steps. FT can only be compiled in cluster size that is a power of two. In addition, `MASTER_INSTANCE_TYPE` and `NODE_INSTANCE_TYPE` parameters are updated to `m1.small`, `c1.medium` and `m1.medium` instance types as necessary. Sample result of the execution is copied to Appendix C. The analysis of the CG benchmark is in chapter 7.

### 5.6.3.3 Embarrassingly Parallel Benchmark

EP is written in Fortran language and comes in different classes. EP is compiled for Class A on the master node of the cluster. A sample output of the `make` command that is run to build UNIX/Linux EP executable for four-node on `c1.medium` instance type is shown in Appendix B under Benchmark Commands.

```
root@master:/home/ec2-user/NPB3.3-MPI# make EP NPROCS=4  
CLASS=A
```

This command creates an executable ep.A.4 that can be successfully run on a cluster of four nodes. Unlike other benchmarks, NPB benchmarks must be compiled specifically to build executable for a cluster size. For example, when the executable file ep.A.4 is run on a three-node cluster, a run time error is received indicating the number of processes (nodes) is not matching with the executable. So, in order to build an executable file for an eight-node cluster the above command is run with NPROCS=8 in command line argument that created ep.A.8.

Once the compilations are complete, all these are executed using `mpiexec` command and the results are redirected to text files for later analysis. Below is the sample command that is executed on a four-node cluster of `cl.medium` cluster for class A:

```
root@master:/home/ec2-user/NPB3.3-MPI# mpiexec -host  
master, node001, node002, node003 bin/ep.A.4 >  
output/ep.A.4_3.txt
```

The configuration file's `CLUSTER_SIZE` parameter is updated to 1, 2, 4, 6, and 8 to build the cluster of that size before the above steps. In addition, `MASTER_INSTANCE_TYPE` and `NODE_INSTANCE_TYPE` parameters are updated to



m1.small, c1.medium m1.medium instance types as necessary. Sample result of the execution is copied to Appendix C. The analysis of the EP benchmark is in chapter 7.

#### 5.6.3.4 Stop/Terminate the cluster

SSH tool WinSCP is used to transfer the result files on to the local development machine for analysis. Secure connection is established with the master node for transferring the files. After the output text files are transferred back to the local windows development machine the cluster is stopped. A stopped cluster can be restarted at any time as necessary. An example screenshot of stopping an eight-node cluster is available in Appendix D under EC2 screenshots.

## Chapter 6

### SETTING UP, CONFIGURING AND BENCHMARKING WINDOWS AZURE

The benchmarking of Windows Azure consists of several steps, which are explained in the subsections below.

#### 6.1 Building Windows Binaries

The prerequisite software for building the windows binaries on the local Windows machine is VirtualBox and the HPC Linux guest OS on the VM is created with VirtualBox.

Since the executable files created out of Amazon EC2 processes are for UNIX/Linux, they could not be directly executed on the Windows Azure platform. Therefore, windows binaries had to be built using a cross-compiler. In order to achieve this task VirtualBox is installed on the local Windows machine (host machine). VirtualBox created a VM (guest machine). This VM had an IP address that is later used to connect to it using WinSCP for file transfers. HPC Linux Operating System, a Linux distribution is installed on this VM. The link to download HPC Linux is available in Appendix A. HPC Linux comes with components called PToolsWin and x86\_64-w64-mingw32-gcc cross-compiler. When the windows binaries (.exe) are created using PToolsWin and the cross-compiler for each of the benchmarks, some dll files are also created in the process. These dll files along with

the benchmark executable (.exe) are zipped and the entire package is transferred back to the Windows host machine. The entire package is necessary for the benchmark to be executed on the Windows platform. WinSCP is used to transfer the files back and forth between the guest OS and the host OS.

## 6.2 Pre-requisites on the Local Windows Development Machine

- Windows Azure HPC Scheduler SDK 64-bit
- Windows Azure Subscription
- Windows PowerShell
- Microsoft Silverlight

## 6.3 Deploy Windows Azure HPC Scheduler via PowerShell

Windows Azure HPC Scheduler [MSDN12A] includes the components that enable the user to launch and manage HPC applications in Windows Azure platform. The scheduler supports submitting and managing HPC MPI jobs and processes, and hence works with MPI versions of STREAM, IOR and NPB benchmarks. The HPC Scheduler SDK package (version 1.6) is available for download from <http://www.microsoft.com/en-us/download/details.aspx?id=28015>. Once the Windows Azure HPC Scheduler is deployed, it creates a Hosted Service containing a cluster of VMs (nodes) in the Windows Azure platform [Paratools12A] containing a head node and

several compute nodes. The size of the cluster depended on configurations in the configuration and definition files.

The screenshot of a successful creation of an eight-node cluster is shown in Appendix E under Windows Azure Screenshots. PowerShell also lets the user create a certificate file (.cer) that is uploaded to the Management Certificates in Windows Azure Management Portal. In addition, it also creates the SQL Azure persistent database for storage. The process creates Azure storage along with the Hosted Service that is later used for benchmarking synchronizing in a cluster.

### 6.3.1 Service Configuration and Service Definition Files

During the Windows Azure HPC scheduler deployment Windows PowerShell uses service configuration (.cscfg) and service definition (.csdef) files to create a cluster. The service definition file defines all the roles in the cluster such as HeadNode and ComputeNode. It also defines the types of instance needed for these roles. Service configuration file on the other hand defines the number of instances needed for both head node and compute nodes.

After the HPC Scheduler is deployed successfully, the Windows Azure Management Portal appears with all the nodes in the ready state as shown in Appendix E under Windows Azure Screenshots.

## 6.4 Connect to HeadNode on the Cluster on Windows Azure

Remote Desktop Protocol (RDP) connection utility is used to connect to the HeadNode of the cluster. As shown in the screenshot of Windows Azure Management Portal in Appendix E, the 'connect' button is used to connect to the head node. It starts a remote connection with the node after a secure login. The Windows Azure Screenshots in Appendix E shows an example RDP connection to a head node.

After a successful connection, the desktop of the head node as shown under Windows Azure Screenshots in Appendix E is displayed on the local machine. From the Windows local machine, the benchmark zipped package is copied over to the head node of the cluster.

## 6.5 Windows Azure Firewall Configuration for MPI Communication

This is a very important step in making the benchmark ready for execution in the cluster. It involves running some commands using the PowerShell in a sequence on the HeadNode that unzipped and uploaded the benchmark files to the Azure storage and made them available for all the nodes in the cluster to execute. In order to run MPI jobs in a cluster it is important to open the firewall between the compute nodes [Paratools12B]. First step is run 'hpcpack create' command that created a package in a compressed format [TechNet12A]. The command is as shown below:

```
PS C:\approot> hpcpack create C:\approot\benchmarks.zip
C:\approot\benchmarks
```

The next step is to run the ‘hpcpack upload’ command. It uploads the compressed package to the Azure storage. As mentioned above the dependent dll files are also part of the package.

The next step is to synchronize the package from the Azure storage to all the nodes in the cluster. This is done by running the following command:

```
PS C:\approot>clusrun /nodegroup:computenode hpcsync
```

Finally the following ‘clusrun’ command is run. This command registered the benchmark binary to all nodes of the cluster and opens up the firewall for MPI communications. Below is the command run on a small instance eight-node cluster for IOR benchmark:

```
PS D:\Users\sinadmin> clusrun /nodegroup:ComputeNode
hpcfutil register IOR.exe
```

The output of the `clusrun` command is shown in Appendix B under Benchmark Commands.

The IOR.exe file is now successfully registered to on all the nodes so it can run on them in parallel. They all returned code 0 indicating success.

## 6.6 Create and Submit MPI Jobs for Executing Benchmarks

The HPC Job Manager utility is used to create, submit and monitor the MPI jobs from the head node. HPC Job Manager presents an easy to use User Interface (UI) and is part of HPC Pack 2008. The UI takes command lines as input to execute the benchmark. HPC Pack 2008 is part of Windows HPC Server 2008 R2 OS. It allows the user to monitor the progress of the jobs and categorizes them as failed, active, cancelled and finished. The status of all jobs that are successfully executed are automatically changed from 'active' to 'finished' state. The job is in active state for as long as it is running. The screenshot of the finished jobs in HPC Job Manager is shown in Appendix E under Windows Azure Screenshots.

### 6.6.1 STREAM Benchmark

STREAM is run from the HeadNode of the cluster. STREAM benchmark's executable file along with dependent dll files are synced to all the nodes before execution.

Following is an example command that is used to run the benchmark on a four-node cluster of small instance type in the HPC Job Manager UI.

```
mpiexec -np 4 C:\Resources\ Directory\  
c6b0e3213a6649099e59530e7834fb4b.  
ComputeNode.Microsoft.Hpc.Azure.LocalStorage.Application\be  
nchmarks\2012-05-29T232012.0000000Z\stream_mpi.exe
```

The argument `-np` is the number of processes (nodes) the benchmark is run on in parallel. The results of the execution are copied to a text file and later moved to the local Windows machine for analysis.

Service configuration and service definition files are used to control the size and the instance type. Thus, the benchmark is run on cluster sized with 1, 2, 4, 6, and 8 nodes for small and medium instance types as required. Sample result of the execution is copied to Appendix C. The analysis of the STREAM benchmark is in chapter 7.

### 6.6.2 Interleaved Or Random Benchmark

IOR is run from the HeadNode of the cluster. IOR benchmark's executable file along with dependent dll files are synced to all the nodes before execution. Following is an example command that is used to run the benchmark on a four-node cluster of small instance type in the HPC Job Manager UI.



```
mpiexec -np 4 C:\Resources\ Directory\  
c6b0e3213a6649099e59530e7834fb4b.  
ComputeNode.Microsoft.Hpc.Azure.LocalStorage.Application\be  
nchmarks\2012-05-29T232012.0000000Z\IOR.exe -b 1g -t 4m
```

The argument `-np` is the number of processes (nodes) the benchmark is run on in parallel. The results of the execution are copied to a text file and later moved to the local windows machine for analysis.

Service configuration and service definition file are used to control the size and the instance type. Thus, the benchmark is run on cluster sized with 1, 2, 4, 6, and 8 nodes for small and medium instance types as required. Sample result of the execution is copied to Appendix C. The analysis of the IOR benchmark is in chapter 7.

### 6.6.3 NAS Parallel Benchmarks

As with EC2, we consider three NPB 3.3 benchmarks to execute on Windows Azure platform: Conjugate Gradient (CG), Fourier Transform (FT), and Embarrassingly Parallel (EP) benchmark.

### 6.6.3.1 Conjugate Gradient Benchmark

NBP CG is run from the HeadNode of the cluster. CG benchmark's executable file along with dependent dll files are synced to all the nodes before execution. Following is an example command used to run the benchmark on a four-node cluster of small instance type in the HPC Job Manager UI.

```
mpiexec -np 4 C:\Resources\ Directory\  
c6b0e3213a6649099e59530e7834fb4b.  
ComputeNode.Microsoft.Hpc.Azure.LocalStorage.Application\be  
nchmarks\2012-05-29T232012.0000000Z\ CG.A.4.exe
```

The argument `-np` is the number of processes (nodes) the benchmark is run on in parallel. The results of the execution are copied to a text file and later moved to the local windows machine for analysis.

CG can only be compiled in cluster size that is a power of two. Service configuration and service definition file are used to control the size and the instance type. Thus, the benchmark is run on cluster sized with 1, 2, 4, and 8 nodes for small and medium instance types as required. Sample output of the `mpiexec` command is available in Appendix C. The analysis of the CG benchmark is in chapter 7.

### 6.6.3.2 Fourier Transform Benchmark

NBP FT is run from the HeadNode of the cluster. FT benchmark's executable file along with dependent dll files are synced to all the nodes before execution. Following is an example command used to run the benchmark on a four-node cluster of small instance type in the HPC 2008 R2 Job Manager UI:

```
mpiexec -np 4 C:\Resources\ Directory\  
c6b0e3213a6649099e59530e7834fb4b.  
ComputeNode.Microsoft.Hpc.Azure.LocalStorage.Application\be  
nchmarks\2012-05-29T232012.0000000Z\ FT.A.4.exe
```

The argument `-np` is the number of processes (nodes) the benchmark is run on in parallel. The results of the execution are copied to a text file and later moved to the local windows machine for analysis.

FT can only be compiled in cluster size that is a power of two. Service configuration and service definition file are used to control the size and the instance type. Thus, the benchmark is run on cluster sized with 1, 2, 4, and 8 nodes for small and medium instance types as required. Sample output of the `mpiexec` command is available in Appendix C. The analysis of the FT benchmark is in chapter 7.

### 6.6.3.3 Embarrassingly Parallel Benchmark

NPB EP is run from the HeadNode of the cluster. EP benchmark's executable file along with dependent dll files are synced to all the nodes before execution. Following is an example command used to run the benchmark on a four-node cluster of small instance type in the HPC Job Manager UI:

```
mpiexec -np 4 C:\Resources\ Directory\  
c6b0e3213a6649099e59530e7834fb4b.  
ComputeNode.Microsoft.Hpc.Azure.LocalStorage.Application\be  
nchmarks\2012-05-29T232012.0000000Z\ EP.A.4.exe
```

The argument `-np` is the number of processes (nodes) the benchmark is run on in parallel. The results of the execution are copied to a text file and later moved to the local windows machine for analysis.

Service configuration and service definition file are used to control the size and the instance type. Thus, the benchmark is run on cluster sized with 1, 2, 4, 6, and 8 nodes for small and medium instance types as required. Sample output of the `mpiexec` command is available in Appendix C. The analysis of the EP benchmark is in chapter 7.

## Chapter 7

### ANALYSIS OF RESULTS

The Microsoft Excel 2010 built-in function T-TEST was used and a statistical analysis of the results obtained was performed. The T-TEST function was used with two datasets as input, one for EC2 and one for Windows Azure that gave a p-value as output. The p-value is a number that is frequently used as a measure of comparison of two datasets. A p-value not exceeding 0.05 is considered as indication of statistically significant difference between the datasets and a p-value exceeding 0.05 indicating statistically insignificant difference. Each dataset for a benchmark was run with increasing number of nodes 1, 2, 4, 6 and 8. And each benchmark inherently ran several iterations before giving an average value. These average values were used to create the datasets for comparison for T-TEST.

#### 7.1 STREAM Benchmark

STREAM benchmark primarily measures the memory bandwidth. The MPI version of STREAM benchmark was used to run the benchmark on multiple processors. The sustained memory bandwidth was measured for four computational kernels: copy, scale, add, and triad. However, the analysis was mainly focused on stream triad as it performs a

combination of vector multiplication by a constant, and a sum on two source and one destination vectors thus allowing both scale and add operations.

The STREAM benchmark by default ran for 10 iterations and gave an output of average values when executed in a cluster for both small and medium instance types. For example in the below matrix a value of 4344.1400 for a two-node cluster on EC2 was the average of 10 iterations for Copy operation. The Add and Triad operations also ran for 10 iterations. An example of such an output is available in Appendix C.

#### 7.1.1 EC2 Standard Small Instance (m1.small) Versus Azure Small

The STREAM benchmark is mainly used to analyze the memory bandwidth of EC2 and Azure. Hence, the focus is mainly on how the trends of average megabytes per second (MB/s) vary between the two public clouds rather than the average time for specific operations, which seem to be inconsistent most of the time.

The average MB/s for the copy operation as shown in Figure 3 was higher for the EC2 small instance even as the number of nodes was varied as 1, 2, 4, 6 and 8. Though the average time taken for this operation seemed to be slightly inconsistent, the Azure small instance took more time than the EC2 instance did.

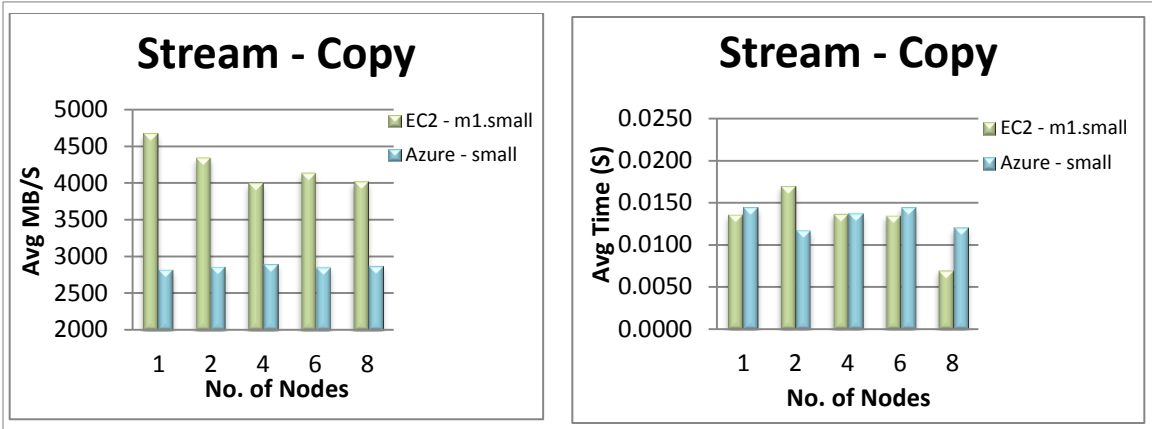


Figure 3: STREAM Copy - m1.small and Azure Small Instance Average

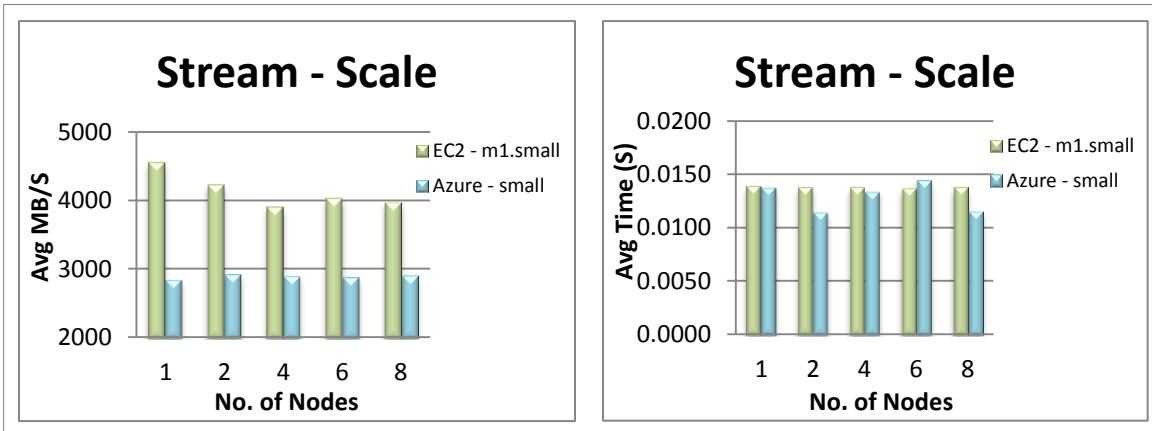


Figure 4: STREAM Scale - m1.small and Azure Small Instance Average

Scale adds a simple arithmetic operation. Similar to the copy operation, the average MB/s for EC2 was higher than the Azure instance for all the nodes as shown in Figure 4. Though the average time was inconsistent for most of the runs, the Azure instance took less time for most nodes.

