Hawk-i HPC Cloud Benchmark Tool

Visakh Muraleedharan

August 24, 2012

MSc in High Performance Computing
The University of Edinburgh
Year of Presentation: 2012
Abstract

Scientific computing unravels the mysteries of science by constructing mathematical models and numerical algorithms to simulate scientific scenarios. This requires massive computational power and High-Performance Computing (HPC) solutions. Cluster and grid have been the answer for massive computing needs for a long time. These HPC solutions are often hard and expensive to setup, maintain and use. Cloud computing is a model of delivering the compute infrastructure in which computation and storage can be dynamically provisioned on a pay as you go model. Using a vendor cloud service like Amazon Web Service (AWS) can significantly reduce the effort to access these on-demand high performance resources.

In this work we create a live benchmark tool (Hawk-i) to study how we can usefully compare the suitability of Amazon EC2 instance types for scientific applications. We studied the memory scalability and performance of Spectral methods and N-Body Methods in a cluster built on top of cloud infrastructure. We have done a detailed analysis of the performance results and found some possible solutions to get better performance out of the cloud.

Hawk-i tool proved to be very useful to generate large amount of benchmark data which made this study possible. With the simple user interface, it is also very easy to use service for non-technical users who have little knowledge of cloud or cluster environments.
Contents

Chapter 1 Introduction ...................................................................................................1

Chapter 2 Background ..................................................................................................3

2.1 Cloud computing ....................................................................................................3

  2.1.1 Cloud Computing Deployment models ........................................................3

  2.1.2 Cloud computing service models ....................................................................4

2.2 HPC in cloud ..........................................................................................................5

2.3 Amazon Web Services and EC2 ..........................................................................6

  2.3.1 Amazon Instance types ................................................................................7

  2.3.2 Amazon Machine Image ................................................................................8

  2.3.3 EC2 Pricing ....................................................................................................9

2.4 Parallel Applications and 13 Motifs ......................................................................10

  2.4.1 Spectral Methods .........................................................................................11

  2.4.2 N-Body Methods .........................................................................................12

  2.4.3 Ping Pong .....................................................................................................13

Chapter 3 Experimental design .....................................................................................14

3.1 Design Decisions and Assumptions ........................................................................14

  3.1.1 Instances used ..............................................................................................14

  3.1.2 Starcluster ....................................................................................................15

  3.1.3 AMI used .....................................................................................................16
3.1.4 Benchmarks ................................................................. 16
3.1.5 System interface .......................................................... 18
3.1.6 Execution cost .............................................................. 18
3.2 System Design ............................................................... 19
  3.2.1 Architecture ............................................................... 19
  3.2.2 Control Sequence ......................................................... 21
  3.2.3 Database Design .......................................................... 23
3.3 Implementation .................................................................. 25
  3.3.1 Environment setup ........................................................ 25
  3.3.2 Hawk-i Implementation .................................................. 26
  3.3.3 Benchmark Implementation .............................................. 30
Chapter 4 Results and Analysis ............................................. 33
  4.1 Benchmark results ........................................................... 33
    4.1.1 Spectral Methods (FFT Benchmark) ............................... 33
    4.1.2 N-Body Methods (MD Benchmark) ................................. 37
    4.1.3 Network Test (Ping Pong Benchmark) .............................. 38
  4.2 Detailed Analysis of results ............................................... 40
    4.2.1 Execution Time .......................................................... 40
    4.2.2 Memory scaling ........................................................ 42
    4.2.3 Stability of Performance ............................................... 44
    4.2.4 Cost effective solution ............................................... 45
  4.3 Analysis of Hawk-i .......................................................... 46
Chapter 5 Conclusions and Future Work .................................. 48
  5.1 Conclusions ................................................................ 48
5.2 Future Work

Appendix A  Hawk-i User Manual

Appendix B  AMI Details

Appendix C  Manager Instance Configuration

Appendix D  Benchmark data

Appendix E  EC2 API Tools Configuration

Appendix F  Source Code and Repository

References
List of Tables

Table 1: Amazon EC2 Instance types ................................................................. 7
Table 2: Pricing per instance-hour consumed for each EC2 instance ..................... 9
Table 3: CPU Clock speed comparison in m1.small instances .............................. 41
Table 4: CPU Cache Size comparison of EC2 Instances ..................................... 42
Table 5: t1.micro consecutive benchmark comparison with m1.small .................... 57
Table 6: EC2 Instances Start-up time ................................................................. 57
Table 7: Time scaling for FFT benchmark .......................................................... 58
Table 8: Standard instances memory limit and execution time .............................. 58
Table 9: High memory instances memory limit and execution time ....................... 58
Table 10: High CPU instances memory limit and execution time ......................... 58
Table 11: Cluster Compute instances memory limit and execution time ............... 59
Table 12: MD benchmark results ...................................................................... 59
Table 13: Ping pong benchmark results ............................................................. 60
Table 14: FFT problem size and memory required .............................................. 60
Table 15: Available memory in EC2 Instances ................................................... 61
Table 16: Stability of results ............................................................................. 61
List of Figures

Figure 1: Google trends comparison of search terms (grid, cluster and cloud) ..........6
Figure 2: Amazon Web Services Logo ..............................................................................6
Figure 3: Temperature Chart of the 13 Motifs .................................................................11
Figure 4: Spectral Methods Application Areas ................................................................11
Figure 5: A snapshot from an Ab-initio molecular dynamics simulation .....................13
Figure 6: Performance consistency comparison of EC2 micro instance ......................15
Figure 7: Startup time comparison of two-node EC2 SGE Cluster .............................17
Figure 8: High Level Interaction Diagram .......................................................................19
Figure 9: Manager Instance Interaction Diagram ..........................................................20
Figure 10: Benchmark execution sequence diagram ....................................................22
Figure 11: Entity relationship diagram ...........................................................................23
Figure 12: Directory layout of Hawk-i System ..............................................................26
Figure 13: Hawk-i command line interface ....................................................................27
Figure 14: Live benchmark input submission form .......................................................28
Figure 15: Results log of benchmarks execution ............................................................29
Figure 16: Form to add new benchmarks ......................................................................29
Figure 17: Adding New inputs .......................................................................................30
Figure 18: FFT Execution time scaling in standard Instances ....................................34
Figure 19: FFT execution time and memory limit in Standard Instances ....................34
Figure 20: FFT execution time and memory limit in high memory instances ..........35
Figure 21: FFT execution time and memory limit for high CPU instances............36
Figure 22: FFT execution time and memory limit for cluster compute instances.....37
Figure 23: MD execution time ($2^{16}$) comparison for all EC2 instance types.........38
Figure 24: Time per message (2MB) results from EC2 Ping pong Benchmark.........39
Figure 25: Hawk-i shell execution for CPU test..................................................40
Figure 26: FFT memory requirement comparison to EC2 Standard Instance memory.43
Figure 27: Swap Memory in m1.small..................................................................44
Figure 28: Standard deviation of execution for $2^{16}$ problem size ......................45
Figure 29: Cost Efficiency Calculation for MD benchmark Amazon EC2.............45
Acknowledgements

I would like to thank my project supervisor Mr. Neil Chue Hong for his advice and supervision during this project.