# of Nodes	Copy		Scale		Add		Triad	
	EC2	Azure	EC2	Azure	EC2	Azure	EC2	Azure
1	4677.0648	2823.5417	4555.1579	2839.9931	4724.8672	2732.7379	4526.2273	2588.2466
2	4344.1400	2861.4200	4229.2200	2916.2400	4369.2400	2718.8900	4217.0000	2518.5000
4	4006.1600	2897.0100	3906.7900	2894.7800	3990.6200	2715.4600	3891.3600	2513.3200
6	4142.4000	2853.2100	4034.4600	2876.0200	4135.5500	2732.6400	4015.7100	2582.7300
8	4022.2300	2870.1800	3973.8700	2897.7400	4052.0600	2743.3300	3938.8500	2574.3300
P-value	0.000358437		0.000384909		0.00033495		0.000145284	

Table 3: STREAM - m1.small and Azure Small Instance Average MB/S

# of Nodes	Copy		Scale		Add		Triad	
	EC2	Azure	EC2	Azure	EC2	Azure	EC2	Azure
1	0.0136	0.0144	0.0139	0.0138	0.0236	0.0232	0.0240	0.0242
2	0.0170	0.0117	0.0138	0.0115	0.0203	0.0238	0.0207	0.0286
4	0.0137	0.0137	0.0138	0.0134	0.0236	0.0236	0.0241	0.0244
6	0.0135	0.0144	0.0137	0.0145	0.0169	0.0188	0.0240	0.0198
8	0.0070	0.0120	0.0138	0.0116	0.0270	0.0218	0.0307	0.0235
P-value	0.877795774		0.235662165		0.984262524		0.787531899	

Table 4: STREAM - m1.small and Azure Small Instance Average Time in S



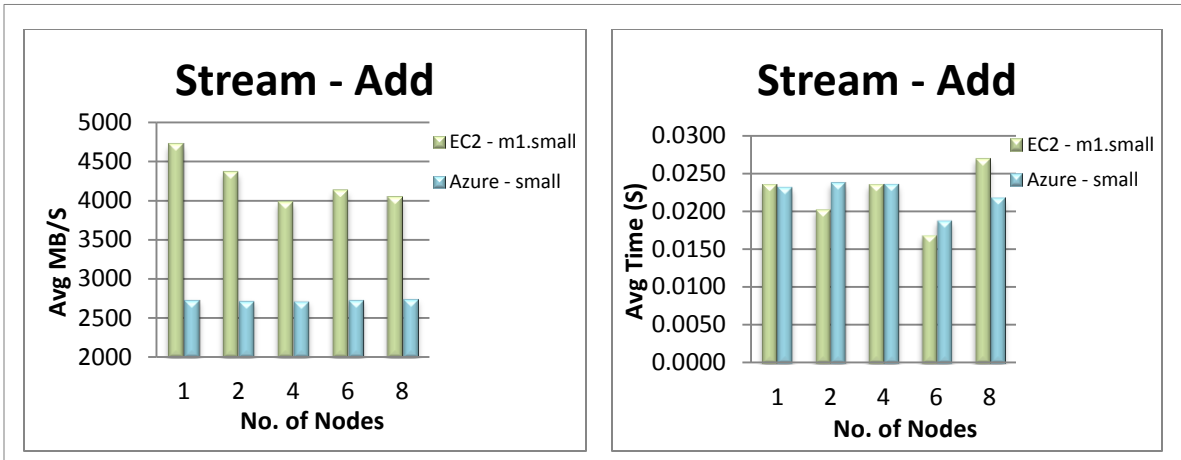


Figure 5: STREAM Add - m1.small and Azure Small Instance Average

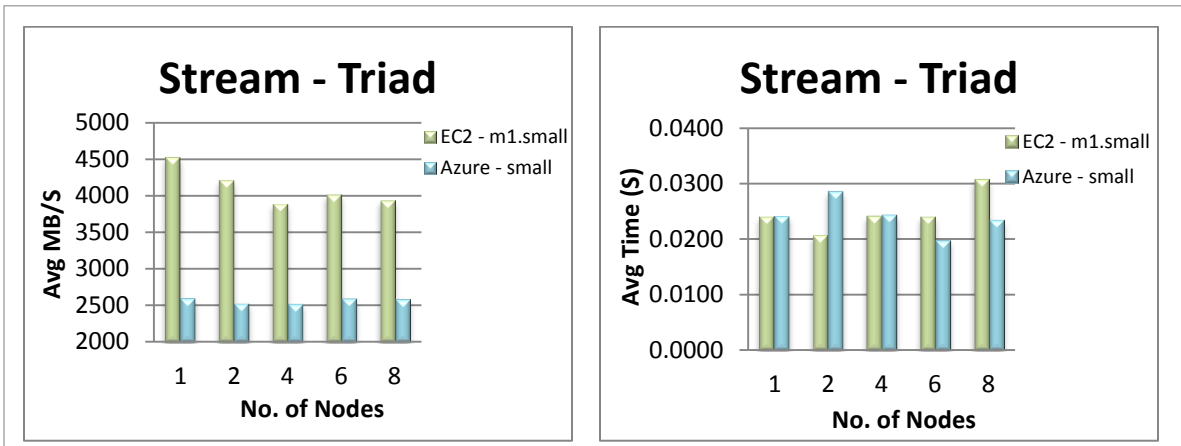


Figure 6: STREAM Triad - m1.small and Azure Small Instance Average

A statistical analysis was performed to determine if the difference in throughput between the two clouds was significant. The difference in throughput (avg MB/s) for all four operations copy, scale, add and triad between EC2 and Azure were found to be statistically significant with a p-value of 0.0001 for STREAM Triad (Table 3).

Graphically, the add and triad operations as shown in Figure 5 and Figure 6 also indicated that the average throughput for EC2 were higher than Azure for small instance types.

The throughput in EC2 was much higher than Windows Azure cloud for the small instance

type. This might be because of the different processors provided in EC2's small instance and Azure's small instance (section 4.3).

The difference in the average time taken by both EC2 and Azure for small instance type was found to be statistically insignificant with a p-value of 0.79 for STREAM Triad for the small instance type (Table 4).

#### 7.1.2 EC2 Standard Medium Instance (m1.medium) Versus Azure Medium

When the STREAM benchmark was run on the medium instance type, EC2 m1.medium instance type provided higher throughput (MB/s) for all the four computational kernels while the number of nodes were being varied as 1, 2, 4, 6, and 8. This is shown in Figure 7, Figure 8, Figure 9 and Figure 10 for copy, scale, add and triad operations respectively below. In addition, the average time taken for each of the operations by EC2 medium instance was much less compared to the Azure medium instance. This might also be because EC2's m1.medium instance has a processor of two EC2 compute units (equivalent to CPU capacity of 1.0-1.2 GHz 2007 Opteron or Xeon processor) as opposed to the quad-core AMD Opteron processor of Azure medium instance.

The RAM size plays a key role for measuring the memory bandwidth using STREAM benchmark. Evidently, m1.medium instance type of EC2 outperformed Azure medium instance that has an equivalent hardware configuration of 3.75 GB.

# of Nodes	Copy		Scale		Add		Triad	
	EC2	Azure	EC2	Azure	EC2	Azure	EC2	Azure
1	4019.5780	3280.6003	3854.0625	3266.85	3975.1726	2905.03	3862.2325	2680.64
2	4008.3000	3253.7200	3844.3500	3275.60	4039.0700	2912.85	3900.6300	2686.60
4	3988.0100	3395.3300	3892.5000	3405.06	3965.9000	3012.19	3873.1200	2780.18
6	4090.2200	3396.2000	3986.4100	3421.25	4083.1400	3017.80	3942.8500	2789.75
8	4073.4800	3364.6700	3945.0800	3378.15	4047.3700	2969.25	3947.1000	2741.68
P-value	2.62036E-07		1.50158E-06		9.53245E-10		4.57339E-10	

Table 5: STREAM - m1.medium and Azure Medium Instance Average MB/S

# of Nodes	Copy		Scale		Add		Triad	
	EC2	Azure	EC2	Azure	EC2	Azure	EC2	Azure
1	0.0082	0.0109	0.0085	0.0119	0.0176	0.0226	0.0179	0.0249
2	0.0084	0.0135	0.0112	0.0128	0.0150	0.0215	0.0181	0.0207
4	0.0113	0.0129	0.0087	0.0108	0.0178	0.0186	0.0156	0.0218
6	0.0086	0.0106	0.0116	0.0120	0.0157	0.0215	0.0158	0.0238
8	0.0107	0.0116	0.0082	0.0099	0.0152	0.0167	0.0153	0.0181
P-value	0.021299775		0.075432341		0.019157137		0.007508744	

Table 6: STREAM - m1.medium and Azure Medium Instance Average Time in S

The statistical analysis performed on the datasets for average MB/s and Average time showed that the difference between the two instance types were statistically significant for all the four operations with a p-value of 4.5E-10 for the average MB/s of STREAM Triad (Table 5) and 0.007 for the average time of STREAM Triad (Table 6).

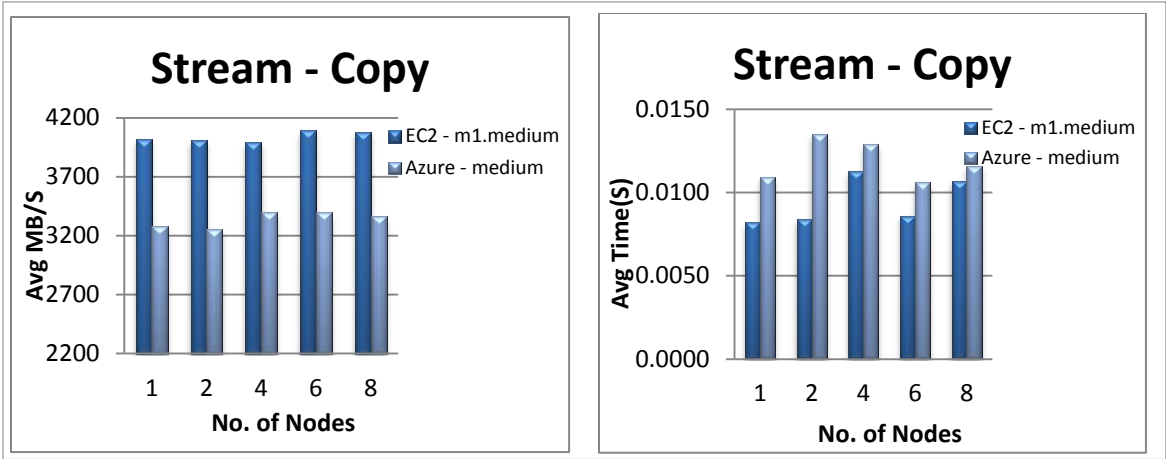


Figure 7: STREAM Copy - m1.medium and Azure Medium Instance

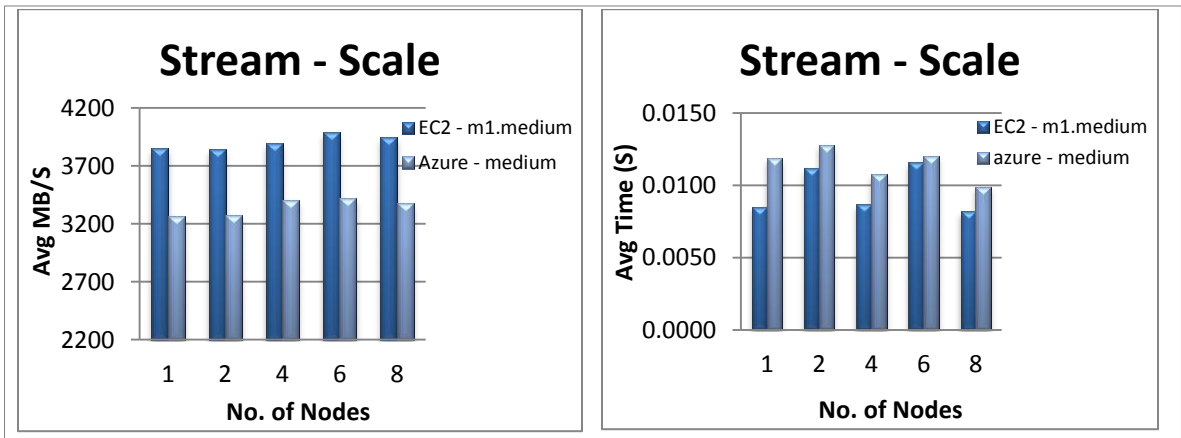


Figure 8: STREAM Scale - m1.medium and Azure Medium Instance

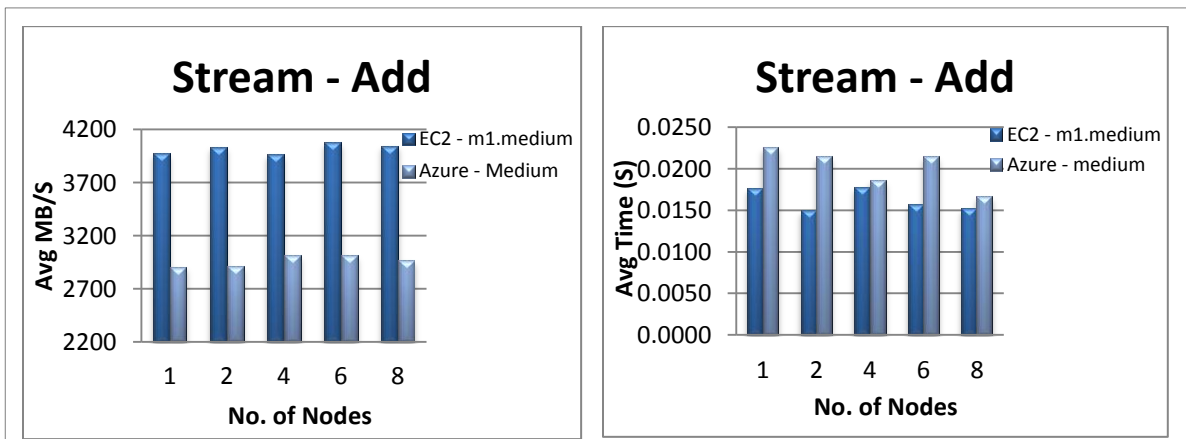


Figure 9: STREAM Add- m1.medium and Azure Medium Instance

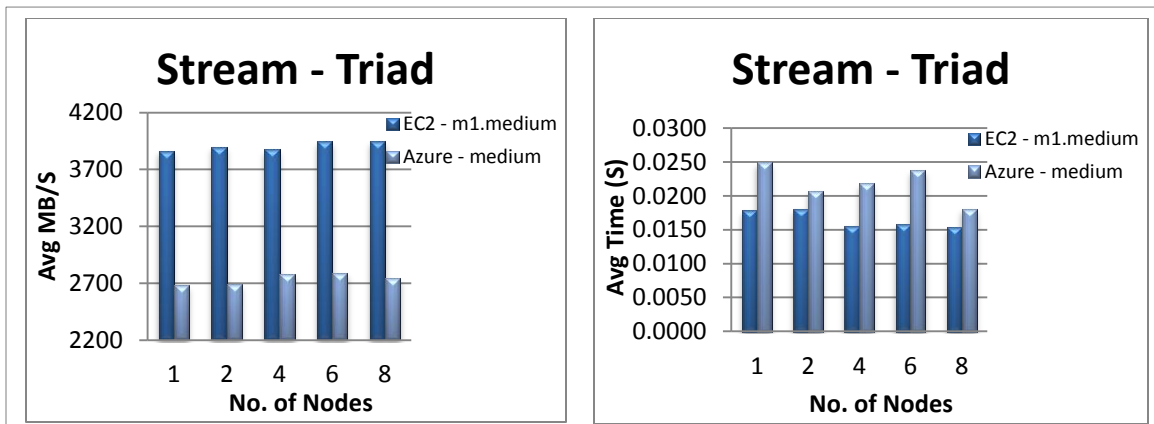


Figure 10: STREAM Triad - m1.medium and Azure Medium Instance

### 7.1.3 EC2 High-CPU Medium instance (c1.medium) Versus Azure Medium

The STREAM benchmark was also run on another medium instance type c1.medium of EC2 and compared with the Azure medium instance. This was done to analyze if c1.medium performed better than Azure medium instance. C1.medium has five times better computing power than m1.medium but has less RAM (1.75 GB) than m1.medium (section 4.3).

The results showed that the average time taken by c1.medium was less than Azure medium instance, which means c1.medium was faster than Azure. However, the throughput (average MB/s) of c1.medium was less than Azure medium for the copy and scale operations as shown in the graphs in Figure 11 and Figure 12 respectively and more than Azure medium instance type for Add and Triad operations as shown in the graphs in Figure 13 and Figure 14 respectively.

# of Nodes	Copy		Scale		Add		Triad	
	EC2	Azure	EC2	Azure	EC2	Azure	EC2	Azure
1	3699.3944	3280.6003	3576.5643	3266.85	3743.5912	2905.03	3679.2813	2680.64
2	3125.1700	3253.7200	3258.4500	3275.60	3480.2400	2912.85	3207.8600	2686.60
4	3294.3600	3395.3300	3245.5400	3405.06	3334.8800	3012.19	3345.7000	2780.18
6	3173.5200	3396.2000	3147.3600	3421.25	3225.1600	3017.80	3204.8600	2789.75
8	3051.7000	3364.6700	3020.9300	3378.15	3178.9600	2969.25	3187.1500	2741.68
P-value	0.586556499		0.354999199		0.012026079		0.002426183	

Table 7: STREAM - c1.medium and Azure Medium Instance Average MB/S

# of Nodes	Copy		Scale		Add		Triad	
	EC2	Azure	EC2	Azure	EC2	Azure	EC2	Azure
1	0.0101	0.0109	0.0094	0.0119	0.0133	0.0226	0.0137	0.0249
2	0.0107	0.0135	0.0103	0.0128	0.0149	0.0215	0.0154	0.0207
4	0.0097	0.0129	0.0094	0.0108	0.0134	0.0186	0.0134	0.0218
6	0.0097	0.0106	0.0103	0.0120	0.0149	0.0215	0.0152	0.0238
8	0.0106	0.0116	0.0109	0.0099	0.0148	0.0167	0.0145	0.0181
P-value	0.033210533		0.04892193		0.003889974		0.002175783	

Table 8: STREAM - c1.medium and Azure Medium Instance Average Time in S

From these results, it can also be seen that the throughput for c1.medium was more than that of m1.medium. This might be because of the higher CPU capacity provided by the EC2's High-CPU medium instance than the Azure medium instance (section 4.3).