I would also like to thank all the developers of MIT Starcluster project who gave active support to my research and development work in this project.
Chapter 1

Introduction

Cloud computing is becoming widely accepted by the scientific community for high performance computing and data analysis. For example, CERN [1] announced that it will join the Helix Nebula Initiative and this would double its data processing power [2]. The Helix Nebula[3] initiative is a pilot program by the vendor cloud providers and research organizations looking for a breakthrough in European cloud computing industry by carrying out scientific research in the cloud.

There are several well established cloud computing providers like Amazon Web Services who provide affordable cloud computing services to High Performance Computing (HPC) and non-HPC users. These kind of public cloud offerings are very attractive to small/medium research institutes which don’t have sufficient economic support or staff to setup their own cluster or cloud infrastructure. The ‘pay-as-you-go billing model’ of cloud computing services allows these users to leverage high end compute capabilities on-demand with zero infrastructure or maintenance cost.

When it comes to cloud computing, there are several misconceptions [4], one of which is that a commercial cloud is cheaper than operating your own system. Studies[4] show that commercial clouds are not cheap, but cost effective for individual projects that need high requirement bursts or high throughput applications like Monte-Carlo simulations[5] and Bioinformatics[6]. This doesn’t mean that cloud is not useful for other types of applications. The important aspect here is to identify how different algorithms scales in a public cloud before investing a significant amount of time and effort to port the applications to the cloud infrastructure.

In this dissertation, we create a live benchmark tool (Hawk-i) to investigate how different scientific algorithms scale in different Cloud instances. We calculate the execution time and supported problem size in different Amazon EC2 instances for different class of problems in the area of high performance computing.

We classify the benchmarks based on 13 motifs[7] that can represent the computation patterns of all the HPC applications. Our experiment implements two of these patterns, spectral methods and N-body simulation, to test the usability of this live benchmark tool. Since most of the scientific applications are parallel programs, we also benchmarked the message passing capacity of Amazon EC2 instances by using a parallel ping pong program.
Amazon Web Services provides an Elastic Compute Cloud service that allows users to configure virtual machines and use it on-demand. There are different types of virtual machines [Table 1] provided by AWS EC2 with each one having its own unique features. We compare these instances using Hawk-i and try to identify the best instance for a particular problem and problem size.

In cloud computing, the hardware resources are virtualised and shared by different users and this can affect stability of performance. When it comes to scientific computing it is very essential to choose an instance with stable performance. So we also conducted a detailed study of different Amazon Instances with the help of Hawk-i to identify the stability of performance of each EC2 Instance. Using Hawk-i, we also try to find the reason for this difference in performance and ways to get better performance from EC2 instances.

The organisation of the rest of the dissertation is as follows:

Chapter 2: A brief overview of cloud computing and its possibilities in scientific community, Amazon Web Services and classification of parallel applications in scientific computing.

Chapter 3: A discussion of the design decisions and assumptions followed by the complete system design and implementation details of Hawk-i tool. This chapter also discuss the sample benchmarks created and tested in Hawk-i.

Chapter 4: The results obtained using Hawk-i in different Amazon EC2 Instances are reported and analysed with in-depth analysis of the unexpected results. This chapter also discusses how effective Hawk-i was in providing information to choose an EC2 instance type for a scientific problem.

Chapter 5: The conclusion of the dissertation work together with a brief summary of research and analysis followed by some suggestions to improve Hawk-i. A discussion of possible further research in related area is also suggested in this chapter.
Chapter 2

Background

2.1 Cloud computing

In 1960 John McCarthy predicted that "computation may someday be organised as a public utility" [8]. Availability of high capacity networks and low cost hardware as well as the adoption of hardware virtualisation [9] technology has made this prediction come true. These technologies and delivery models called "cloud computing" have enabled a large community of end-users to obtain computing and storage capacity on-demand.

There are several aspects to cloud computing which makes it attractive to users. The most important one is the illusion of infinite computing resources on-demand which reduces the user's effort to plan ahead for resource provisioning. Another feature is that cloud computing allow users to start small in a test environment and grow bigger without any upfront commitments to service providers. Finally, from a pricing point of view, provision to pay only for the computing, storage or network resources that you use makes cloud a cost effective solution for users [10].

2.1.1 Cloud Computing Deployment models

Cloud infrastructure can be deployed in several ways depending on several aspects like cost effectiveness, privacy and security. Four deployment models of cloud computing are:

Public Cloud

When the cloud is available to the general public in a pay as you go manner, it is called a public cloud. Public cloud infrastructure is owned and maintained by vendors that can support very large infrastructure and hence they are also known as vendor clouds. Amazon Web Services[11] which is studied in this project would fall in this category. Some of the other major companies offering cloud services are Rackspace[12], Microsoft[13] and Google[14]. Typically, end-users registered for this service using a credit card and they are provided access to this cloud infrastructure via internet.

Private Cloud
A cloud infrastructure operating for a single organisation is a private cloud. This can be managed by the organisation or by a third party. Typically a private cloud is hosted within the organisation network but they can also be hosted externally. To provide virtualisation to users cloud software stacks like OpenNebula[15], Eucalyptus[16] and OpenStack[17] are used. Private cloud provides the benefits of public clouds listed above while avoiding the issues concerning security of data and performance of public clouds. Even though the organisation has to setup and manage the infrastructure, it can be less expensive than public clouds [4].

Community Cloud

When a private cloud infrastructure is deployed by two or more organisations having similar requirements, it becomes a community cloud. The cost of operating the cloud is shared by the organisations in the community. In community cloud, cost need not directly translate to currency in a community cloud. Usage can be credits based or fixed usage based on the agreement between the organisations.

Hybrid Cloud

To benefit from the advantages of multiple deployment models, the two or more cloud infrastructures can be bound together to form a hybrid cloud. It requires onsite resources accessible without internet that provides flexibility and remote server based infrastructure to provide scalability. Hybrid clouds can be very useful with the implementing concepts like cloud bursting[18]. The software application running on an internal organisational cloud is dynamically ported to a public cloud to address the increase in resource demand. To enable portability of applications in hybrid model, the independent infrastructures should be technology compliant.