The difference between the Azure medium instance and c1.medium were also found to be statistically significant for both average MB/s and average time for STREAM triad (Table 7 and Table 8).

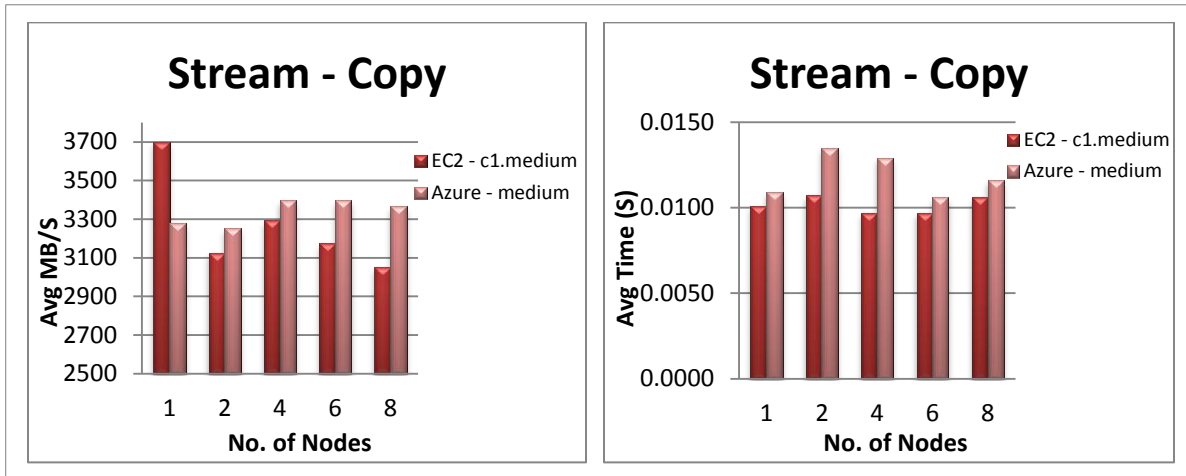


Figure 11: STREAM Copy - c1.medium and Azure Medium Instance

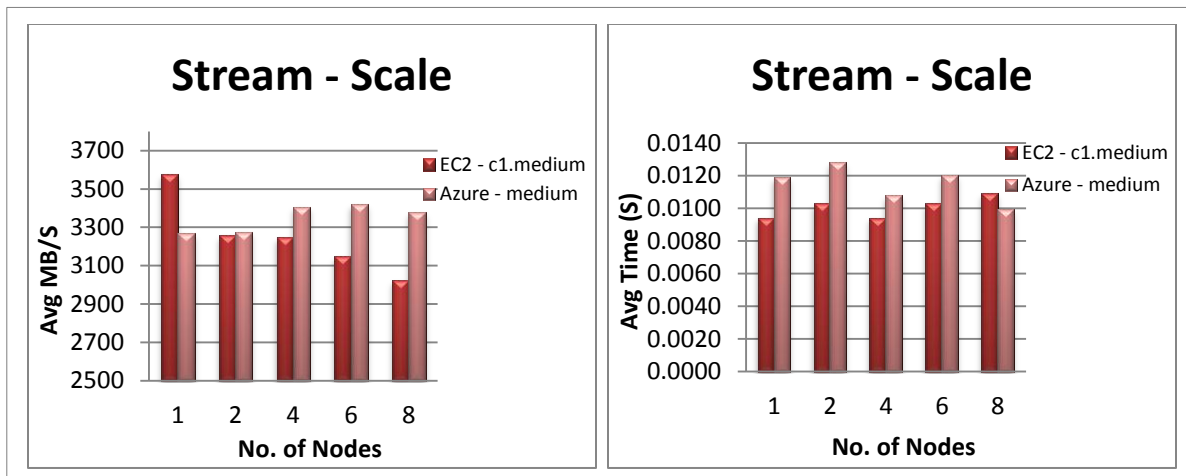


Figure 12: STREAM Scale - c1.medium and Azure Medium Instance

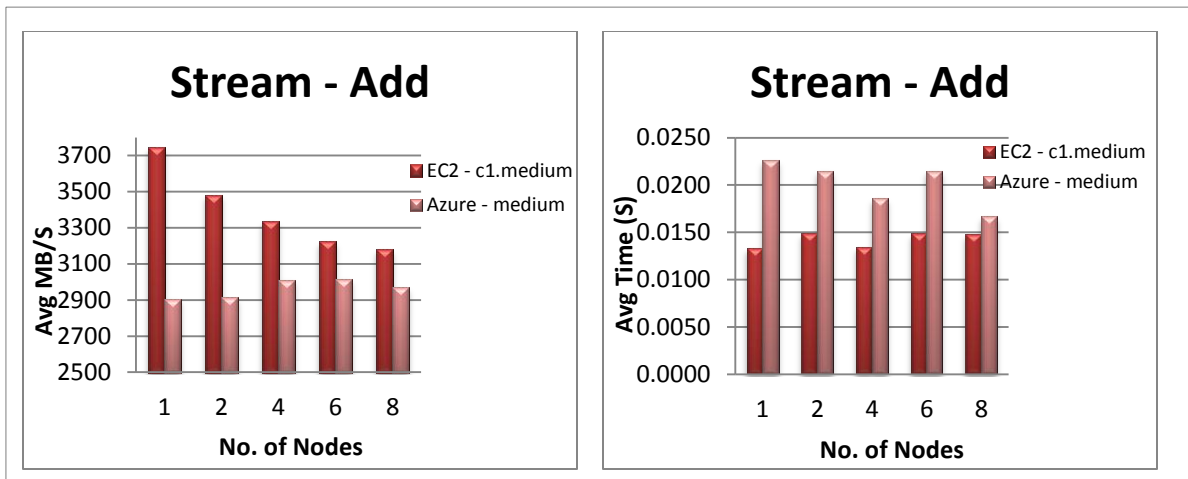


Figure 13: STREAM Add - c1.medium and Azure Medium Instance

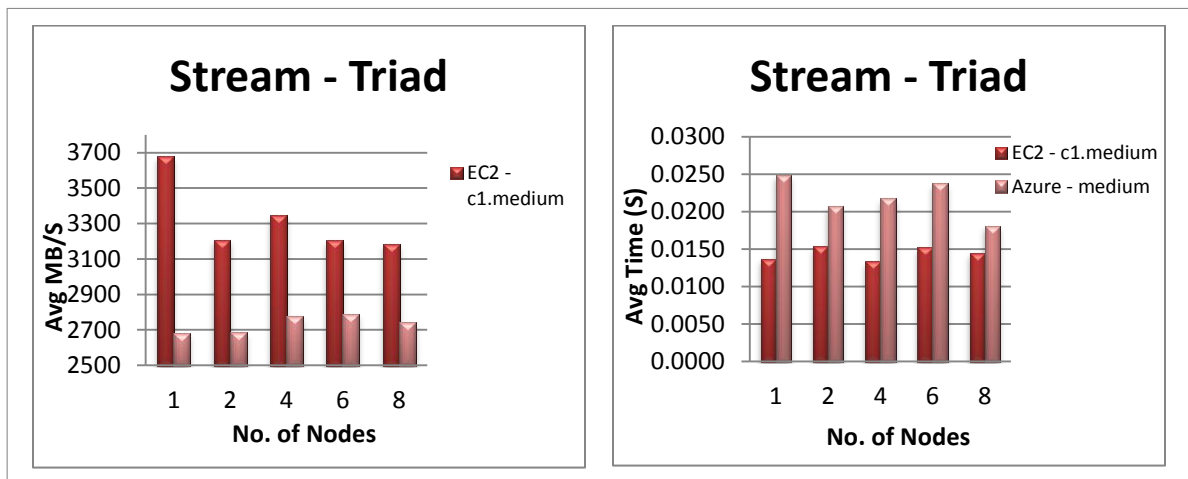


Figure 14: STREAM Triad - c1.medium and Azure Medium Instance

## 7.2 Interleaved Or Random Benchmark

IOR benchmark tests the system performance by focusing on parallel/sequential read/write operations that are typical of scientific applications. The data are written and read using independent parallel transfers of equal-sized blocks of contiguous bytes that



cover the file with no gaps and that do not overlap each other. The test consists of creating a new file, writing it with data, then reading the data back.

Caching appeared to have a big impact on READ/WRITE performance on both EC2 and Windows Azure platforms. Therefore, the READ/WRITE values appeared to be higher after the first iteration of the execution. For this reason, this benchmark was run for one iteration only so non-cached data could be analyzed from a performance perspective.

#### 7.2.1 EC2 Standard Small Instance (m1.small) Versus Azure Small

The graph in Figure 15 shows the read and writes performance on Amazon EC2 m1.small instance type and Azure small instance type. A block size of 1 GiB was used during the execution, which means that a test file of 1 GiB was written and then read while the benchmark was executed. A transfer size of 4 MB was used which implies that each read operation will read the data in the chunks of 4 MB until the entire file of 1 GiB was read.

Since the block size was 1 GiB and the RAM size in both the small instances was 1.7 GB this test had the benefit of buffered caching. When large HPC applications are run on the Cloud, it is important to understand how well the buffered caching would help in the performance of the applications.

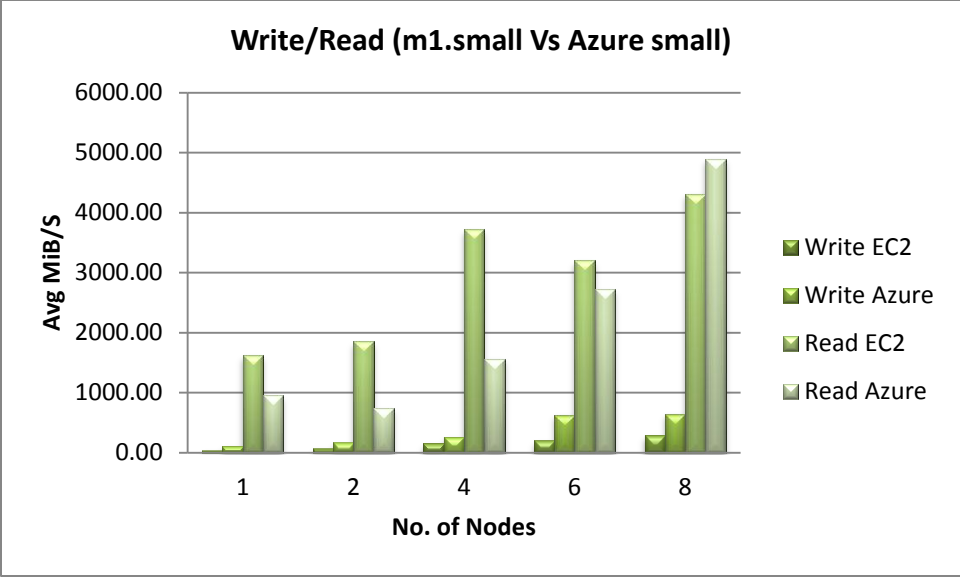


Figure 15: IOR - m1.small and Azure Small Instance

# of Nodes	Write		Read	
	EC2	Azure	EC2	Azure
1	35.39	114.11	1620.84	949.15
2	69.78	165.59	1851.46	736.30
4	155.11	263.74	3717.61	1561.50
6	199.97	629.11	3199.55	2725.77
8	285.51	645.70	4311.75	4894.71
P-value	0.069476615		0.217196694	

Table 9: IOR - m1.small and Azure Small Instance Average MiB/S

The write performance in EC2 was better than that in Azure. This was because the Amazon EC2 instance VM and the local instance store volumes are located in the same physical server; interaction with this storage was very fast, particularly for sequential

access. Local instance store volumes are ideal for temporary storage of information that is continually changing, such as buffers, caches, scratch data, and other temporary content. Amazon EC2 instance storage is designed for this purpose.

A statistical analysis was performed and the values showed that the difference for both the write and read performance between the two instance types was found to be statistically insignificant with a p-value of 0.06 for Write operation and 0.92 for Read operation (Table 9).

#### 7.2.2 EC2 Standard Medium Instance (m1.medium) Versus Azure Medium

In Figure 16, the read performances of Azure and EC2 were seen to be better than the write performances of the two clouds (Table 8). Azure write performance was slightly better than EC2 write performance and was found to be statistically significant.

However, the read performance of EC2 was higher than Azure's read performance. Both the medium instances had similar hardware configuration with EC2 having 3.75 GB RAM and Azure having 3.5 GB RAM. In EC2 the test file was created and read from the local instance storage, which was located on the same server as of the VM itself, whereas in Azure it was read from the windows storage account. This might have been the reason for the difference in read performances on the two clouds.

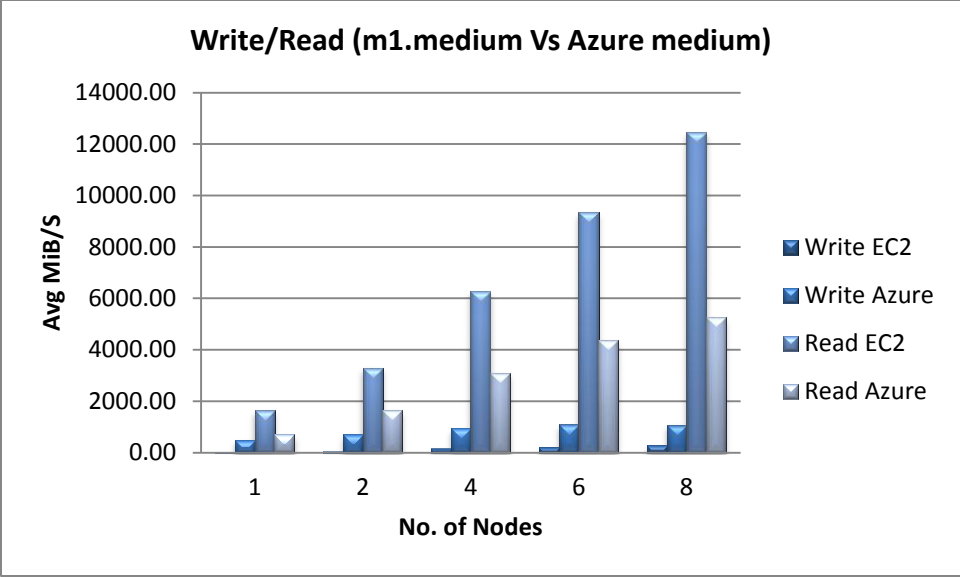


Figure 16: IOR - m1.medium and Azure Medium Instance

# of Nodes	Write		Read	
	EC2	Azure	EC2	Azure
1	12.52	500.67	1629.23	699.25
2	54.87	702.84	3292.59	1624.49
4	145.92	950.5	6279.59	3088.93
6	209.56	1082.62	9359.45	4365.26
8	283.09	1075.59	12446.01	5262.51
P-value	0.001564276		0.149689864	

Table 10: IOR - m1.medium and Azure Medium Instance Average MiB/S

The difference between m1.medium and Azure medium instances was found to be significant statistically for the write performance with a p-value of 0.0015 (Table 10) unlike the small instance types.

### 7.2.3 EC2 High-CPU Medium Instance (c1.medium) Versus Azure Medium

As in Figure 17, again the read performances were better than write performances. The difference in write performance between the two clouds was comparatively less than the difference between their read performances. The read performance of EC2 cloud was better than Azure Read performance. This was because of the fact that in EC2, the file was accessed from the local instance storage that was faster to access. Also, since the RAM size in c1.medium was only 1.7 GB compared to 3.75 GB in m1.medium it appeared that the read performance was better in m1.medium than c1.medium due to better buffering effect.

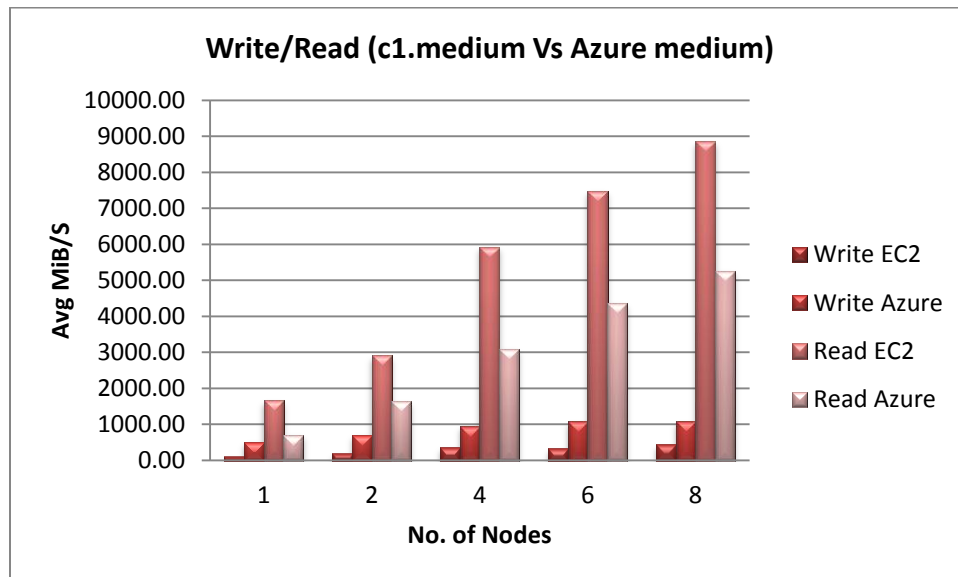


Figure 17: IOR - c1.medium and Azure Medium Instance

# of Nodes	Write		Read	
	EC2	Azure	EC2	Azure
1	92.50	500.67	1658.09	699.25
2	189.77	702.84	2906.45	1624.49
4	345.24	950.5	5922.35	3088.93
6	320.45	1082.62	7485.01	4365.26
8	435.07	1075.59	8851.69	5262.51
P-value	0.003723622		0.185188569	

Table 11: IOR - c1.medium and Azure Medium Instance Average MiB/S

The statistical analysis performed showed that the difference in write performance between c1.medium and Azure medium instance type was statistically significant with a p-value of 0.0037 (Table 11).

### 7.3 NAS Parallel Benchmarks (NPB -CG, FT, EP)

In EP benchmark there was no communication between the nodes, hence it was a pure test for computation performance of the instance and the CPU capacity of the small instances would play a vital role. The CG benchmark was quite memory intensive and it proved to be a test for communication performance. The FT benchmark was used to test both the computation and communication performance of the instances. For each of the benchmarks both the execution time and Mop/s (Millions of Operations/s) were measured.

The NPB benchmarks CG inherently ran 15 iterations and gave an output of average values when executed in a cluster. For example in the below matrix for NPB-CG benchmark, a value of 6.86 for a two-node cluster on EC2 was the average of 15 iterations.

### 7.3.1 EC2 Standard Small Instance (m1.small) Versus Azure Small

Though both EC2 and Azure small instance types used for this experiment had equivalent hardware configurations, Azure small instance was found to be faster than EC2 m1.small. The difference in the underlying architecture and implementation of MPI in the two clouds might have attributed to this behavior. MPI impacts the communication performance. The execution time taken by m1.small increased as the number of nodes was varied and was highest when CG was run on 8 nodes as shown in the graph below in Figure 18, whereas in Azure the execution time did not vary much even when the nodes were varied as 1, 2, 4, 6, and 8 nodes. In all the cases, execution time of CG on Azure small instance was less compared to m1.small. Also, the Millions of Operations generated per second (Mop/s) by Azure small instance was greater than m1.small for all the number of nodes as shown in graph below in Figure 18. The EC2 standard small instance did not seem to perform well for the communication intensive task.

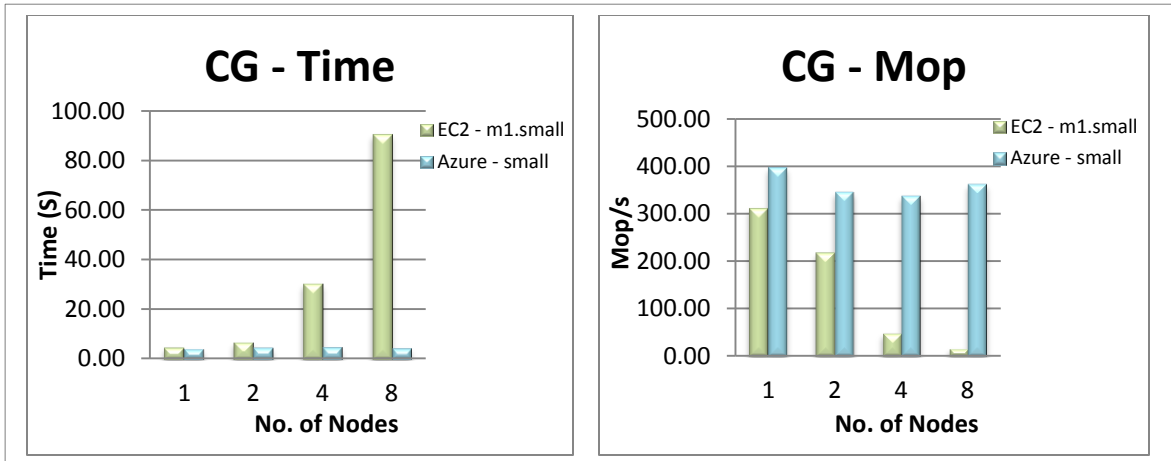


Figure 18: CG - m1.small and Azure Small Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	4.80	3.77	311.65	397.45
2	6.86	4.33	218.21	345.86
4	30.53	4.43	49.02	338.03
8	90.64	4.13	16.51	362.59
P-value	0.242657801		0.053827059	

Table 12: CG - m1.small and Azure Small Instance

The FT benchmark inherently ran 6 iterations and gave an output of average values when executed in a cluster. For example in Table 13 below for NPB-FT benchmark, a value of 27.03 for a two-node cluster on EC2 was the average of six iterations.



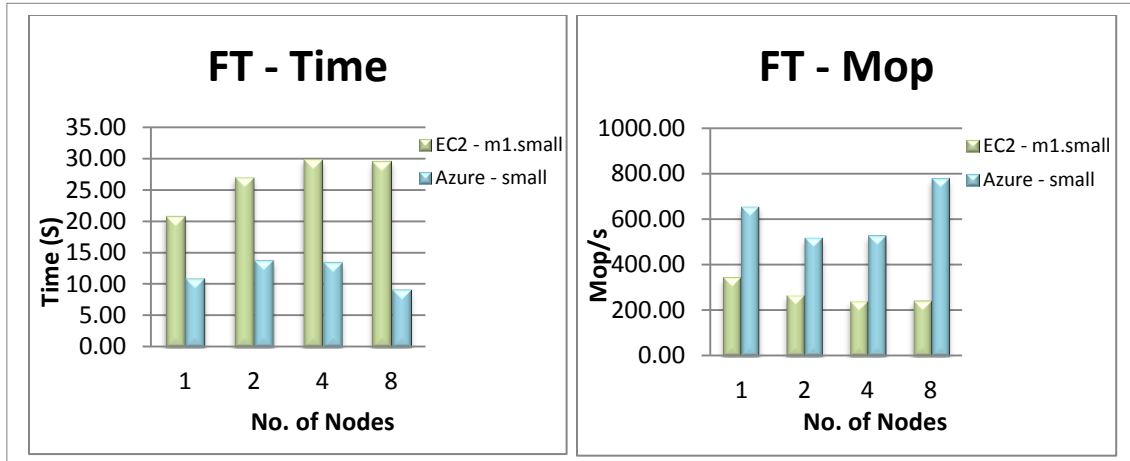


Figure 19: FT - m1.small and Azure Small Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	20.86	10.90	342.05	654.68
2	27.03	13.73	264.04	519.95
4	29.88	13.47	238.82	529.92
8	29.61	9.17	241.05	778.64
P-value	0.00		0.01	

Table 13: FT - m1.small and Azure Small Instance

In small instance type, Azure instance was found to be much faster than the EC2 instance. Though both instance types have equivalent configurations, Azure small instance took much less time than EC2 m1.small for the execution of FT benchmark as shown in Figure 19 above. This might be because of the processor type used in EC2 versus Windows Azure. The details of hardware specifications are outlined in section 4.3 for both the cloud architectures. It was also observed that the mop generated per second was higher for Azure medium instance than m1.small for all the 1, 2, 4, 6, and 8 nodes.

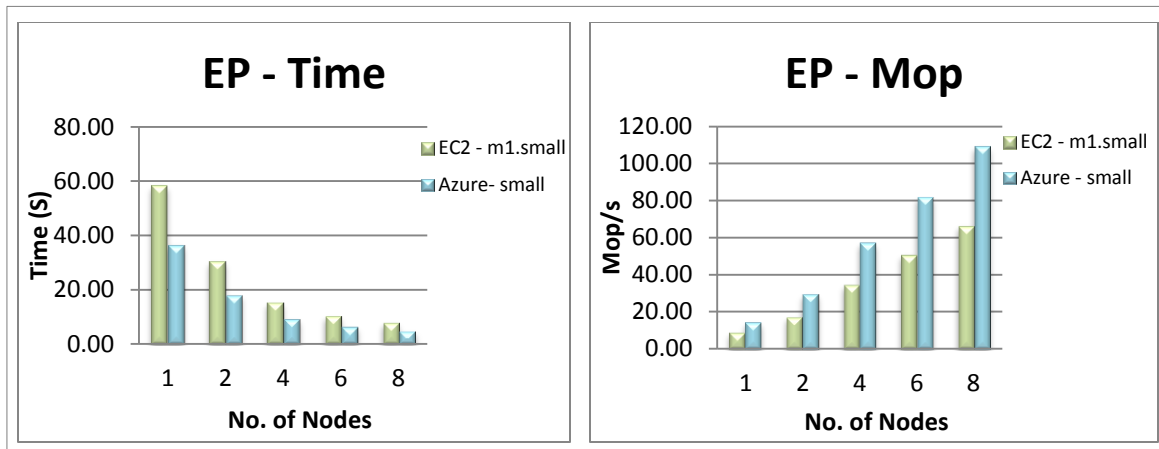


Figure 20: EP - m1.small and Azure Small Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	58.42	36.44	9.19	14.73
2	30.59	18.05	17.55	29.75
4	15.48	9.35	34.68	57.44
6	10.56	6.55	50.83	81.93
8	8.09	4.93	66.37	108.97
P-value	0.413384128		0.294118542	

Table 14: EP - m1.small and Azure Small Instance

In both EC2 and Azure small instance types, the execution time for EP benchmark kept decreasing as the number of nodes were being increased and it took the least time when run on eight nodes and maximum time when run on one node. This is shown in the graph in Figure 20 above. In addition, the Mop/s increased as the number of nodes increased with the highest Mop/s obtained when the benchmark was run on eight nodes on both the Azure and EC2 small instance types. When the two instance types are compared it

appears that Azure small instance was faster than the EC2 small instance and also Azure small instance generates more Mop/s than the EC2 small (m1.small) instance type. This was because of the difference in the type of underlying processor that the two small instances provide. M1.small has 1 virtual core with 1 EC2 compute unit (section 4.3), whereas Azure small instance provides quad-core processor of 2.10 GHz.

Though graphically there seemed to be much difference between the EC2 and Azure small instance types, a statistical analysis was performed to see if these differences were significant. This analysis showed that the difference in values between the two small instance types were statistically significant for CG benchmark with a p-value of 0.05 (Table 12) and for FT benchmark with a p-value of 0.01 for Mop/s (Table 13).

### 7.3.2 EC2 Standard Medium Instance (m1.medium) Versus Azure Medium

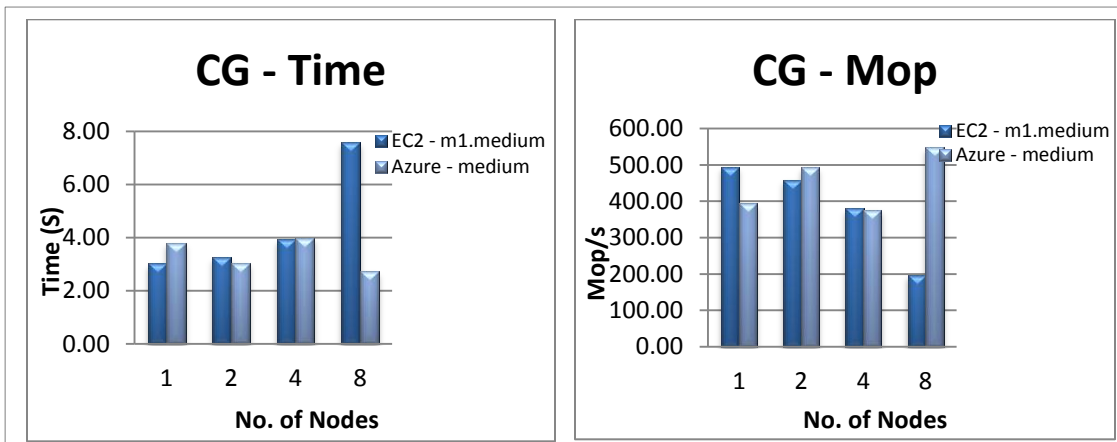


Figure 21: CG - m1.medium and Azure Medium Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	3.03	3.80	494.02	393.77
2	3.26	3.04	458.82	492.04
4	3.93	3.98	380.37	375.71
8	7.57	2.73	197.57	547.98
P-value	0.40		0.41106823	

Table 15: CG - m1.medium and Azure Medium Instance

Both Azure medium instance and EC2's m1.medium have equivalent configurations. M1.medium performed better on one node, taking less execution time than Azure medium instance and generating more mop per second as shown in the graph in Figure 21 above. The performance of m1.medium deteriorates as the number of nodes was varied as 2, 4, 6, and 8 nodes. Azure medium instance performed better than m1.medium instance on all other nodes. The difference in the MPI implementations (MS-MPI and MPICH2) between Windows Azure and EC2, which is required for communication performance in a cluster might have contributed to this behavior. The differences seemed to be statistically insignificant when a statistical analysis was performed with the values in Table 15 above.

For the FT benchmark it was observed that Azure medium instance performed better than EC2 standard medium instance (m1.medium) only when the benchmark was executed on one node as shown in the graph below in Figure 22 below. The difference in the processor type (section 4.3) and implementation of MPI (MS-MPI and MPICH2) in EC2

and Windows Azure might have contributed to this behavior. When the number of nodes were varied as 2, 4, 6, and 8, the performance of m1.medium got better and was found to be faster than Azure medium instance.

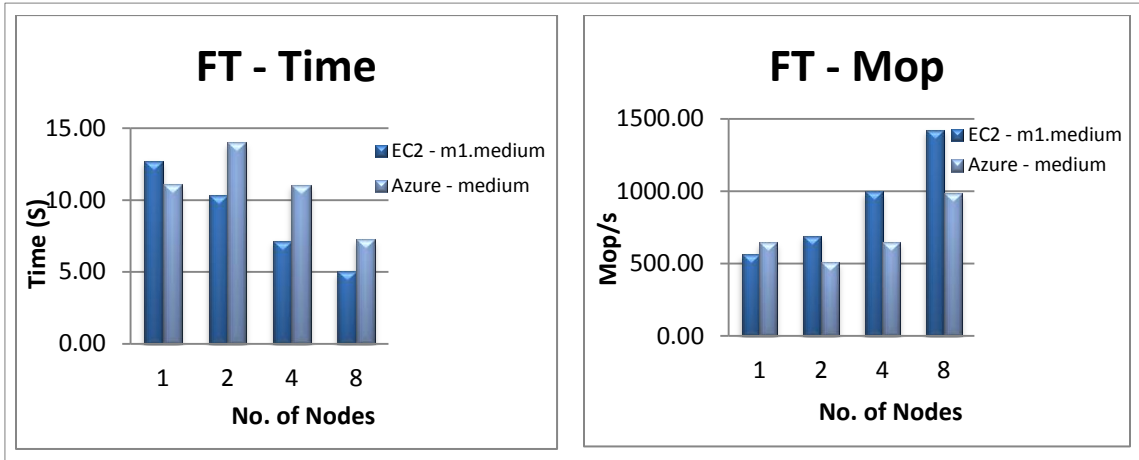


Figure 22: FT - m1.medium and Azure Medium Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	12.71	11.06	561.52	645.40
2	10.35	14.02	689.66	509.00
4	7.14	11.02	999.00	647.52
8	5.02	7.23	1421.80	987.38
P-value	0.39		0.36	

Table 16: FT - m1.medium and Azure Medium Instance

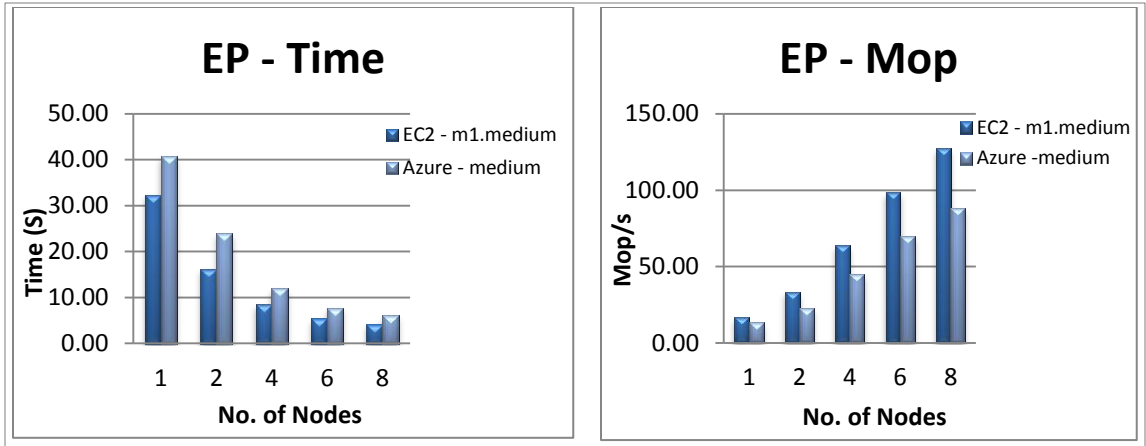


Figure 23: EP - m1.medium and Azure Medium Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	32.32	40.70	16.61	13.19
2	16.16	23.87	33.23	22.49
4	8.40	11.97	63.90	44.85
6	5.46	7.69	98.29	69.83
8	4.22	6.08	127.30	88.24
P-value	0.582828501		0.442356217	

Table 17: EP - m1.medium and Azure Medium Instance

Unlike the small instance type, in medium instance type the EC2's m1.medium was faster and performed better than the Azure medium instance. The time taken for the EP benchmark kept decreasing as the number of nodes was increased from one to eight for both m1.medium instances and the Azure medium instances. Also, the Mop/s generated was least when the benchmark was run on one single node and increased as the number

of nodes were varied from one to eight and was maximim when run on eight nodes. These behaviors are shown in the graphs in Figure 23 above. The EC2 m1.medium instance type took less time than Azure medium instance and generated greater Mop/s than Azure Medium instance type. The difference in the processor types in EC2 and Azure medium instances seems to affect the performance. M1.medium has one virtual core with 2 EC2 compute units (section 4.3) while Azure medium instance provides 2.09 GHz processor speed and has two cores.

A statistical analysis was performed to determine how significant the difference between m1.medium and Azure medium instance types was. This difference was found to be statistically insignificant as shown from the p-values in Table 15, Table 16 and Table 17. For CG benchmark, EC2 performed better on one node and for FT benchmark Azure performed better on one node only with its performance deteriorating for rest of the nodes.

### 7.3.3 EC2 High-CPU Medium Instance (c1.medium) Versus Azure Medium

C1.medium was found to be faster than the Azure medium instance. The performance of c1.medium got better as the number of nodes was increased. The execution time taken by c1.medium was less than that taken by Azure medium instance and also the the Mop generated per second by c1.medium was higher than Azure medium instance. This is reflected in the graph in Figure 24 below. Though c1.medium has a RAM of 1.7 GB compared to 3.75 GB RAM size of Azure medium instance, c1.medium performs better

than the latter. So, the high cpu power of c1.medium compared to Azure medium instance seemed to contribute towards the better performance of c1.medium.