2.1.2 Cloud computing service models

Cloud computing services are normally categorized as Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). High performance computing can significantly utilise each of these models. The difference between the service models is based on the abstraction level at which the service is provided to the end user. The end users have the freedom to customise the applications and environment above the abstracted level. For example, in IaaS, a virtual hardware is provided to the end user and the user then decides on the operating system and applications to be installed on the system. We will discuss each of these service models with existing examples in the public clouds to understand their features.

Infrastructure as a Service

In Infrastructure as a Service model, cloud provides virtual equipment including hardware, storage and networking components. The service provider is responsible for creating, running, and maintaining it. In public cloud, the client typically pays on a per-use basis for use of the equipment. Today, the most widely used IaaS provider is Amazon Web Services. Amazon provides different types if Infrastructures for different
pricing. Amazon Simple Storage Service (S3) and Elastic Block Storage (EBS) are methods of data storage. Being a key-based storage which is highly scalable, Amazon S3 is capable of handling data integrity and fault tolerance. Amazon EBS storage provides virtual block storage devices that can easily be associated with elastic compute instances to persistently store data. S3 charges for the amount of data transferred to and from the device and the number of metadata operations performed. Both EBS and S3 charges for data stored per month, but there is no charge for data transferred within an Availability Zone[19]. Data transferred between EC2 instances and S3 located in different Availability Zones in the same region will be charged Regional Data Transfer and data transferred between devices in different regions will be charged on both sides of transfer as Internet Data Transfer. Eucalyptus, OpenNebula and OpenStack are open source software packages that can be used to create a private cloud Infrastructure service. Most of the services provided by AWS including machine image management, persistent storage etc. are provided by these packages. They often provide interfaces similar to vendor clouds like Amazon EC2 which allow the use of same set of tools and methods.

Platform as a Service

Platform as a Service (PaaS) helps in building and delivering applications by providing a computing platform. It includes facilities for different phases of a project lifecycle including design development, deployment and testing. Windows Azure is a good example of a popular PaaS service. Azure provides an environment for development, hosting, and management of applications. Applications hosted in Azure platform are provided compute resources and storage on-demand. Azure has two virtual machine instance types: the web role instance and the worker role instance. To store data and access it from these virtual machines Azure provides Blob storage Service. Even though Azure platform is designed for web applications, its use for HPC applications is being explored[20].

Software as a Service

Software as a Service (SaaS) provides end users with direct access to an application with a specific function. Applications and related data are centrally hosted in the cloud and accessed via a web browser. Examples in the public clouds include services like Google drive[21] and Microsoft Skydrive[22]. Documents, images, spread sheets and presentations can be created and imported using a web interface. Both of these services provide desktop application plugins to synchronise data created using the software suits like Microsoft Office. This gives users the convenience of accessing their application data from any workstation or mobile device.

2.2 HPC in cloud

Cloud computing started in context of web applications and enterprise systems that have completely different requirements when compared to high performance computing (HPC) applications like scientific computing. Traditionally these applications rely on HPC centres with dedicated systems connected through high
bandwidth interconnect and uses parallel file systems. Cloud provides the ability to manage or modify the software environment and get on-demand access to virtual resources which makes it attractive to HPC users. There are researches [4] done to test if the cloud resources can be used to replace the existing HPC systems or supplement them.

Figure 1: Google trends comparison of search terms (grid, cluster and cloud)

The scale of comparison in Figure 1 is based on the average worldwide traffic of grid computing in all years. This trend show how popular cloud computing is in comparison to grid computing and cluster computing. This popularity and large community support is another aspect that attracts small and medium HPC users to embrace cloud computing services.

2.3 Amazon Web Services and EC2

In 2006, Amazon.com started offering a collection of cloud computing services over the internet. This collection of web services is called Amazon Web Services[11]. Today, Amazon Web Services provides an IT infrastructure platform for business in the cloud with data centre locations in the U.S., Europe, Brazil, South America, Singapore, and Japan.

Figure 2: Amazon Web Services Logo
Amazon Elastic Compute Cloud (Amazon EC2) is an IaaS that provides resizable or elastic compute capacity in the cloud. It is designed to provide the freedom of choosing development environment and software stack to the developers. Amazon EC2’s web service interface allows users to launch and configure capacity with minimal effort. The time required to launch a new instance is less than a few minutes allowing users to quickly increase or decrease capacity, as their computing requirements change.

2.3.1 Amazon Instance types

- Standard instances having Memory-CPU ratio suitable for most applications
- Micro instance providing small and consistent CPU instance with burst in capacity when additional cycles are available.
- High Memory instance for supporting high throughput application which require good memory caching
- High CPU instances for compute intensive application
- Cluster Compute for high performance applications that require high CPU and increased network performance
- Cluster GPU instances having General Purpose Graphics Processing Units together with high CPU and networking to support HPC and media processing applications
- High I/O instances having Solid State Disks to support large amount of I/O operations

The different instance types provided by amazon each of these categories are shown in Table 1.

<table>
<thead>
<tr>
<th>Type</th>
<th>API Name</th>
<th>CPU Units</th>
<th>CPU (Cores)</th>
<th>Memory (GiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>m1.small</td>
<td>1</td>
<td>1</td>
<td>1.7</td>
</tr>
<tr>
<td>Medium</td>
<td>m1.medium</td>
<td>2</td>
<td>1</td>
<td>3.7</td>
</tr>
<tr>
<td>Large</td>
<td>m1.large</td>
<td>4</td>
<td>2</td>
<td>7.5</td>
</tr>
<tr>
<td>Extra Large</td>
<td>m1.xlarge</td>
<td>8</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>High-Memory Extra Large</td>
<td>m2.xlarge</td>
<td>6.5</td>
<td>2</td>
<td>17.1</td>
</tr>
<tr>
<td>High-Memory Double Extra Large</td>
<td>m2.2xlarge</td>
<td>13</td>
<td>4</td>
<td>34.2</td>
</tr>
<tr>
<td>High-Memory Quadruple Extra Large</td>
<td>m2.4xlarge</td>
<td>26</td>
<td>8</td>
<td>68.4</td>
</tr>
<tr>
<td>High-CPU Medium</td>
<td>c1.medium</td>
<td>5</td>
<td>2</td>
<td>1.7</td>
</tr>
<tr>
<td>High-CPU Extra Large</td>
<td>c1.xlarge</td>
<td>20</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>Cluster Compute</td>
<td>cc1.4xlarge</td>
<td>33.5</td>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>Cluster Compute</td>
<td>cc2.8xlarge</td>
<td>88</td>
<td>16</td>
<td>60.5</td>
</tr>
<tr>
<td>Cluster GPU</td>
<td>cg1.4xlarge</td>
<td>33.5</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>High I/O Quadruple Extra Large</td>
<td>hi1.4xlarge</td>
<td>35</td>
<td>8</td>
<td>60.5</td>
</tr>
</tbody>
</table>

Table 1: Amazon EC2 Instance types
In the table all instance types have their name starting with the class of instance they belong. The only exception is the Standard instance which includes the first four types in Table 1. API Name is the unique name used to identify the instance type. CPU cores and CPU memory are the virtual resources available for each type of instance. CPU Units/EC2 compute unit is specific to EC2 instances where one compute unit is equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor [23].