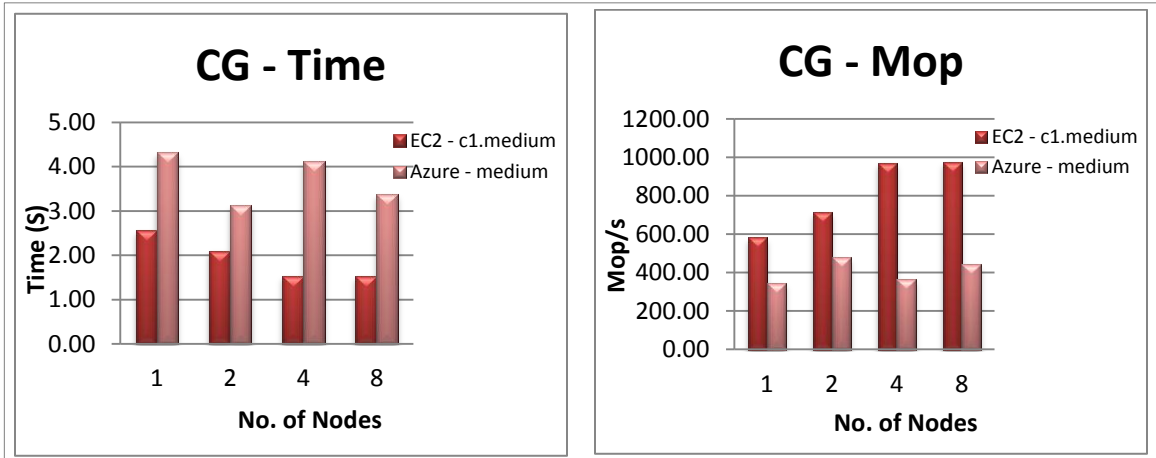


Figure 24: CG - c1.medium and Azure Medium Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	2.56	4.33	583.53	345.24
2	2.1	3.13	714.1	478.06
4	1.54	4.12	969.91	363.38
8	1.53	3.38	974.97	443.09
P-value	0.003321119		0.020320918	

Table 18: CG - c1.medium and Azure Medium Instance



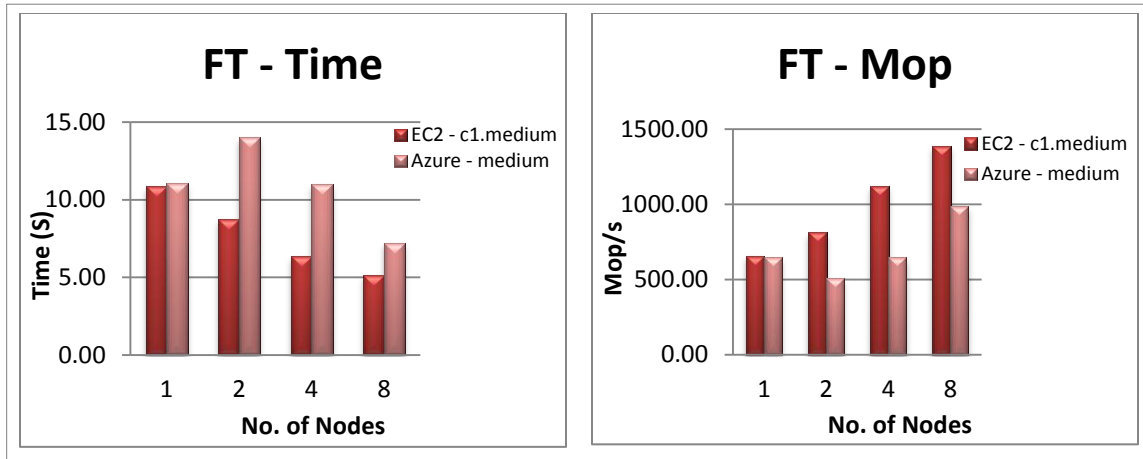


Figure 25: FT - c1.medium and Azure Medium Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	10.86	11.06	657.03	645.40
2	8.75	14.02	815.47	509.00
4	6.36	11.02	1121.55	647.52
8	5.14	7.23	1387.23	987.38
P-value	0.16		0.47	

Table 19: FT - c1.medium and Azure Medium Instance

It appeared that EC2 high-CPU medium instance (c1.medium) was faster than the Azure medium instance. It shows that the high CPU power provided by EC2 c1.medium instance has helped in this behavior (section 4.3). FT depends on both the computation power of the instances and also the communication performance. As the number of nodes were varied from 1, 2, 4, 6, and 8 the execution time for FT decreased for c1.medium and was also found to be less than that of Azure medium instance. Similarly,

the Mop generated per second was higher for c1.medium than Azure medium instance for all the nodes. The graphs in Figure 25 reflect this behavior.

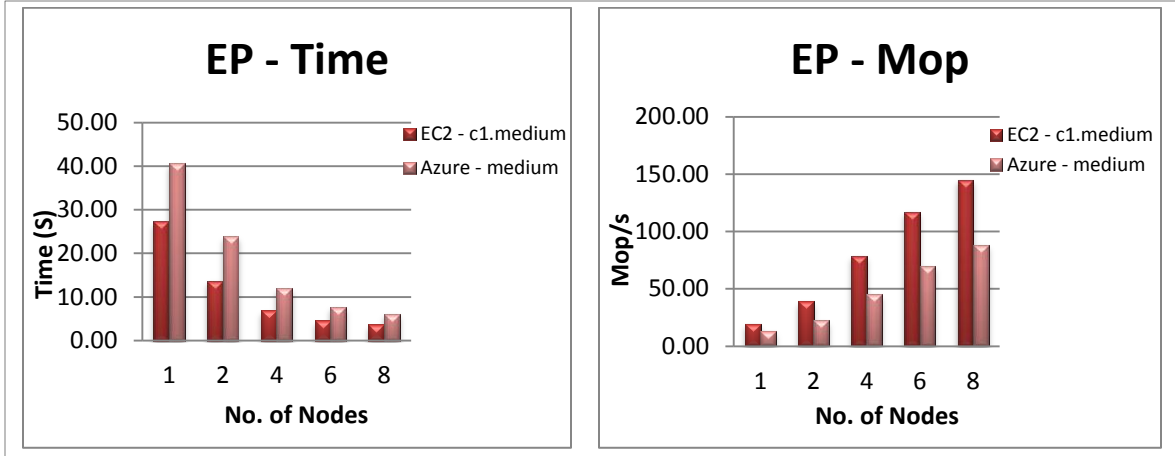


Figure 26: EP - c1.medium and Azure Medium Instance

# of Nodes	Time (S)		Mop/s	
	EC2	Azure	EC2	Azure
1	27.34	40.70	19.63	13.19
2	13.66	23.87	39.30	22.49
4	6.83	11.97	78.65	44.85
6	4.58	7.69	117.21	69.83
8	3.70	6.08	144.93	88.24
P-value	0.410045319		0.278304561	

Table 20: EP - c1.medium and Azure Medium Instance

C1.medium was faster than EC2's m1.medium instance type (section 7.3.2). C1.medium has five times more CPU power than m1.medium. C1.medium has two virtual cores and 5 EC2 compute units whereas Azure medium instance has two virtual cores with 2.09

GHz processing power. The EC2 c1.medium was much faster than Azure medium instance. C1.medium took much less time than Azure medium instance. The time taken by EP benchmark kept decreasing as the number of nodes were varied and was least while it was run on 8 nodes. The Mop generated per second was found to be increasing in both c1.medium and Azure medium instance types as the nodes were varied as 1, 2, 4, 6, and 8 nodes with m1.medium generating more mop per second than Azure medium instance. These behaviors are reflected in the graphs in Figure 26 above. This difference in the mop generated per second was found to be larger than the difference between m1.medium and Azure medium instance. This might be because of the higher computing power provided by the c1.medium instance type than m1.medium.

The statistical analysis performed showed that the difference between c1.medium and Azure medium instance types was found to be statistically significant for the CG benchmark with a p-value of 0.003 for the execution time and a p-value of 0.02 for Mop/s (Table 18). The p-values for FT (Table 19) and EP (Table 20) benchmarks were found to be statistically insignificant.

## Chapter 8

### CONCLUSION

The conclusions are categorized by the following output types: Memory bandwidth, I/O Performance and Communication & Computational performance for Amazon EC2 and Windows Azure cloud platforms and are based on the detailed analysis performed in Chapter 7.

Memory bandwidth appeared to be more for EC2's standard small instance m1.small when compared to Windows Azure's small instance type. The graphs in the detail analysis indicated that memory bandwidth was consistently higher for EC2 than Windows Azure when cluster size increased. Further statistical analysis confirmed the same behavior. Memory bandwidth fared better for EC2 compared to Windows Azure. Though a detailed analysis was performed on the STREAM benchmark, it was hard to conclude which of the two small instances was faster between Amazon EC2 and Windows Azure when execution time was considered as a measure.

The detailed analysis of EC2's standard medium instance m1.medium and Azure's medium instance type clearly showed that m1.medium was faster and provided much better memory bandwidth compared to Azure's medium instance type. As the cluster size

increased the shared memory bandwidth showed increased performance for EC2 compared to Azure. In comparison of EC2's High-CPU medium instance (c1.medium) and Azure's medium, EC2 clearly showed better memory bandwidth.

I/O comparison involved measuring READ and WRITE operations with varied number of nodes on both Amazon EC2 and Windows Azure. From the detailed analysis it appeared that difference for both read and write performances was insignificant in both the platforms for EC2's m1.small and Azure's small instance types. This means that small instances performed almost same for read and write in EC2 and Windows Azure. This behavior was further confirmed from the statistical analysis that proved it insignificant.

For the medium instances, Azure's medium instance appeared to have better write performance than EC2's m1.medium write performance. At the same time, EC2 performed better for read operation over Azure's medium instance. Detailed analysis proved this behavior. This was because on EC2, the data was written and read from the local instance storage. EC2's other medium instance c1.medium showed similar behavior when compared to Azure's medium instance.

It was clear that Windows Azure performed better than EC2 in both communication and computational performances for small instance types with increasing number of nodes in a cluster. Problem Class A was selected to run the NPB benchmarks on both the platforms. The communication and computational power consistently increased with the

increasing number of nodes in both the platforms. However, Windows Azure performance was better in all the cases when compared to EC2.

On the medium instances communication performance (CG) of Azure (medium instance) appeared to have performed better than EC2's m1.medium instance when the number of nodes was varied as 2, 4, 6, and 8 nodes except on one node where m1.medium performed better. Nevertheless, the Computational performance (EP) of m1.medium instance was better than Azure's medium instance for all the nodes.

The communication performance of EC2's High-CPU medium instance, c1.medium appeared to be better than Azure's medium instance. Same behavior was observed for computational power also. EC2's c1.medium instance computational performance was better compared to Azure's medium instance because of the high CPU power of c1.medium instance.

Overall, it appeared that Amazon EC2 was well suited for memory intensive applications. Both Small and Medium Amazon instance types showed this behavior. Windows Azure on the other hand appeared to be better for communication performance for both small and medium instance types. For computational and communication performance perspective Amazon EC2's c1.medium instance type appeared to be more suitable over Window's Azure's comparable instance types.

## 8.1 Future Research

From the research accomplished during the course of this thesis, there appeared to be a lot of scope for benchmarking and comparing performances of various public cloud computing platforms. The scope of this thesis was limited to benchmarking Amazon EC2 and Windows Azure with STREAM, IOR and NPB benchmarks. NPB3.3 benchmark alone has 12 benchmarks with six Problem Classes. Each problem class is a level of complexity of the benchmark problem. These problem classes can be further explored to benchmark the cloud platforms. Amazon and Microsoft are innovating and regularly implementing newer instance types and are supporting Operating Systems that were not supported earlier. Windows Azure particularly is evolving at the time of writing this thesis and is adding new Operating Systems. So, it presents a lot of scope for research on assessing performance both from commercial and scientific perspective.

This thesis benchmarked the small and medium instance types of both Amazon EC2 and Windows Azure. A cluster of 8 nodes was used for this purpose and HPC benchmarks were executed on the same by increasing the number of nodes that measured memory bandwidth, I/O performance and Communication & Computational performance of these two cloud platforms. This helped in understanding how the performance was impacted when the number of nodes in the cluster was increased and how it was impacted when the instance types were varied from small to medium at the same time. But, since HPC

applications require clusters with really high computing power, the large and extra-large (or the cluster compute instance type in EC2) instance types and also larger cluster sizes could be used to get significant performance improvements.

There are organizations and researchers who have performed experiments on Amazon EC2 but not many are out there working on Windows Azure. So the benchmarking process presented some challenges including building the windows binaries that are compatible with MS-MPI for Azure platform. These benchmarks were written in C and Fortran languages and are inherently supported by GCC and MPICC compiler for MPI versions. They were not written for benchmarking Windows platform. Cross-compilers had to be used to build the windows binaries. This area can be further explored for benchmarking Windows Platforms.



## REFERENCES

### Print Publications:

[Amedro10]

Amedro, B., F.O. Baude, D. Caromel, C. Delbe', I. Filali, F. Huet, E. Mathias, and O. Smirnov, *Cloud Computing: Principles, Systems and Applications*, Springer, Guildford, 2010.

[Evangelinos08]

Evangelinos, C. and Hill, C. N., "Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2", *Proceedings of Cloud Computing and Its Applications, ACM Workshop (CCA'08)*, New York ACM, (October 2008).

[Gillam10]

Gillam, L. and N. Antonopoulos, *Cloud Computing: Principles, Systems and Applications*, Springer, Guildford, 2010.

[Ghoshal11]

Ghoshal, D., Canon, R, C, and Ramakrishnan, N., "I/O performance of virtualized cloud environments", *Proceedings of the second international workshop on Data intensive computing in the clouds*, pp. 71-80, NY, 2011.

[Hazelhurst08]

Hazelhurst, S., "Scientific computing using virtual high-performance computing: a case study using the Amazon elastic computing cloud", SAICSIT. *Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*. ACM, New York. 2008, pp. 94-103.

[SunMicrosystems09]

"Sun Microsystems Introduction to Cloud Computing Architecture", White Paper, 1st Edition, June 2009.

[Velte10]

Velte, A., T. J. Velte, and R.C. Elsenpeter, *Cloud Computing: A Practical Approach*, McGraw Hill Professional, San Francisco, 2010.

[Wong99]

Wong, C.F., Martin, R. P., Arpaci-Dusseau, R.H. and Culler, D.E., Architectural Requirements and Scalability of the NAS Parallel Benchmarks, 99 Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM) Article No.41, New York, 1999.

Electronic Publications:

[AWS12A]

“What was a EC2 Compute Unit and why did you introduce it?”,  
[http://aws.amazon.com/ec2/faqs/#What\\_is\\_an\\_EC2\\_Compute\\_Unit\\_and\\_why\\_did\\_you\\_introduce\\_it](http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it), last accessed April 10, 2012.

[AWS12B]

“Amazon Elastic Block Storage”, <http://aws.amazon.com/ebs/>, last accessed June 14, 2012.

[Marquand10]

Alan Le Marquand, “Windows Azure Platform. Inside the Cloud. Microsoft's Cloud World Explained Part 2”, <http://technet.microsoft.com/en-us/video/windows-azure-platform-inside-the-cloud-microsofts-cloud-world-explained-part-2.aspx>, May 5 2010.

[McCalpin95C]

“Adjust the Problem Size”,  
[https://asc.llnl.gov/computing\\_resources/purple/archive/benchmarks/memory/membench\\_bm\\_readme.html#STREAMS](https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/memory/membench_bm_readme.html#STREAMS), last accessed June 14, 2012.

[McCalpin95A]

“Multiprocessor Runs”, Department of Computer Science,  
<http://www.cs.virginia.edu/stream/ref.html>, last accessed February 12, 2012.

[McCalpin95B]

McCalpin, John D., "Memory Bandwidth and Machine Balance in Current High Performance Computers", IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995.

[Microsoft12]

“Windows HPC Server 2008 R2”, <http://www.microsoft.com/en-us/server-cloud/windows-server/high-performance-computing-hpc.aspx>, last accessed July 14, 2012.

[MPICH12]

“MPICH2”, <http://www.mcs.anl.gov/research/projects/mpich2/>, last accessed Jan 15, 2012.

[MSDN12A]

“Windows Azure HPC Scheduler”, <http://msdn.microsoft.com/en-us/library/windowsazure/hh545593.aspx>, last accessed Feb 27, 2012.

[MSDN12B]“Overview of the sample Azure service”, [http://msdn.microsoft.com/en-us/library/hh560251\(v=vs.85\).aspx#BKMK\\_tools](http://msdn.microsoft.com/en-us/library/hh560251(v=vs.85).aspx#BKMK_tools), last accessed August 13 2012.

[Paratools12]

“The Windows Azure HPC Scheduler”,  
<http://www.paratools.com/Azure/HowToHPCScheduler>, last accessed May 29, 2012.

[Sourceforge12]

“IOR HPC Benchmark”, <http://ior-sio.sourceforge.net/>, last accessed Jan 4, 2012.

[StarCluster12A]

“WhatwasStarCluster?”, <http://web.mit.edu/star/cluster/docs/latest/overview.html>, last accessed May 14, 2012.

[StarCluster12B]

”HPC on AWS”, <http://aws.amazon.com/hpc-applications/>, last accessed May 14, 2012.

[StarCluster12C]

“StarCluster”, <http://web.mit.edu/star/cluster/>, Last accessed May 14, 2012.

[StarCluster12D]

“NFS Shares”, <http://web.mit.edu/star/cluster/docs/latest/overview.html#nfs-shares>, last accessed June 15, 2012.

[StarCluster12E]

“Installing on Windows”,  
<http://web.mit.edu/star/cluster/docs/latest/installation.html?highlight=windows>, last accessed June 15, 2012.

[TechNet12A]

“Deploying Applications to Azure Nodes in Windows HPC Server 2008 R2”,  
[http://technet.microsoft.com/en-us/library/hh162018\(v=ws.10\).aspx](http://technet.microsoft.com/en-us/library/hh162018(v=ws.10).aspx), last accessed May 30, 2012.

[TechNet12B]

“Overview of HPC Job Manager”, <http://technet.microsoft.com/en-us/library/cc972829.aspx>, last updated May 20, 2009, last accessed May 4, 2012.

[TechNet12C]

“7 Things to Know About Windows Azure Capacity”,  
<http://technet.microsoft.com/en-us/cloud/gg663909.aspx>, last accessed July 3, 2012.

[WinFirewall12]

“Configure Windows Azure Firewall for MPI”,  
<http://www.paratools.com/Azure/HowToFirewall>, last accessed June 2, 2012.

[WindowsAzure12]

“The Components of Windows Azure”, <http://www.windowsazure.com/en-us/develop/net/fundamentals/intro-to-windows-azure/#components>, last accessed May 4, 2012.

## APPENDIX A

### Metrics

#### MPI STREAM Benchmark:

Memory bandwidth rate: Memory bandwidth is typically measured in bytes/sec or megabytes/sec (MB/s). STREAM benchmark outputs the memory bandwidth in MB/s and the same unit is also used in graphs and charts as required for Copy, Scale, Add and Triad for Amazon EC2's Standard Small instance and High-CPU medium instance.