2.3.2 Amazon Machine Image

An Amazon Machine Image (AMI) is a pre-configured template operating system which is used to launch an EC2 instance. Users can create their custom AMI using the developer tools. A machine image is compressed and split into a set of 10MB chunks and stored in Amazon S3 storage. Even though each AMI has a pointer to a default kernel, user may choose a kernel from a list of compatible kernels maintained by Amazon and its partners like Red hat and Canonical. There are two types of AMIs in Amazon EC2 based on the type of virtualisation used by the Machine Image

Paravirtual Machine (PVM) Images

Paravirtualisation [24] is a virtualisation technique which provides a software interface to guest virtual machines. These interfaces are similar to that of the actual hardware but not identical. This modified interface reduces the virtual machine’s execution time spent in performing operations which are considerably more hard to run in a virtual environment compared to a non-virtualized environment. In this technique the interface/API provide by the hypervisor [25] is used by the guest operating system. So this requires modification of guest operating system.

Amazon instance type (Standard, High Memory, High CPU & High I/O) that use paravirtual machine based virtualization (PVM) only support Paravirtual (PV) AMIs. This doesn’t provide support to EC2 Cluster instances and GPU instances. These AMIs are modified to support paravirtualisation by hypervisor.

Hardware Virtual Machine (HVM) Images

Hardware-assisted Virtualization [26] is another virtualisation technique that enables efficient full virtualization assisted by hardware capabilities. In this method, the host processors provide explicit support for virtualisation. Complete virtualization is used to simulate a full hardware environment in which a guest operating system executes without any modification and unaware of the virtualisation. Hardware assisted virtualisation was supported by x86 processors (Intel VT-x or AMD-V) in 2006 and now, many modern processors support virtualisation. Intel and AMD have made modifications to Xen[27] to support these architectures.

In EC2 Cluster instances and GPU instances run as HVM instances. With HVM virtualization the guest virtual machine runs as if it were running on a native hardware platform. The network and storage drivers are still uses paravirtualisation for improved
performance[28]. These paravirtual device drivers bypass the emulation of disk and network I/O and thereby boost the performance of HVM guests.

2.3.3 EC2 Pricing

Amazon has a policy of pay only for what you use. They have three types of purchasing options:

- On-Demand Instances: Instances that let users pay for compute capacity in an hourly basis with no upfront payments or long-term commitments
- Spot Instances: Instances that allows the customers to set a maximum price they are willing to pay for running a particular instance type
- Reserved Instances: Instances that let you make an upfront payment for an instance, typically by reserving it for a one or three year term at a lower rate

On demand price list of different Linux instance usage for US East (N. Virginia) is shown below in Table 2.

<table>
<thead>
<tr>
<th>Region: US East (N.Virginia)</th>
<th>Linux/UNIX Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Standard On-Demand Instances</strong></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>$0.080 per Hour</td>
</tr>
<tr>
<td>Medium</td>
<td>$0.160 per Hour</td>
</tr>
<tr>
<td>Large</td>
<td>$0.320 per Hour</td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.640 per Hour</td>
</tr>
<tr>
<td><strong>Micro On-Demand Instances</strong></td>
<td></td>
</tr>
<tr>
<td>Micro</td>
<td>$0.020 per Hour</td>
</tr>
<tr>
<td><strong>High-Memory On-Demand Instances</strong></td>
<td></td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.450 per Hour</td>
</tr>
<tr>
<td>Double Extra Large</td>
<td>$0.900 per Hour</td>
</tr>
<tr>
<td>Quadruple Extra Large</td>
<td>$1.800 per Hour</td>
</tr>
<tr>
<td><strong>High-CPU On-Demand Instances</strong></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>$0.165 per Hour</td>
</tr>
<tr>
<td>Extra Large</td>
<td>$0.660 per Hour</td>
</tr>
<tr>
<td><strong>Cluster Compute Instances</strong></td>
<td></td>
</tr>
<tr>
<td>Quadruple Extra Large</td>
<td>$1.300 per Hour</td>
</tr>
<tr>
<td>Eight Extra Large</td>
<td>$2.400 per Hour</td>
</tr>
<tr>
<td><strong>Cluster GPU Instances</strong></td>
<td></td>
</tr>
<tr>
<td>Quadruple Extra Large</td>
<td>$2.100 per Hour</td>
</tr>
<tr>
<td><strong>High-I/O On-Demand Instances</strong></td>
<td></td>
</tr>
<tr>
<td>Quadruple Extra Large</td>
<td>$3.100 per Hour</td>
</tr>
</tbody>
</table>

Table 2: Pricing per instance-hour consumed for each EC2 instance
An EC2 instance is metered from the time it is launched until it is terminated and each partial instance will be calculated as a full hour.

2.4 Parallel Applications and 13 Motifs

In 2004 Phillip Colella[29] suggested a list of algorithms known as “Seven Dwarfs” for high-end applications in the physical sciences.

- Structured Grids
- Unstructured Grids
- Spectral Methods
- Dense Linear Algebra
- Sparse Linear Algebra
- N-Body Methods
- Monte Carlo

These dwarfs are defined at a high level of abstraction to explain their behaviour across different HPC applications and each class of dwarfs have similarity in computation and communication. According to his definition, applications of a particular class can be implemented differently with the change in numerical methods over time, but the underlying patterns have remained the same over generations of change and will remain the same in future implementations.

A group of researchers at University of California at Berkeley have studied and upgraded these classifications to 13 dwarfs[30], where a dwarf is defined as an algorithmic way to capture a pattern of computation and communication. According to their new studies the applications in HPC can be classified as the following 13 motifs[7]

- Dense Linear Algebra
- Sparse Linear Algebra
- Spectral Methods
- N-Body Methods
- Structured Grids
- Unstructured Grids
- MapReduce
- Combinational Logic
- Graph Reversal
- Dynamic Programming
- Backtrack and Branch-and-Bound
- Graphical Models
- Finite State Machines
The figure shows the importance of these motifs to different application areas. Since this work is related to HPC applications, the following subsections will discuss two widely used patterns according to this classification. They are Spectral Methods and N-Body methods.

### 2.4.1 Spectral Methods

In scientific computing, a class of techniques used to numerically solve differential equations involving the use of Fast Fourier Transform are called spectral methods. Spectral methods are used in many application areas in scientific computations like fluid dynamics, quantum mechanics and weather prediction.
**Fast Fourier transform**

Discrete Fourier transform (DFT) is widely used by many science and engineering calculations. Fast fourier transform is a discrete fourier transform that reduces the computational complexity of N point DFT from $2N^2$ to $2N \log_2 N$.

Jean Baptiste Joseph Fourier (1768-1830) first used Fourier transforms when he was working on the theory of heat. FFT is a linear transformation which takes temporal or spatial information as input and converts into data in the frequency domain.

**2.4.2 N-Body Methods**

N-Body Methods depends on interactions between many dynamic particles in a domain. These particles will be usually under the influence of forces from other particles Some of the different types of N-Body algorithms widely used include particle-particle methods, where calculation on each point depends on all other points in the domain, leading to an $O(N^2)$ calculation, and hierarchical particle methods, which combine interactions from multiple points to reduce the computational complexity to $O(N)$ or $O(N \log N)$. Scientific applications using this calculation pattern include molecular dynamics,

**Molecular Dynamics**

Molecular dynamics is a simulation of physical movements of molecular and sub molecular particles like atoms and molecules. The interaction between atoms or molecules over a period of time is recorded and this gives an idea of the motion of atoms. The common strategy is to calculate the trajectories of molecules and atoms by numerically calculating the forces between particles for a system of interacting particles using the Newton's equations of motion.
Figure 5: A snapshot from an Ab-initio molecular dynamics simulation [31]

The method is now commonly applied in materials science and modelling of bio molecular particles.

2.4.3 Ping Pong

In ping pong benchmark two processes are involved in the execution. From one process a message or ping is sent to the other process and this message is bounced back to the first process as reply or pong. MPI blocking send and receive is used to pass these messages between processors. This ping pong pattern is continued in a loop with fixed number of iterations. The total time is divided by number of iterations to find the time per message or latency.
Chapter 3

Experimental design

This chapter provides the analysis, design and implementation of Hawk-i tool. We will first discuss the decisions made for creating the live benchmark tool and what changes were made in the initial proposal during the progress of this project. System design and implementation sections explain the different stages of development and challenges faced.

3.1 Design Decisions and Assumptions

In this section we explain how the initial proposal turned into a project design and how the design decisions were made and the assumptions.

3.1.1 Instances used

AWS provides several EC2 Instances [Table 1]. Features of these instances were carefully studied to select the instances to be included in the benchmark tool.

Amazon EC2 instances are grouped into seven families as described in background. We have included all the instances except Micro instance and GPU instance.

GPU instance has been excluded for this work since creation of GPU benchmarks were out of scope of this project. Adding GPU instances in future will not be difficult, if there is a good GPU application benchmark available.

Micro instance was initially included during the initial test prototypes. The performance of t1.micro was analysed continuously using a sample dashboard created during project preparation. The results recorded over a long period of time showed anomalies in the performance of EC2. To verify this, execution time of t1.micro was compared with m1.small instance using a compute intensive kernel of molecular dynamics code [F.5].
The results showed that even though t1.micro produced the result in least time for first execution, consecutive executions have shown a really poor performance whereas m1.small gave consistent performance.

On further investigation of Micro instance behaviour, we found [32] that this is an induced limitation in micro instance since it is configured to support low throughput applications like websites. These applications typically have a periodic burst in compute cycle requirement. Considering this issue with micro instance to provide consistent performance forced us to exclude it from the list of supported instances.

Even though Micro instance was excluded from instance list, it was identified as a suitable instance to host the manager application and webserver for Hawk-i tool. The compute cycle burst was found to be useful to respond to multiple instances or requests. It is also available in a very low running cost of $0.020 per hour in US East (Virginia) region.

Initial plan was to keep all the selected instances to run continuously. Considering the cost of execution, this was revised and booting up instances on-demand was devised.

### 3.1.2 Starcluster

With the experience from Cluster building Challenge [33], creation of test clusters in Amazon EC2 instances during initial tests were fairly easy. The virtualised environment reduced the effort of configuring hardware and software. The real challenge was when we tried to automate cluster creation using EC2 command line tools. Through discussions with developers in AWS community, we came to know about Starcluster project [34] in Massachusetts Institute of Technology which is released under LGPL License [35].
Starcluster is an open source cluster computing toolkit used to create and manage SGE cluster in AWS EC2. We decided to use Starcluster command line tools to create manage and terminate clusters during automated benchmark execution. Our previous experience in building bare metal heterogeneous cluster with Sun Grid Engine and familiarity of using Ness[36] was useful in using Starcluster.

3.1.3 AMI used

As described in background AMI contains details of a running operating system that could be used to clone similar instances. All nodes in the cluster will run the same AMI and all nodes will have the same configuration.

We were in need of software like Sun Grid Engine to manage automatic cluster configuration since we were testing cluster performance of EC2 cloud. In addition to that we were also planning to install OpenMPI library to support message passing programming in our cluster.

For this project we decided to use two types of AMIs [Appendix B], one for x86_64 and one for HVM since the cluster compute instance required HVM Image. These AMIs were provided by Starcluster community. They included minimal software stack for distributed and parallel computing. Starcluster AMIs contains the following software and configuration:

- Open Grid Scheduler/Grid Engine[37]: An open-source batch queuing system for managing distributed resources.
- Automatic configuration of NFS[38], grid engine and password-less SSH
- OpenMPI 1.4.3[39]: Library for writing message passing programs
- Other packages like ATLAS [40], NumPy [41] and IPython [42] which are not used in this project.

It was useful and convenient to use these AMIs designed for scientific computing. Besides that there was an active support from Starcluster community mailing list [43].