Also, average time, minimum time and the maximum time for each operation were calculated and documented in seconds. When multiple cores are used for experimentation, memory bandwidth will be determined and represented in the same way as above.

This MPI version of this benchmark is downloaded from the below url:  
<http://www.cs.virginia.edu/stream/FTP/Code/Versions/>

#### IOR Benchmark (POSIX Mode):

In POSIX mode, the benchmark was run like all other MPI programs. IOR generates a detailed output file that indicates the parameters used to initiate the runs. The maximum read and writes are reported in MiB/sec. 1 Mebibytes (MiB) = 1,048,576 bytes. To get MB/sec MiB/sec must be multiplied by 1.048.

Block Size: Contiguous bytes to write per task (e.g., 8, 4k, 2m, 1g, i.e., the whole size of the written data)

Transfer Size: Size of transfer in bytes (e.g., 8, 4k, 2m, 1g, i.e., the amount of data of a single I/O operation)

Repetitions: Number of repetitions of test

File-per-process: Accesses a single file for each processor; default was a single file accessed by all processors

Example:

Max Write: 106.07 MiB/sec (111.22 MB/sec)

Max Read: 87.04 MiB/sec (91.27 MB/sec)

This benchmark is downloaded from the below url:  
<http://sourceforge.net/projects/ior-sio/>

MPI NAS Parallel Benchmarks:

EP, FT and CG benchmarks were run on the Amazon instances on 1, 2, 4, and 8 nodes and the corresponding execution time in seconds were measured. The Million operations per second (Mop/s) for each benchmark is also measured.

This benchmark is downloaded from the below url:  
<https://www.nas.nasa.gov/cgi-bin/software/start>

## APPENDIX B

### Configuration File and Benchmark Commands

#### StarCluster Configuration File:

```
#####
## StarCluster Configuration File ##
#####

[global]
# configure the default cluster template to use when starting a cluster
# defaults to 'smallcluster' defined below. this template should be
usable
# out-of-the-box provided you've configured your keypair correctly
DEFAULT_TEMPLATE=m1.small-AMI-cluster
# enable experimental features for this release
ENABLE_EXPERIMENTAL=True
# number of seconds to wait when polling instances (default: 30s)
#REFRESH_INTERVAL=15
# specify a web browser to launch when viewing spot history plots
#WEB_BROWSER=chromium

[aws info]
# This is the AWS credentials section.
# These settings apply to all clusters
# replace these with your AWS keys
AWS_ACCESS_KEY_ID = AKIAJVYCYC2QTZVPNCQA
AWS_SECRET_ACCESS_KEY = YOnrYCbG07NvxOcrZkHchpwsATn3MnIEPVJ01Nr5
# replace this with your account number
AWS_USER_ID= 390135667176
# Uncomment to specify a different Amazon AWS region (OPTIONAL)
# (defaults to us-east-1 if not specified)
# NOTE: AMIs have to be migrated!
#AWS_REGION_NAME = eu-west-1
#AWS_REGION_HOST = ec2.eu-west-1.amazonaws.com
# Uncomment these settings when creating an instance-store (S3) AMI
(OPTIONAL)
#EC2_CERT = /path/to/your/cert-asdf0as9df092039asdfi02089.pem
#EC2_PRIVATE_KEY = /path/to/your/pk-asdfasd890f200909.pem
# Uncomment these settings to use a proxy host when connecting to AWS
#aws_proxy = your.proxyhost.com
#aws_proxy_port = 8080
#aws_proxy_user = yourproxyuser
#aws_proxy_pass = yourproxypass

# Sections starting with "key" define your keypairs
```

```

# (see the EC2 getting started guide tutorial on using ec2-add-keypair
to learn
# how to create new keypairs)
# Section name should match your key name e.g.:
[key winkey]
#KEY_LOCATION= ~/.ssh/mykey.rsa
KEY_LOCATION= S:\THESIS\EC2\winkey.rsa
# You can of course have multiple keypair sections
# [key my-other]
# KEY_LOCATION=/home/myuser/.ssh/id_rsa-my-other-gsg-keypair

# Sections starting with "cluster" define your cluster templates
# Section name is the name you give to your cluster template e.g.:
# [cluster smallcluster]

[cluster m1.small-AMI-cluster]

# change this to the name of one of the keypair sections defined above
KEYNAME = winkey

# number of ec2 instances to launch
CLUSTER_SIZE = 8
# create the following user on the cluster
CLUSTER_USER = ec2-user

PLUGINS = mpich2

# optionally specify shell (defaults to bash)
# (options: tcsh, zsh, csh, bash, ksh)
CLUSTER_SHELL = bash

# AMI to use for cluster nodes. These AMIs are for the us-east-1
region.
# Use the 'listpublic' command to list StarCluster AMIs in other
regions
# The base i386 StarCluster AMI is ami-899d49e0
# The base x86_64 StarCluster AMI is ami-999d49f0
# The base HVM StarCluster AMI is ami-4583572c
NODE_IMAGE_ID = ami-999d49f0
# instance type for all cluster nodes
# (options: cg1.4xlarge, c1.xlarge, m1.small, c1.medium, m2.xlarge,
t1.micro, cc1.4xlarge, cc2.8xlarge, m1.large, m1.xlarge, m2.4xlarge,
m2.2xlarge)
NODE_INSTANCE_TYPE = m1.small

# Uncomment to disable installing/configuring a queueing system on the
# cluster (SGE)
#DISABLE_QUEUE=True
# Uncomment to specify a different instance type for the master node
(OPTIONAL)
# (defaults to NODE_INSTANCE_TYPE if not specified)

```



```

MASTER_INSTANCE_TYPE = m1.small

# Uncomment to specify a separate AMI to use for the master node.
(OPTIONAL)
# (defaults to NODE_IMAGE_ID if not specified)
MASTER_IMAGE_ID = ami-999d49f0

# availability zone to launch the cluster in (OPTIONAL)
# (automatically determined based on volumes (if any) or
# selected by Amazon if not specified)
#AVAILABILITY_ZONE = us-east-1c
# list of volumes to attach to the master node (OPTIONAL)
# these volumes, if any, will be NFS shared to the worker nodes
# see "Configuring EBS Volumes" below on how to define volume sections
#VOLUMES = myvol1

[plugin mpich2]
setup_class = starcluster.plugins.mpich2.MPICH2Setup

# list of plugins to load after StarCluster's default setup routines
(OPTIONAL)
# see "Configuring StarCluster Plugins" below on how to define plugin
sections
#[cluster t1-micro-trial-cluster]
#PLUGINS = mpich2
#KEYNAME = mykey
#NODE_INSTANCE_TYPE = t1.micro
#CLUSTER_SIZE = 2
#NODE_IMAGE_ID = ami-31814f58
# list of permissions (or firewall rules) to apply to the cluster's
security
# group (OPTIONAL).
#PERMISSIONS = ssh, http
# Uncomment to always create a spot cluster when creating a new cluster
from
# this template. The following example will place a $0.50 bid for each
spot
# request.
#SPOT_BID = 0.50

#####
## Defining Additional Cluster Templates ##
#####

# You can also define multiple cluster templates.
# You can either supply all configuration options as with smallcluster
above,
# or create an EXTENDS=<cluster_name> variable in the new cluster
section to
# use all settings from <cluster_name> as defaults. Below are a couple
of
# example cluster templates that use the EXTENDS feature:

```

```

# [cluster mediumcluster]
# Declares that this cluster uses smallcluster as defaults
# EXTENDS=smallcluster
# This section is the same as smallcluster except for the following
settings:
# KEYNAME=my-other-gsg-keypair
# NODE_INSTANCE_TYPE = c1.xlarge
# CLUSTER_SIZE=8
# VOLUMES = biodata2
# [cluster largecluster]
# Declares that this cluster uses mediumcluster as defaults
# EXTENDS=mediumcluster
# This section is the same as mediumcluster except for the following
variables:
# CLUSTER_SIZE=16
#####
## Configuring EBS Volumes ##
#####

# A new [volume] section must be created for each EBS volume you wish
to use
# with StarCluster. The section name is a tag for your volume. This tag
is used
# in the VOLUMES setting of a cluster template to declare that an EBS
volume is
# to be mounted and nfs shared on the cluster. (see the commented
VOLUMES
# setting in the example 'smallcluster' template above)
# Below are some examples of defining and configuring EBS volumes to be
used
# with StarCluster:

# Sections starting with "volume" define your EBS volumes
# Section name tags your volume e.g.:
# [volume myvol1]
# (attach 1st partition of volume vol-c9999999 to /home on master node)
# VOLUME_ID = vol-c9999999
# MOUNT_PATH = /home

# Same volume as above, but mounts to different location
# [volume biodata2]
# (attach 1st partition of volume vol-c9999999 to /opt/ on master node)
# VOLUME_ID = vol-c9999999
# MOUNT_PATH = /opt/

# Another volume example
# [volume oceandata]
# (attach 1st partition of volume vol-d7777777 to /mydata on master
node)
# VOLUME_ID = vol-d7777777
# MOUNT_PATH = /mydata

```

```

# Same as oceandata only uses the 2nd partition instead
# [volume oceandata]
# (attach 2nd partition of volume vol-d7777777 to /mydata on master
node)
# VOLUME_ID = vol-d7777777
# MOUNT_PATH = /mydata
# PARTITION = 2
#####
## Configuring StarCluster Plugins ##
#####

# Sections starting with "plugin" define a custom python class which
can
# perform additional configurations to StarCluster's default routines.
These
# plugins can be assigned to a cluster template to customize the setup
# procedure when starting a cluster from this template
# (see the commented PLUGINS setting in the 'smallcluster' template
above)
# Below is an example of defining a plugin called 'myplugin':

# [plugin myplugin]
# myplugin module either lives in ~/.starcluster/plugins or is
# in your PYTHONPATH
# SETUP_CLASS = myplugin.SetupClass
# extra settings are passed as arguments to your plugin:
# SOME_PARAM_FOR_MY_PLUGIN = 1
# SOME_OTHER_PARAM = 2

#####
## Configuring Security Group Permissions ##
#####

# [permission ssh]
# protocol can be: tcp, udp, or icmp
# protocol = tcp
# from_port = 22
# to_port = 22
# cidr_ip = <your_ip>/32

# example for opening port 80 on the cluster to a specific IP range
# [permission http]
# protocol = tcp
# from_port = 80
# to_port = 80
# cidr_ip = 18.0.0.0/24

```

### STREAM Commands:

```

root@master:/home/ec2-user/STREAM-MPI# mpicc -DPARALLEL_MPI -O3 -o
stream_mpi stream_mpi.c

```

```
root@master:/home/ec2-user/STREAM-MPI# mpiexec -host master, node001,
node002, node003 ./stream_mpi > output/cl.m_n4.1.txt
```

## IOR Commands:

```
root@master:/mnt/ec2-user/IOR/src/C# make
```

```
mpicc -g -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -c IOR.c
mpicc -g -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -c utilities.c
mpicc -g -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -c
parse_options.c
mpicc -g -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -c aiori-POSIX.c
mpicc -g -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -c aiori-
noMPIIO.c
mpicc -g -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -c aiori-
noHDF5.c
mpicc -g -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -c aiori-
noNCMPI.c
mpicc -o IOR IOR.o utilities.o parse_options.o \
        aiori-POSIX.o aiori-noMPIIO.o aiori-noHDF5.o aiori-
noNCMPI.o \
```

```
root@master:/mnt/ec2-user/IOR/src/C# mpiexec -host master, node001,
node002, node003 ./IOR -b 1g -t 4m > output/cl.m_n4.1.txt.
```

## NPB Commands:

### CG:

```
root@master:/home/ec2-user/NPB3.3/NPB3.3-MPI# make cg NPROCS=4 CLASS=A
The output for this command appears as below.
```

```
root@master:/home/ec2-user/NPB3.3-MPI# make CG NPROCS=4 CLASS=A
```

```
=====
=          NAS Parallel Benchmarks 3.3          =
=          MPI/F77/C                             =
=====
```

```
cd CG; make NPROCS=4 CLASS=A
```

```
make[1]: Entering directory `/home/ec2-user/NPB3.3-MPI/CG'
make[2]: Entering directory `/home/ec2-user/NPB3.3-MPI/sys'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/home/ec2-user/NPB3.3-MPI/sys'
../sys/setparams CG 4 A
```

```
mpif77 -c -I/usr/local/include -O CG.f
mpif77 -O -o ../bin/CG.A.4 CG.o ../common/randi4.o
../common/print_results.o ../common/timers.o -L/usr/local/lib -lmpi
make[1]: Leaving directory `/home/ec2-user/NPB3.3-MPI/CG'
root@master:/home/ec2-user/NPB3.3-MPI# mpiexec -host master, node001,
node002, node003 bin/cg.A.4 > output/cg.A.4_3.txt
```

FT:

```
root@master:/home/ec2-user/NPB3.3-MPI# make FT NPROCS=4 CLASS=A
=====
=      NAS Parallel Benchmarks 3.3      =
=      MPI/F77/C                        =
=====

cd FT; make NPROCS=4 CLASS=A
make[1]: Entering directory `/home/ec2-user/NPB3.3-MPI/FT'
make[2]: Entering directory `/home/ec2-user/NPB3.3-MPI/sys'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/home/ec2-user/NPB3.3-MPI/sys'
../sys/setparams FT 4 A
mpif77 -c -I/usr/local/include -O FT.f
mpif77 -O -o ../bin/FT.A.4 FT.o ../common/randi4.o
../common/print_results.o ../common/timers.o -L/usr/local/lib -lmpi
make[1]: Leaving directory `/home/ec2-user/NPB3.3-MPI/FT'

root@master:/home/ec2-user/NPB3.3-MPI# mpiexec -host master, node001,
node002, node003 bin/ft.A.4 > output/ft.A.4_3.txt
```

EP:

```
root@master:/home/ec2-user/NPB3.3-MPI# make EP NPROCS=4 CLASS=A
=====
=      NAS Parallel Benchmarks 3.3      =
=      MPI/F77/C                        =
=====

cd EP; make NPROCS=4 CLASS=A
make[1]: Entering directory `/home/ec2-user/NPB3.3-MPI/EP'
make[2]: Entering directory `/home/ec2-user/NPB3.3-MPI/sys'
make[2]: Nothing to be done for `all'.
make[2]: Leaving directory `/home/ec2-user/NPB3.3-MPI/sys'
../sys/setparams EP 4 A
mpif77 -c -I/usr/local/include -O EP.f
mpif77 -O -o ../bin/EP.A.4 EP.o ../common/randi4.o
../common/print_results.o ../common/timers.o -L/usr/local/lib -lmpi
make[1]: Leaving directory `/home/ec2-user/NPB3.3-MPI/EP'

root@master:/home/ec2-user/NPB3.3-MPI# mpiexec -host master, node001,
node002, node003 bin/ep.A.4 > output/ep.A.4_3.txt
```

### Windows Azure Firewall Configuration for MPI Communication:

```
PS C:\approot> hpcpack create C:\approot\benchmarks.zip
C:\approot\benchmarks
PS C:\approot>clusrun /nodegroup:computenode hpcsync
```

```

PS D:\Users\sinadmin> clusrun /nodegroup:ComputeNode hpcfwutil register
IOR.exe C:\Resources\Directory\bbc7bb0ba58942cdb

9c6785d69c92464.ComputeNode.Microsoft.Hpc.Azure.LocalStorage.Applicatio
n\benchmarks\2012-05-29T232012.000000Z\IOR.exe

----- COMPUTENODE8 returns 0 -----
-----
Successfully registered application IOR.exe

----- COMPUTENODE7 returns 0 -----
-----
Successfully registered application IOR.exe

----- COMPUTENODE6 returns 0 -----
-----
Successfully registered application IOR.exe

----- COMPUTENODE5 returns 0 -----
-----
Successfully registered application IOR.exe

----- COMPUTENODE4 returns 0 -----
-----
Successfully registered application IOR.exe

----- COMPUTENODE3 returns 0 -----
-----
Successfully registered application IOR.exe

----- COMPUTENODE2 returns 0 -----
-----
Successfully registered application IOR.exe

----- COMPUTENODE1 returns 0 -----
-----
Successfully registered application IOR.exe

----- Summary -----
8 Nodes succeeded
0 Nodes failed
PS D:\Users\sinadmin>

```

## APPENDIX C

### Sample Results

Sample Result for STREAM benchmark run on a four-node cluster:

STREAM version \$Revision: 5.8 \$

-----  
This system uses 8 bytes per DOUBLE PRECISION word.  
-----

Array size = 2000000, Offset = 0  
Total memory required = 45.8 MB.  
Each test is run 10 times, but only  
the \*best\* time for each is used.  
-----

Printing one line per active thread....  
-----

Your clock granularity/precision appears to be 1 microseconds.  
Each test below will take on the order of 5850 microseconds.  
(= 5850 clock ticks)

Increase the size of the arrays if this shows that you are  
not getting at least 20 clock ticks per test.  
-----

WARNING -- The above is only a rough guideline.  
For best results, please be sure you know the precision of  
your system timer.  
-----

Function	Rate (MB/s)	Avg time	Min time	Max time
Copy:	3699.8023	0.0091	0.0086	0.0107
Scale:	3563.4602	0.0094	0.0090	0.0107
Add:	3754.6921	0.0134	0.0128	0.0155
Triad:	3775.6051	0.0134	0.0127	0.0154

-----

Solution Validates  
-----

No. of nodes 4; nodes with errors: 0  
Minimum Copy MB/s 3146.88  
Average Copy MB/s 3313.54  
Maximum Copy MB/s 3699.80  
Minimum Scale MB/s 3109.27  
Average Scale MB/s 3245.54  
Maximum Scale MB/s 3563.46  
Minimum Add MB/s 3154.40  
Average Add MB/s 3334.88  
Maximum Add MB/s 3754.69  
Minimum Triad MB/s 3170.40  
Average Triad MB/s 3345.70  
Maximum Triad MB/s 3775.61

## Sample Result for IOR run on a four-node cluster:

IOR-2.10.3: MPI Coordinated Test of Parallel I/O

Run began: Wed Jun 6 16:12:19 2012  
Command line used: ./IOR -b lg -t 4m  
Machine: Linux master

### Summary:

api = POSIX  
test filename = testFile  
access = single-shared-file  
ordering in a file = sequential offsets  
ordering inter file = no tasks offsets  
clients = 4 (1 per node)  
repetitions = 1  
xfersize = 4 MiB  
blocksize = 1 GiB  
aggregate filesize = 4 GiB

Operation (OPs)	Max (MiB) Mean (OPs)	Min (MiB) Std Dev	Mean (MiB) Mean (s)	Std Dev	Max (OPs)	Min
write	345.24	345.24	345.24	0.00	86.31	
86.31	86.31	0.00	11.86424	EXCEL		
read	5922.35	5922.35	5922.35	0.00	1480.59	
1480.59	1480.59	0.00	0.69162	EXCEL		

Max Write: 345.24 MiB/sec (362.01 MB/sec)  
Max Read: 5922.35 MiB/sec (6210.04 MB/sec)

Run finished: Wed Jun 6 16:12:32 2012

## Sample Result for NBP-CG run on a four-node cluster:

NAS Parallel Benchmarks 3.3 -- CG Benchmark

Size : 14000  
Iterations : 15  
Number of active processes : 4  
Number of nonzeros per row : 11  
Eigenvalue shift : .200E+02

iteration	r	zeta
1	0.30634143529489E-12	19.9997581277040
2	0.31096276403002E-14	17.1140495745506
3	0.30804037245735E-14	17.1296668946143
4	0.31368886171027E-14	17.1302113581193
5	0.30931762620174E-14	17.1302338856353
6	0.30711211120903E-14	17.1302349879482
7	0.30014434726280E-14	17.1302350498916
8	0.30091464390590E-14	17.1302350537510
9	0.30845738922029E-14	17.1302350540101



10	0.30464804270749E-14	17.1302350540284
11	0.30356703468820E-14	17.1302350540298
12	0.30110387739490E-14	17.1302350540299
13	0.29937783924423E-14	17.1302350540299
14	0.30298504149112E-14	17.1302350540299
15	0.30223982636897E-14	17.1302350540299

Benchmark completed

VERIFICATION SUCCESSFUL

Zeta is 0.1713023505403E+02  
Error is 0.5226337199892E-13

CG Benchmark Completed.