3.1.4 Benchmarks

We started with the idea of finding a benchmark that has computational pattern similar to an existing scientific application. By running this benchmark in cloud the users should be able to estimate how their application would run in the same infrastructure. Performance benchmarks like LMBench[44], Bonnie[45], and Cache Bench[46] were investigated to study different benchmark implementation. We also researched existing application benchmarks that can be standardised and used to measure application performance in cloud. Scientific computational suites like OpenFOAM[47] and AMBER[48] were studied to create benchmarks that can mimic these HPC applications. From these studies we learned the complexity of normal scientific computing applications and how they can’t be used for benchmarking. We had to find or create something simpler for testing EC2 performance using Hawk-i.
To test the functionality of Hawk-i benchmark tool, we were planning to create micro benchmarks mimicking the execution behaviour of existing real life HPC applications. During the research to do this work, we found a study by University of Berkley [7] in 2008 to classify of benchmarks based on the computation pattern. This approach was taken for selecting benchmarks to be added in Hawk-i. It was not possible to add all 13 benchmark types during this time estimated for the project hence we had to select two motifs. Spectral methods (FFT) and N-body (MD) were selected since they were widely used in scientific computing. Besides these two benchmarks, in order to test the EC2 performance for a high network utilising problem we decided to add ping pong communication benchmark.

From the beginning of the project, it was decided to measure only the execution time and memory scalability. As discussed before, there are several benchmarks like LMBench and Bonnie to test performance analysis of hardware or virtual hardware. The purpose of these benchmarks will be to gather results useful to a non-technical user which are how big of a problem can be executed and how long will it take to get the results.

The next question was to include the start-up time with different instances. The start-up time of all instances using x86_64 AMI was compared to see any significant difference. In this comparison, we used a two node cluster to include the SGE configuration time as well.

\[\text{Figure 7: Startup time comparison of two-node EC2 SGE Cluster}\]

The results [D.2] show that the mean time value for two-node SGE cluster start-up is 119.33 seconds with a standard deviation of 12.5 seconds. Since the start-up time is found to be approximately two minutes it was not significant for the cost of execution. In EC2, since the cost is measured in a one hour window and two minutes in that large window doesn’t cause any extra charge and start-up is a onetime process in cluster creation. So, we decided to exclude start-up time from our benchmark results.
3.1.5 System interface

Several existing dashboards like Cloud Climate [49] and Cloud Harmony [50] analyse and report cloud performance benchmarks for deploying web applications and services. We compared these dashboards and it was identified that they can be taken as a reference for building a benchmark tool for scientific applications. This was considering the usability of these tools for web developers. The same flow of control was considered useful for any application types tested in cloud. Even though it was useful in displaying the archived benchmark results for a variety of cloud infrastructure, none of these tools provided scientific application benchmarks or on-demand benchmarking option. Such a tool can be useful for scientists to test how a particular problem and problem size performs in a cloud instance.

In the initial design there were two interfaces to the Hawk-i system, a command line interface to add and schedule benchmarks and a web interface to view the results. Scheduled execution cost of the system was too high. Hence we thought of an alternative on-demand benchmark using the web interface. We still retain the scheduled benchmark on a small instance to verify the stability of results provided by AWS.

Due to convenience of scripting and wide library availability, we decided to use Python for creating the command line interface. Python being pre-installed in almost all major Linux distributions made this decision easier.

We chose PHP [51], MySQL [52] and Apache [53] to build the web interface since these applications are free to use under GPL License and they have a large user community for support. A prototype of web installation in manager instance was done during the preparation phase to successfully verify the web application performance in a micro instance. Due to our previous experience, Joomla [54] Framework was our first choice for user management and navigation. These open source tools and frameworks were chosen to reduce the development effort and project execution cost.

3.1.6 Execution cost

According to the EC2 documentation, the cost calculation in EC2 is from the time an instance is launched until it is terminated and the calculation window is one hour. Partial instances, execution less than on hour will be billed as a full hour.

Using this calculation all benchmarks running less than one hour in the same instance can provide the same cost and can create confusion. The initial decision to calculate cost had to be changed since it can give the wrong idea to the users. We assume that the users will be interested in knowing how much time it took, since the actual problem size will be much larger than the benchmark data.
3.2 System Design

In this Section, we discuss the system design of Hawk-i system. We will discuss the Architecture where the subsystems and their interactions are explained, Sequence diagram where the different entities communication is shown and Database design to explain the main data entities and their relationship.

3.2.1 Architecture

Entire Hawk-i system was divided into three different subsystems between which the control switches. They are:

- **Compute instance** which is an SGE cluster of amazon instances where the actual benchmark code is compiled and executed
- **Manager instance** that initiates benchmarking and coordinates other Hawk-I subsystems
- **Web user interface** that provides Hawk-i web access for admins managers and normal users

All these subsystems interact with each other as shown in Figure 8.

![Figure 8: High Level Interaction Diagram](image)

In this design, each **Compute instance** is a cluster of EC2 instances clustered using SGE. The code compiles in master node for the benchmark to support the hardware architecture of nodes in cluster. Once the results are obtained and returned to the manager, these instances can be killed. Each instance is created to execute a single benchmark with a unique instance name. This allows multiple cluster instances to be created and executed in Hawk-i system. The cluster instances are created by manager instance and they have sshd[55] running to receive and respond to request from manager instance.
Manager instance is an EC2 micro instance hosting the entire Hawk-i system and managing the data flow between different subsystems. The web interface doesn’t have direct access to compute instance and all communications during benchmarking are happening through manager instance. Manager Instance Interaction Diagram below shows the internal application architecture of manager instance.

![Manager Instance Interaction Diagram](image)

**Figure 9: Manager Instance Interaction Diagram**

Main features of manager instance are:

- It has a web server configured and Hawk-i web tool hosted to communicate with admin, manager and other system users.
- It contains a manager script that receives benchmark instructions from web interface or command line tool and initiates the compute cluster creation.
- Manager instance store the benchmarks which will be executed in compute instances. These benchmarks will be transferred to compute instances during benchmarking.
- A database management system configured in this manager store details of all benchmarks, instances, users and input types.
- Starcluster configured in manager is used to create communicate and terminate clusters for benchmarking.
- A scheduler executes selected benchmarks periodically to measure the stability of results.

Finally, the *Web user Interface* provides the tools required for users to select Amazon EC2 infrastructure, run benchmark program, and view results. It also allows the users to submit new benchmarks and add new inputs for existing benchmark. It provides statistics of previous benchmark executions that can be useful for users to study the results and select an instances that suits their computational requirement.

### 3.2.2 Control Sequence

The data and control flow between the subsystems is discussed in this section. This design is a result of several iterative refinements during different test implementations. The breakdown of this section is based on the key operations in Hawk-i system.

#### Add benchmark and inputs

This functionality allows the admin user to submit new benchmarks and inputs through web interface. The users downloads a template for benchmark modify it by adding the compute kernel and compilation options and submit it in the web interface provided to implement this functionality.

The sequence of events starts with the user inputting new benchmark details like source code and different inputs through a web form. This input is validated and stored in database by a server side program. On successful addition, a status message is displayed to the user.