Class	=	A
Size	=	14000
Iterations	=	15
Time in seconds	=	4.12
Total processes	=	4
Compiled procs	=	4
Mop/s total	=	363.38
Mop/s/process	=	90.84
Operation type	=	floating point
Verification	=	SUCCESSFUL
Version	=	3.3
Compile date	=	20 Apr 2012

Compile options:

MPIF77	=	mpif77
FLINK	=	\$(MPIF77)
FMPI_LIB	=	(none)
FMPI_INC	=	-I/usr/local/include
FFLAGS	=	-O
FLINKFLAGS	=	-O
RAND	=	randi8

Please send the results of this run to:

NPB Development Team  
Internet: npb@nas.nasa.gov

If email is not available, send this to:

MS T27A-1  
NASA Ames Research Center  
Moffett Field, CA 94035-1000

Fax: 650-604-3957

## Sample Result for NBP-FT run on a four-node cluster:

NAS Parallel Benchmarks 3.3 -- FT Benchmark

No input file input ft.data. Using compiled defaults

```
Size           : 256x 256x 128
Iterations     : 6
Number of processes : 4
Processor array : 1x 4
Layout type    : 1D
```

```
T = 1    Checksum = 5.046735008193D+02    5.114047905510D+02
T = 2    Checksum = 5.059412319734D+02    5.098809666433D+02
T = 3    Checksum = 5.069376896287D+02    5.098144042213D+02
T = 4    Checksum = 5.077892868474D+02    5.101336130759D+02
T = 5    Checksum = 5.085233095391D+02    5.104914655194D+02
T = 6    Checksum = 5.091487099959D+02    5.107917842803D+02
```

Result verification successful

class = A

FT Benchmark Completed.

```
Class          = A
Size           = 256x 256x 128
Iterations     = 6
Time in seconds = 11.02
Total processes = 4
Compiled procs = 4
Mop/s total    = 647.52
Mop/s/process  = 161.88
Operation type = floating point
Verification    = SUCCESSFUL
Version        = 3.3
Compile date   = 21 Apr 2012
```

Compile options:

```
MPIF77        = mpif77
FLINK         = $(MPIF77)
FMPI_LIB      = (none)
FMPI_INC      = -I/usr/local/include
FFLAGS       = -O
FLINKFLAGS    = -O
RAND          = randi8
```

Please send the results of this run to:

NPB Development Team

Internet: npb@nas.nasa.gov

If email is not available, send this to:

MS T27A-1  
NASA Ames Research Center  
Moffett Field, CA 94035-1000

Fax: 650-604-3957

### Sample Result for NBP-EP run on a four-node cluster:

NAS Parallel Benchmarks 3.3 -- EP Benchmark

Number of random numbers generated: 536870912  
Number of active processes: 4

#### EP Benchmark Results:

CPU Time = 11.9713  
N = 2<sup>28</sup>  
No. Gaussian Pairs = 210832767.  
Sums = -4.295875165634796D+03 -1.580732573678614D+04

#### Counts:

0	98257395.
1	93827014.
2	17611549.
3	1110028.
4	26536.
5	245.
6	0.
7	0.
8	0.
9	0.

EP Benchmark Completed.

Class	=	A
Size	=	536870912
Iterations	=	0
Time in seconds	=	11.97
Total processes	=	4
Compiled procs	=	4
Mop/s total	=	44.85
Mop/s/process	=	11.21
Operation type	=	Random numbers generated
Verification	=	SUCCESSFUL
Version	=	3.3
Compile date	=	21 Apr 2012

#### Compile options:

MPIF77	=	mpif77
FLINK	=	\$(MPIF77)
FMPI_LIB	=	(none)
FMPI_INC	=	-I/usr/local/include

FFLAGS = -O  
FLINKFLAGS = -O  
RAND = randi8

Please send the results of this run to:

NPB Development Team  
Internet: npb@nas.nasa.gov

If email is not available, send this to:

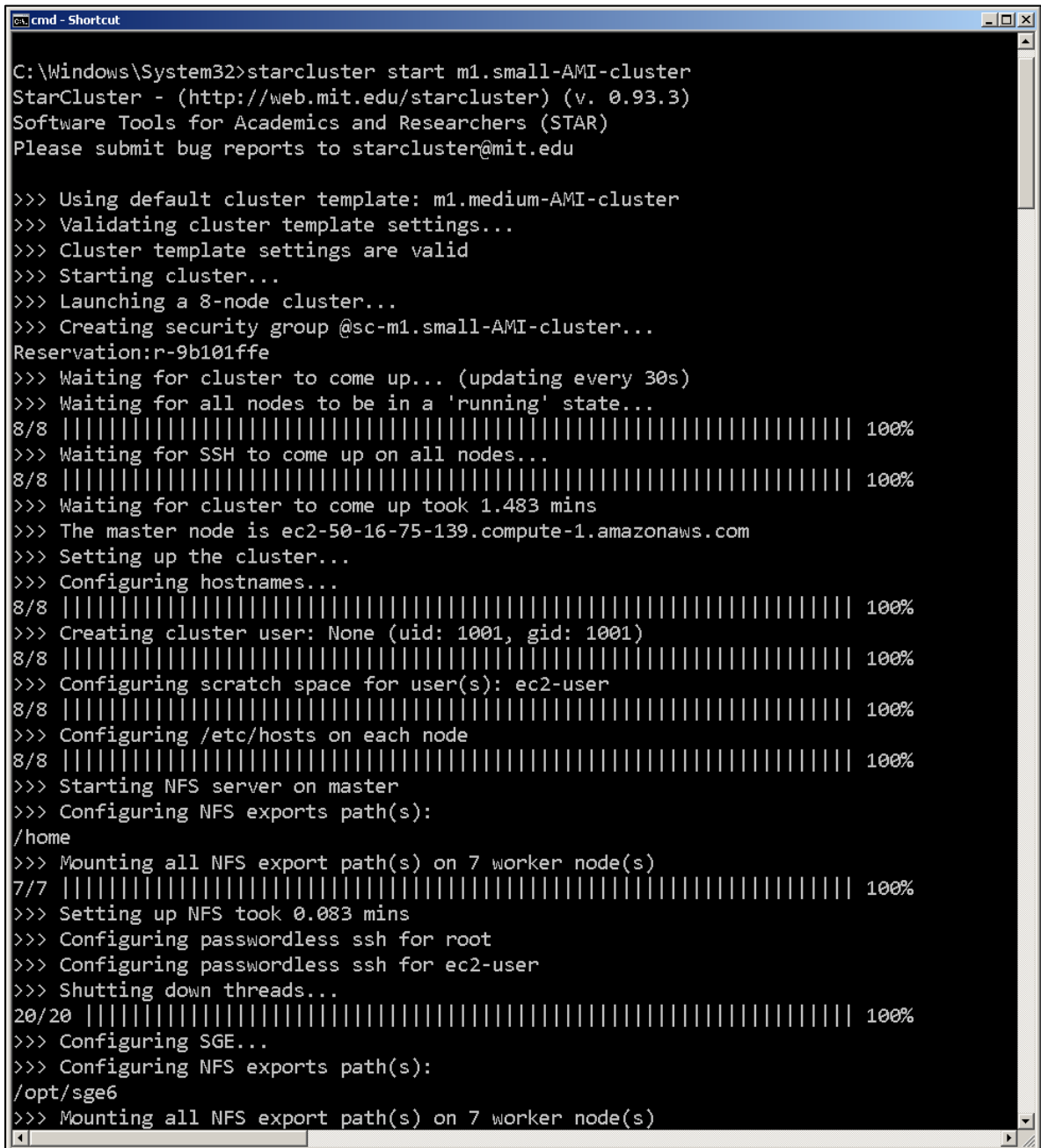
MS T27A-1  
NASA Ames Research Center  
Moffett Field, CA 94035-1000

Fax: 650-604-3957

## APPENDIX D

### EC2 Screenshots

Example of successful start of an eight-node cluster:



```
cmd - Shortcut
C:\Windows\System32>starcluster start m1.small-AMI-cluster
StarCluster - (http://web.mit.edu/starcluster) (v. 0.93.3)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

>>> Using default cluster template: m1.medium-AMI-cluster
>>> Validating cluster template settings...
>>> Cluster template settings are valid
>>> Starting cluster...
>>> Launching a 8-node cluster...
>>> Creating security group @sc-m1.small-AMI-cluster...
Reservation:r-9b101ffe
>>> Waiting for cluster to come up... (updating every 30s)
>>> Waiting for all nodes to be in a 'running' state...
8/8 ||| 100%
>>> Waiting for SSH to come up on all nodes...
8/8 ||| 100%
>>> Waiting for cluster to come up took 1.483 mins
>>> The master node is ec2-50-16-75-139.compute-1.amazonaws.com
>>> Setting up the cluster...
>>> Configuring hostnames...
8/8 ||| 100%
>>> Creating cluster user: None (uid: 1001, gid: 1001)
8/8 ||| 100%
>>> Configuring scratch space for user(s): ec2-user
8/8 ||| 100%
>>> Configuring /etc/hosts on each node
8/8 ||| 100%
>>> Starting NFS server on master
>>> Configuring NFS exports path(s):
/home
>>> Mounting all NFS export path(s) on 7 worker node(s)
7/7 ||| 100%
>>> Setting up NFS took 0.083 mins
>>> Configuring passwordless ssh for root
>>> Configuring passwordless ssh for ec2-user
>>> Shutting down threads...
20/20 ||| 100%
>>> Configuring SGE...
>>> Configuring NFS exports path(s):
/opt/sge6
>>> Mounting all NFS export path(s) on 7 worker node(s)
```

Example of successful start of an eight-node cluster:

```
cmd - Shortcut
>>> Mounting all NFS export path(s) on 7 worker node(s)
7/7 | 100%
>>> Setting up NFS took 0.047 mins
>>> Installing Sun Grid Engine...
7/7 | 100%
>>> Creating SGE parallel environment 'orte'
8/8 | 100%
>>> Adding parallel environment 'orte' to queue 'all.q'
>>> Shutting down threads...
20/20 | 100%
>>> Running plugin mpich2
>>> Creating MPICH2 hosts file
>>> Configuring MPICH2 profile
8/8 | 100%
>>> Setting MPICH2 as default MPI on all nodes
8/8 | 100%
>>> MPICH2 is now ready to use
>>> Use mpicc, mpif90, mpirun, etc. to compile and run your MPI apps
>>> Configuring cluster took 1.392 mins
>>> Starting cluster took 2.905 mins

The cluster is now ready to use. To login to the master node
as root, run:

    $ starcluster sshmaster m1.small-AMI-cluster

If you're having issues with the cluster you can reboot the
instances and completely reconfigure the cluster from
scratch using:

    $ starcluster restart m1.small-AMI-cluster

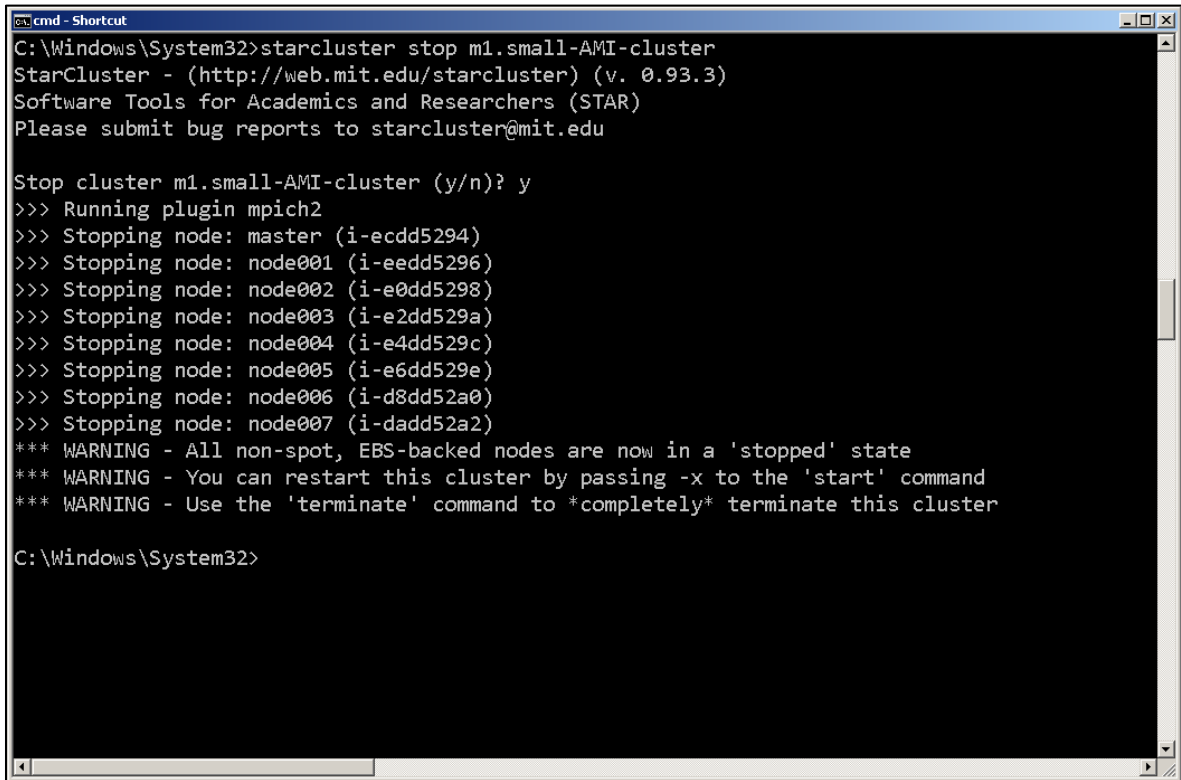
When you're finished using the cluster and wish to terminate
it and stop paying for service:

    $ starcluster terminate m1.small-AMI-cluster

Alternatively, if the cluster uses EBS instances, you can
use the 'stop' command to shutdown all nodes and put them
into a 'stopped' state preserving the EBS volumes backing
the nodes:

    $ starcluster stop m1.small-AMI-cluster
```

Stopping a cluster:



```
C:\Windows\System32>starcluster stop m1.small-AMI-cluster
StarCluster - (http://web.mit.edu/starcluster) (v. 0.93.3)
Software Tools for Academics and Researchers (STAR)
Please submit bug reports to starcluster@mit.edu

Stop cluster m1.small-AMI-cluster (y/n)? y
>>> Running plugin mpich2
>>> Stopping node: master (i-ecdd5294)
>>> Stopping node: node001 (i-eedd5296)
>>> Stopping node: node002 (i-e0dd5298)
>>> Stopping node: node003 (i-e2dd529a)
>>> Stopping node: node004 (i-e4dd529c)
>>> Stopping node: node005 (i-e6dd529e)
>>> Stopping node: node006 (i-d8dd52a0)
>>> Stopping node: node007 (i-dadd52a2)
*** WARNING - All non-spot, EBS-backed nodes are now in a 'stopped' state
*** WARNING - You can restart this cluster by passing -x to the 'start' command
*** WARNING - Use the 'terminate' command to *completely* terminate this cluster

C:\Windows\System32>
```

AWS Management Console showing a cluster of six nodes:

Name	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status	Monitoring	Security Groups	Key Pair Name
<input checked="" type="checkbox"/>	master_m1medium	i-3272eb4b	ebs	m1.medium	running	2/2 checks passed	none	basic	@sc-m1.medium-AMI-cluster	winkey
<input type="checkbox"/>	node001	i-3472eb4d	ebs	m1.medium	running	2/2 checks passed	none	basic	@sc-m1.medium-AMI-cluster	winkey
<input type="checkbox"/>	node002	i-3672eb4f	ebs	m1.medium	running	2/2 checks passed	none	basic	@sc-m1.medium-AMI-cluster	winkey
<input type="checkbox"/>	node003	i-2872eb51	ebs	m1.medium	running	2/2 checks passed	none	basic	@sc-m1.medium-AMI-cluster	winkey
<input type="checkbox"/>	node004	i-2a72eb53	ebs	m1.medium	running	2/2 checks passed	none	basic	@sc-m1.medium-AMI-cluster	winkey
<input type="checkbox"/>	node005	i-2c72eb55	ebs	m1.medium	running	2/2 checks passed	none	basic	@sc-m1.medium-AMI-cluster	winkey

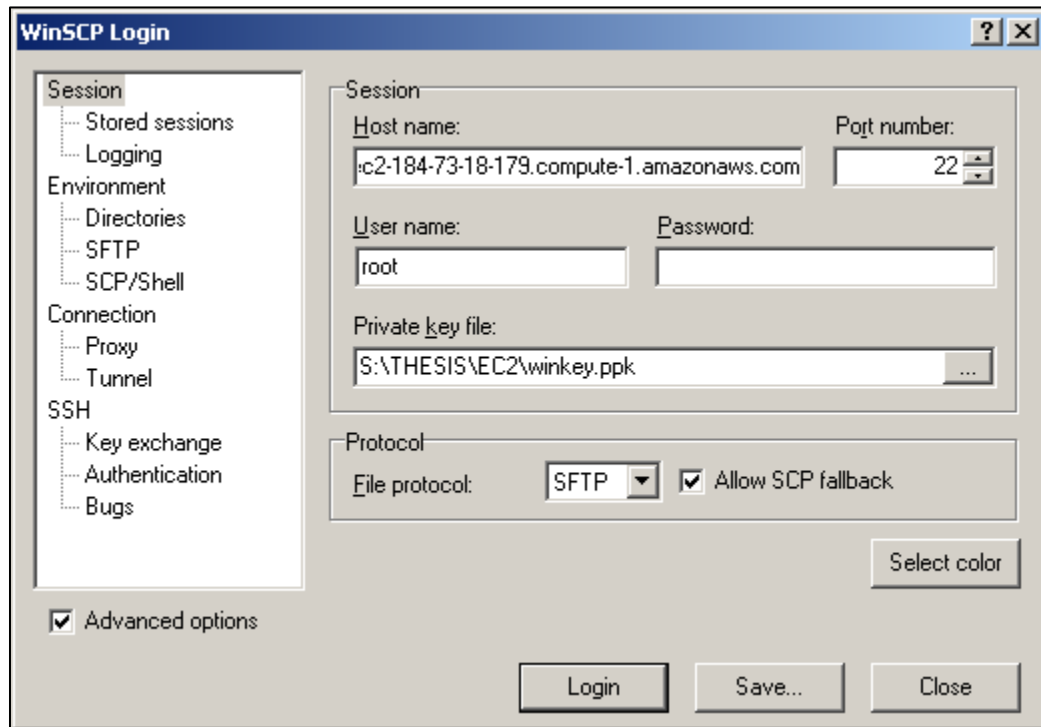
**EC2 Instance: master\_m1medium (i-3272eb4b)**  
ec2-23-22-30-90.compute-1.amazonaws.com

**Description:** starcluster-base-ubuntu-11.10-x86\_64 (ami-999d49f0)  
**AMI:** starcluster-base-ubuntu-11.10-x86\_64 (ami-999d49f0)  
**Zone:** us-east-1a  
**Type:** m1.medium  
**Scheduled Events:** No scheduled events  
**VPC ID:** -  
**Source/Dest. Check:** -  
**Placement Group:** -  
**RAM Disk ID:** -

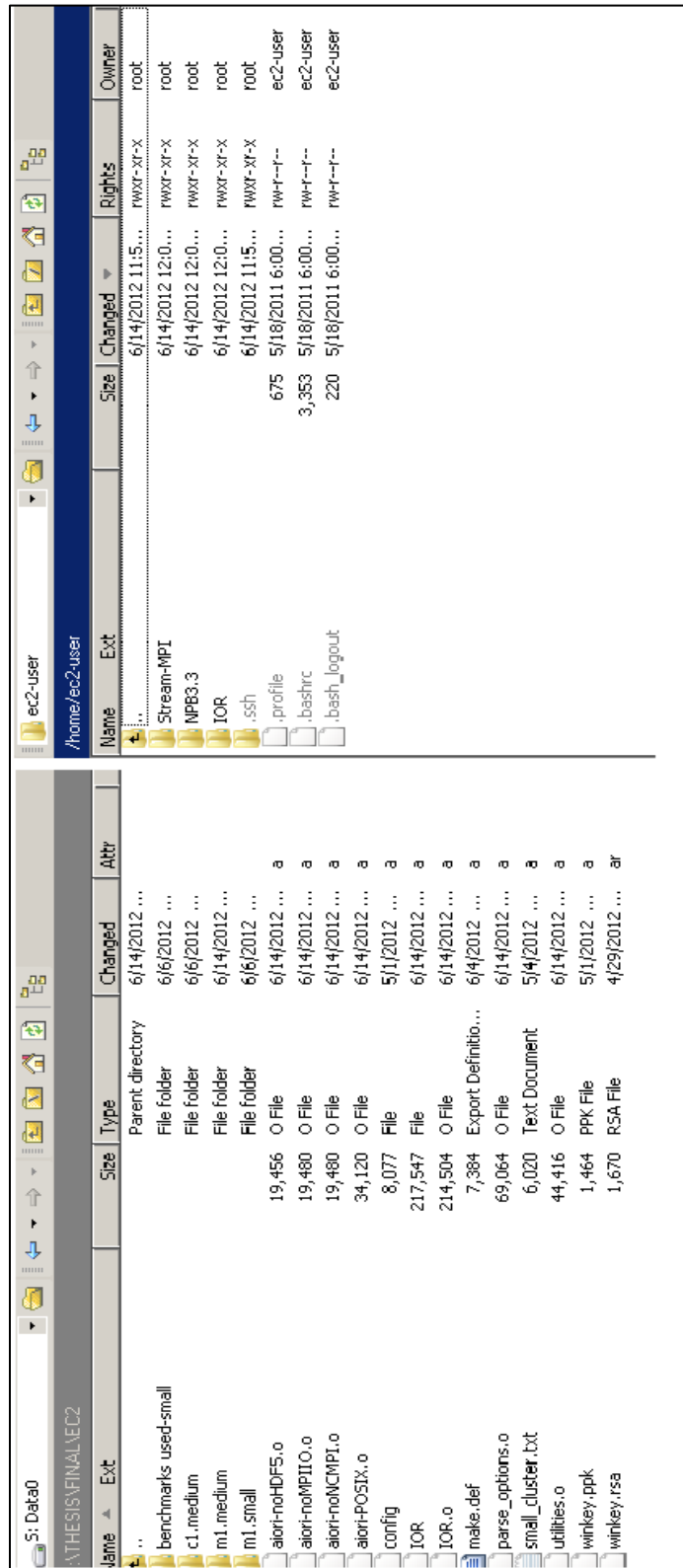
**Alarm Status:** none  
**Security Groups:** @sc-m1.medium-AMI-cluster, view rules  
**State:** running  
**Owner:** 390135667176  
**Subnet ID:** -  
**Virtualization:** paravirtual  
**Reservation:** r-042ff761  
**Platform:** -



WinSCP session screen connecting to master node on m1.medium as root:



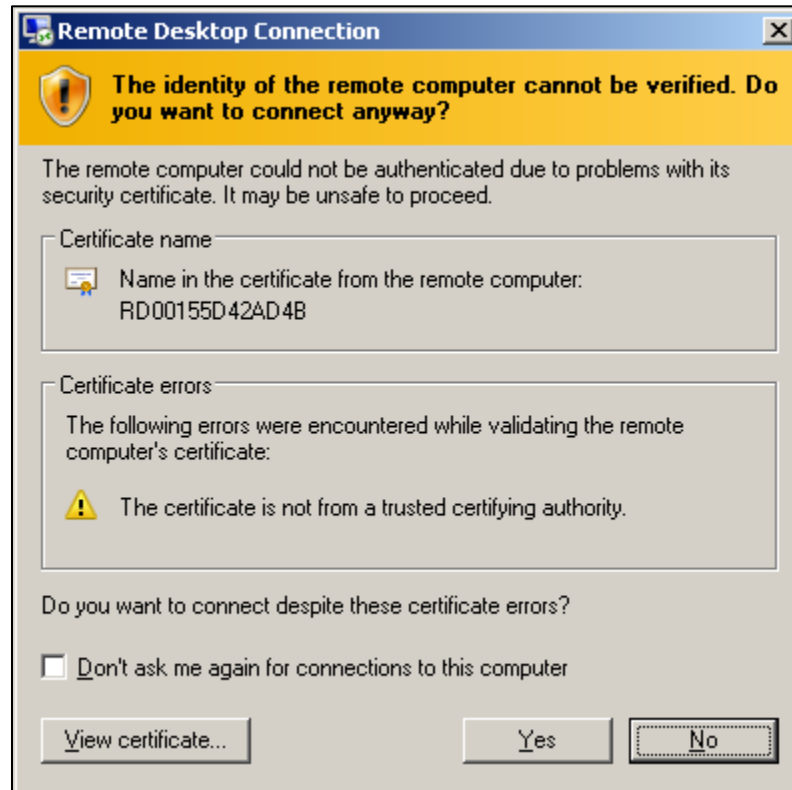
WinSCP screen with files in local system on left and master node on the right:



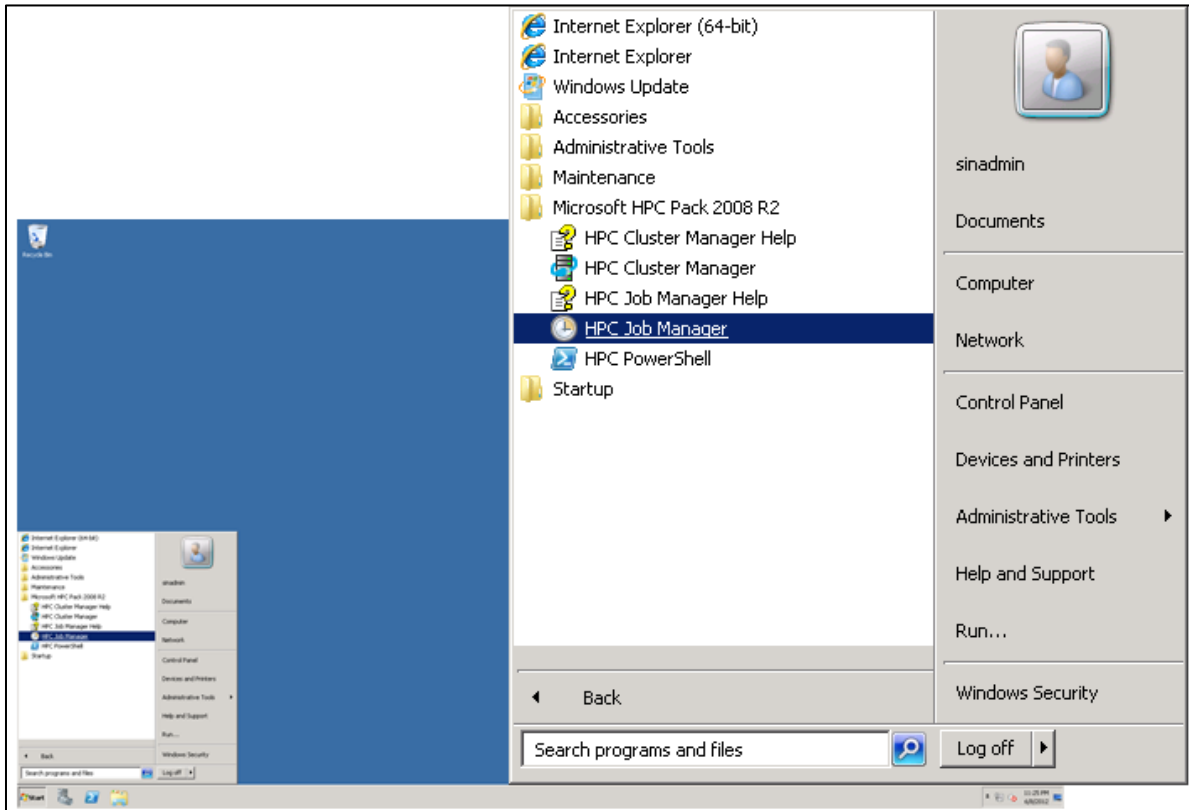
## APPENDIX E

### Azure Screenshots

An RDP connection to the HeadNode:



Desktop of the HeadNode of a Cluster:



## Windows Azure HPC scheduler deployment and eight-node cluster:

```
Administrator C:\Windows\System32\WindowsPowerShell\cmd powershell.exe
Specify a value for the Windows Azure HPC Scheduler administrator user name: sinadmin
Specify a value for the Windows Azure HPC Scheduler administrator password: *****
Specify a value for the Windows Azure HPC Scheduler administrator password (again): *****
Packaging Windows Azure HPC Scheduler
Windows (R) Azure (TM) Packaging Tool version 1.6.0.0
for Microsoft (R) .NET Framework 3.5
Copyright (c) Microsoft Corporation. All rights reserved.

cspack.exe: warning : Cloudservices65 : This build option is obsolete, all packages are unencrypted by default.
Examine the DB existence...
The database seems to be in use. Please confirm you want to override the existing database by rerunning the initializedDB.exe with -f opt
The HPC tables exist.

Overwrite database for this deployment
The database appears to be in use. Overwrite?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] suspend [?] Help (default is "Y"): y
Examine the DB existence...
The HPC tables exist.
Initializing DB schema...
Uploading SchedulerSchema.sql
Uploading SchedulerSP.sql
Uploading SchedulerSPAzure.sql
Uploading CreateHPCDISCDatabase.sql
Successfully installed schema. Elapsed time: 69.6089814
Adding node ComputeNode1 with 2 cores on worker role HeadNode, initially offline
Adding node ComputeNode2 with 2 cores on worker role ComputeNode, initially online
Adding node ComputeNode3 with 2 cores on worker role ComputeNode, initially online
Adding node ComputeNode4 with 2 cores on worker role ComputeNode, initially online
Adding node ComputeNode5 with 2 cores on worker role ComputeNode, initially online
Adding node ComputeNode6 with 2 cores on worker role ComputeNode, initially online
Adding node ComputeNode7 with 2 cores on worker role ComputeNode, initially online
Adding node ComputeNode8 with 2 cores on worker role ComputeNode, initially online
Setup the storage tables...
Finished.
Deploying (cspkg: s:\THEESIS\FINAL\AZURE\medium\WAHSPowershellDeployment\Code\WindowsAzureSchedulerDeployment\Package\output.cspkg, cscfg:
Upload Completed. (Time: 6.272008738333333mins Speed: 275.480302842467kb/s)
Deployment is ready!
PS S:\THEESIS\FINAL\AZURE\medium\WAHSPowershellDeployment\Code>
```

Windows Azure Management Portal with an eight-node cluster:

The screenshot displays the Windows Azure Management Portal interface for an eight-node cluster. The top navigation bar includes the 'Windows Azure Platform' logo and several management options: New Hosted Service, New Staging Deployment, New Production Deployment, Upgrade, Configure, Delete, Start, Stop, Swap, VTP, Configure OS, Reboot, Reimage, Connect, Enable, Configure, and Connect. The main content area shows a navigation pane on the left with options like Deployment Health, Affinity Groups, Management Certificates, Hosted Services (1), Storage Accounts (1), User Management, and CDN. The central table provides a detailed view of the cluster's configuration and status.

Name	Type	Status	Environment	Cores used	Cores quota	Country/Region	Size
3-Month Free Trial	Subscription	Active		20	20	North Central US	
Windows Azure HPC Scheduler Service	Hosted Service	Created		20	20	North Central US	
Certificates							
AzureMediumCis.cloudapp.net	Service Certificate	Created					
Windows Azure HPC Scheduler Script Deployment	Deployment	Ready	Production	20	20		
ComputeNode	Role	Ready	Production				Medium
ComputeNode_IN_0	Instance	Ready	Production				Medium
ComputeNode_IN_1	Instance	Ready	Production				Medium
ComputeNode_IN_2	Instance	Ready	Production				Medium
ComputeNode_IN_3	Instance	Ready	Production				Medium
ComputeNode_IN_4	Instance	Ready	Production				Medium
ComputeNode_IN_5	Instance	Ready	Production				Medium
ComputeNode_IN_6	Instance	Ready	Production				Medium
ComputeNode_IN_7	Instance	Ready	Production				Medium
HeadNode	Role	Ready	Production				Medium
HeadNode_IN_0	Instance	Ready	Production				Medium
FrontEnd	Role	Ready	Production				Medium
FrontEnd_IN_0	Instance	Ready	Production				Medium

Finished jobs in HPC Job Manager:

Cluster AZURES-MALLCLUS - HPC 2008 R2 Job Manager									
File View Actions Options Help									
<span>Back</span> <span>Forward</span> <span>Navigation Pane</span> <span>Actions</span> <span>Filter: Owner</span> <span>Submit time</span> <span>Project name</span> <span>Search: Job name</span>									
Job Management									
Finished (39)									
Job ID	Job Name	State	Owner	Progress	Submit Time				
79	\$_ep.A.8	Finished	snadmin	100%	6/1/2012 5:20:59 PM				
78	\$_ep.A.6	Finished	snadmin	100%	6/1/2012 5:19:52 PM				
77	\$_ep.A.4	Finished	snadmin	100%	6/1/2012 5:18:25 PM				
75	\$_ep.A.2	Finished	snadmin	100%	6/1/2012 5:15:01 PM				
74	\$_ep.A.1	Finished	snadmin	100%	6/1/2012 5:13:13 PM				
72	\$_ep.A.2	Finished	snadmin	100%	6/1/2012 5:11:39 PM				
71	\$_ep.A.1	Finished	snadmin	100%	6/1/2012 5:10:57 PM				
70	\$_ft.A.8	Finished	snadmin	100%	6/1/2012 5:09:31 PM				
69	\$_ft.A.2	Finished	snadmin	100%	6/1/2012 5:07:13 PM				
68	\$_ft.A.4	Finished	snadmin	100%	6/1/2012 5:04:55 PM				
67	\$_ft.A.4	Finished	snadmin	100%	6/1/2012 5:01:28 PM				
66	\$_ft.A.1	Finished	snadmin	100%	6/1/2012 4:59:22 PM				
65	\$_cg.A.8	Finished	snadmin	100%	6/1/2012 4:55:11 PM				
64	\$_cg.A.4	Finished	snadmin	100%	6/1/2012 4:54:03 PM				
48	\$_cg.A.4	Finished	snadmin	100%	6/1/2012 4:40:43 PM				
47	\$_cg.A.2	Finished	snadmin	100%	6/1/2012 4:39:02 PM				
46	\$_cg.A.2	Finished	snadmin	100%	6/1/2012 4:38:31 PM				
43	\$_cg.A.2	Finished	snadmin	100%	6/1/2012 4:33:03 PM				

## VITA

Sindhu Mani earned the Bachelor of Technology degree in Information Technology from Anna University, Chennai, India, in 2007 and expects to receive her Master of Science in Computer Science with Information Systems as major from the University of North Florida in Fall 2012. Sindhu worked as a Project Engineer for two years at Wipro Technologies for JP Morgan Chase as client, in Bangalore, India using Java/ J2EE technologies prior to starting graduate studies. Sindhu worked as a Research Assistant at the University of North Florida for Dr. Sanjay P. Ahuja. During this period she also had three journal publications related to cloud computing. She was also inducted into Upsilon Pi Epsilon and Phi Kappa Phi honor societies because of her academic excellence. Sindhu's academic work included use of C#, ASP.Net, Java, and SQL. She is also a Sun Certified Java Programmer.

Sindhu aspires to work for a technology company that is involved in developing mobile and cloud computing based applications. She is married and enjoys outdoor fun and adventurous activities such as travelling, camping and sky-diving.