#### Scheduled execution of benchmarks

Hawk-i interface show the results of a continuous execution of a benchmark to test the stability of results in EC2 Instances. Since the hardware is virtualised and shared by different cloud users, it is important to see that the cloud provides consistent performance. Hawk-i provides these results to all users in home page to show the reliability of EC2 Instances.

The sequence of steps required for scheduled execution is similar to execution of normal benchmark, the only difference being the execution initiated by scheduler instead of input from a user. In this case the benchmark doesn’t return any value to terminal or web interface, instead the results are stored in database. These results are retrieved and published in the Hawk-i web tool.

#### Execute benchmark

This is one of the main features of Hawk-i using which the users select and run the benchmark in a particular infrastructure with specific instance type, cluster size and input size. The primary interface for this execution is the web interface. The
administrator can bypass to direct script execution using command line execution for testing and other administrative purpose.

![Benchmark execution sequence diagram](image)

**Figure 10: Benchmark execution sequence diagram**

As shown in Benchmark execution sequence diagram, a normal execution starts at the web interface where the users submit the benchmark parameters and cluster parameters. From Web Interface, the parameters submitted by user are passed on to the manager program and the web instance waits for results. In the mean-time, the manager script asks EC2 service to start a cluster with these parameters. Once the request is submitted to EC2 service, manager wait for cluster to boot and establish network connectivity. EC2 service creates the cluster and sends back the cluster details like cluster name and domain name to manager program. On receiving confirmation from EC2 service, the manager copies the benchmark program selected by the user to the new SGE Cluster created. On successful completion of transferring benchmarks, the manager initiates the execution of benchmark with the parameters selected by the user and waits for execution to complete.

The master node in SGE cluster compiles the benchmark source. On successful compilation, the source is submitted for execution in the cluster. After execution, the manager instance is notified and the manager script copied the results from SGE master node to manager instance. These results are then analysed formatted and stored in a
database for reporting in web interface. The last step is to notify the web interface about completion of benchmarking by sending the status of execution together with results. These results are displayed to the user for verification.

**Display reports**

This feature prints history of executions made by the current user. This can be very useful to compare the performance of different instances in EC2. Using the results stored over a long period of time, users can also verify how stable the results are in EC2 instances.

The sequence of events is very short with the user requesting the details through a web interface and the server collecting the results from database for that particular user and publishing it for the user to analyse.

### 3.2.3 Database Design

Hawk-i system stores all instance and benchmark options in a database for easy access and management. Besides that, all results from executing benchmarks are also stored in a database and the users have the option to view their benchmark results at any point of time.

![Entity relationship diagram](image)

**Figure 11: Entity relationship diagram**

The data entities identified for this system are Benchmarks, Inputs, Instances, and Results. In Entity relationship diagram, we can see the different database entities interacting in the system.
In this design, benchmarks, inputs and instances are lookup entities for the data logged as results. The relationship between results entity and other entities is one to many. Inputs and benchmarks are also related by benchmarked in a one to much entity relation.
3.3 Implementation

In this section we discuss the implementation strategies used to develop and test Hawk-i tool. We have divided this discussion into two parts. Implementation of Hawk-I tool and development of sample benchmarks to test the tool. Before going into details of these two sub sections we will discuss the environment setup for the entire project.

3.3.1 Environment setup

Even though the development environment was setup in the beginning phase, few modifications were made during the development. All modules were developed on top of this. This task included creating amazon instances, installing web server and database, configuring EC2 command line tools, configuring Starcluster and creating code repository.

At first, we created an Amazon Web Services account and using this account we launched [Appendix C] an amazon EC2 micro instance. We name it as manager instance and the default security groups were modified to allow SSH and HTTP access to manager instance. During instance creation we created a key pair that is important to gain access to the new instance created. So this file was replicated and stored in different machines to have a secure backup. We have also taken periodic backup of this manager instance using snapshot manager [56] in AWS EC2.

The second task was to install [Appendix C] webserver and database server in this manager instance. We used apt-get [57] to install the LAMP[58] environment that provided an Apache Web Server (Apache/2.2.20), MySQL server (Version 14.14), and PHP (PHP 5.3.6) in a Linux environment. This setup was tested using a ‘hello world’ web programs that used database connections. For ease of access to database, we have also configured a phpMyAdmin-3.4.5[59] tool to modify the data and structure of database elements.

With this configuration, we had a system that was capable enough to prototype the working of Hawk-i. A sample molecular dynamics code which is an older version of the final molecular dynamics benchmark code was tested in this environment. As discussed in design decisions, tests revealed the incapability of micro instance to provide consistent performance and its usefulness for applications with periodic resource requirement bursts. Giving an interval of 30 minutes, the molecular dynamics code was scheduled for execution and the execution time was stored in database. This data was plotted in a web interface using PHP scripting and Google chart API. With this working prototype we were confident to go ahead with the configuration for invoking remote instances, clustering them and running benchmarks.

The next step was to setup EC2 command line tools (CLT) [Appendix E] which were used to launch new instances in AWS EC2. As the name suggests, CLT allow users to do all tasks done through EC2 management console and since they were executed through shell, these tasks could easily be automated and hence we used it in Hawk-i tool. To develop Hawk-i, we also configured boto [60](Version 2.2.2) python package
that provided python libraries to use these command line tools. These command line tools and packages were configured in the manager instance as well as in local machine. The second prototype worked with this configuration where new instances were spawned using these tools from a python manager program to execute sample programs in new instances and store results in manager database.

Finally we configured [Appendix C] a cluster management tool Starcluster which was used to create instances and cluster them using Open Grid Engine (OGS/GE 2011.11). Starcluster was installed using easy_install [61] (Version 0.6.16) from PYthon Package Index (PYPI).

3.3.2 Hawk-i Implementation

This section discusses the implementation of Hawk-i tool which contains development of Manager Instance that spawn instances, benchmark them and store data in database, and Web interface that provide we access to manager functionalities and publish the results stored from different benchmark executions.

The directory tree layout shown in Figure 12 can be useful in understanding the system details explained in the following subsections.

![Figure 12: Directory layout of Hawk-i System](image)

Manager Instance

As shown in Figure 9, this is the heart of this tool and it controls flow of data from different entities in the entire system. The manager instance functionality is implemented using a python manager script, manager.py [F.1]. It interacts with web interface, EC2 and database. The administrator can test benchmark programs and debug Hawk-i using this interface as shown in Figure 13.
This manager script reads the following options from users through command line.

- **Clustername**: User can specify a name for the cluster created through manager. If no name is specified, then the current time stamp is used as the cluster name.
- **ImageId**: The AMI used to create each node that is part of the cluster.
- **InstanceType**: The type of machine used in creating the cluster, selected from different cluster types EC2 provides.
- **Size**: Number of instances to be launched. This will be the cluster size, i.e., number of nodes in the cluster.
- **Benchmark**: Path to benchmark code and scripts (folder). The entire folder will be uploaded to the master node for execution.
- **UserId**: Web user id from web interface. Each execution will be tagged to a user id and the results will be stored in the user’s profile.
- **Results**: Path to benchmark results folder. Even though default results folders will be created during execution, this option allows users to create a results folder for debugging.
- **Archive**: Archive results to database (0/1). By setting/un-setting this option users can choose to enable or disable results being stored in database.
- **Inputdata**: Input parameters option is used to change input parameters. These parameters will be set during program execution allowing users to easily run the program with multiple parameters without modifying the code.
- **Threshold**: This option gives the freedom to stop execution in a particular time frame. If the execution is not completed within the specified threshold the instance is terminated immediately.

Python manager will validate these options and start the cluster by creating a sub process to contact EC2. The program waits for the command to return a success message. The next step is to send the benchmark folder selected over SCP to the master node, again by creating a sub process in python. Once the files have been transferred, manager executes benchmarks over SSH and waits for execution to complete. The results stored by SGE are retrieved again over SCP and stored in a temporary folder. Execution time is extracted from the results and it is stored in MySQL database if the archive option is set by user. The last step is to terminate the cluster by issuing the termination signal to EC2. Results and status messages are printed during all phases of execution and all the operations to communicate with new instances are carried out through the Starcluster API calls.
This manager program can easily be linked to a web interface to accept execution calls from it and to return results back.

Web interface

This is the Hawk-i interface that is used by normal users, managers and administrator to submit benchmarks, execute them and view results and other statistics. This is built in PHP using Joomla-1.5 framework. We created 3 different user types in the system called User, Manager and Admin. Users are normal web users who will be able to view the results published in dashboard for different benchmark executions that are published for public access. This user doesn’t login to the system or interact with the system in any form. The second level of user is the manager who runs existing benchmarks by selecting different input types configured in the system. The primary user is the administrator and he has permission to add new benchmarks, add new input types and execute benchmarks. This system also has a backend to add new users to the existing system.

During implementation of Hawk-i functionalities, we have created three Joomla modules named mod_livebenchmark, mod_reports, mod_benchmarkmanager.

In live benchmark module (mod_livebenchmark) [Figure 14], we have created a form for users to submit the benchmark execution details. The users select these options and submit the form. Data submitted is processed by this module and it extracts the benchmark parameters. These parameters are set as command line options and the manager script is called from the PHP script. Remaining execution is managed by the manager script and the results are returned to this module. This module displays the results like execution time and execution status to the user.

![Figure 14: Live benchmark input submission form](image)

Reports module (mod_reports) collects the previous execution results like execution time, status together with timestamp and benchmark details, and displays them in a sortable table as shown in Figure 15. This can be useful to compare the results of different benchmark executions using Hawk-i tool.
Benchmark Manager Module (mod_benchmarkmanager) provides the admin interface to add new benchmarks and inputs. So, the first part of this module allows users to add new benchmarks. The user is provided with a downloadable template for benchmark code which is modified and submitted by the admin user using a form as shown in Figure 16.

![Figure 15: Results log of benchmarks execution](image)

<table>
<thead>
<tr>
<th>Date</th>
<th>Benchmark</th>
<th>Instance type</th>
<th>Input</th>
<th>Count</th>
<th>Execution Time</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-08-14 10:25:26</td>
<td>m1.small</td>
<td>m1.small</td>
<td>4194304</td>
<td>1</td>
<td>24.3</td>
<td>Success</td>
</tr>
<tr>
<td>2012-08-14 10:17:36</td>
<td>m1.small</td>
<td>m1.small</td>
<td>2097152</td>
<td>1</td>
<td>12.03</td>
<td>Success</td>
</tr>
<tr>
<td>2012-08-14 10:07:39</td>
<td>m1.small</td>
<td>m1.small</td>
<td>524288</td>
<td>1</td>
<td>3.17</td>
<td>Success</td>
</tr>
<tr>
<td>2012-08-14 10:17:33</td>
<td>m1.small</td>
<td>m1.small</td>
<td>1048576</td>
<td>1</td>
<td>8.79</td>
<td>Success</td>
</tr>
<tr>
<td>2012-08-14 10:27:27</td>
<td>m1.small</td>
<td>m1.small</td>
<td>8388508</td>
<td>1</td>
<td>48.75</td>
<td>Success</td>
</tr>
<tr>
<td>2012-08-14 10:28:17</td>
<td>m1.small</td>
<td>m1.small</td>
<td>16777216</td>
<td>1</td>
<td>123.25</td>
<td>Success</td>
</tr>
<tr>
<td>2012-08-14 10:31:39</td>
<td>m1.small</td>
<td>m1.small</td>
<td>33554432</td>
<td>1</td>
<td>278.64</td>
<td>Success</td>
</tr>
</tbody>
</table>

![Figure 16: Form to add new benchmarks](image)

The tar ball submitted is extracted and saved in manager instance as a new benchmark and the different inputs are stored in database together with path to new benchmark folder added in Hawk-i. This benchmarks and inputs are later used by the manager script for execution in cluster.

Second option provided by this module is to add new inputs for existing benchmarks where the user selects a benchmark and add new inputs as shown in Figure 17.
Even though this module helps in adding new content, we haven’t implemented an interface to delete or update existing content like inputs and benchmarks. This has to be done through the phpMyAdmin interface by the administrator.

The navigation to these modules is provided through a top menu which is visible in all module pages.

### 3.3.3 Benchmark Implementation

FFT and MD are the sample benchmarks created to test Hawk-i tool. Besides these two benchmarks a parallel ping pong was used to test the message passing capacity of the cluster built in cloud infrastructure. The benchmarks were created using C programming and it was created in a template designed for execution in Hawk-i system.

This template is nothing but a set of supporting files that assist the remote execution from manager instance. The template file contains the following files:

- Sun Grid Engine Script file (SGE file)
- A Perl script to submit SGE file in queue and wait for results
- A demo C program which will be replaced by the actual C program
- Readme file containing instructions to modify the template

The SGE file contains scripts [F.4] that are normally used to submit the jobs in any SGE cluster whereas the Perl script is a file that is required to support the manager program.

This script can be referred as an agent that executes on initiation from manager script in manager instance. It receives number of processors in which the job is to be submitted in a parallel environment, command line arguments to be forwarded to the benchmark file and threshold value which is the number of seconds the Perl script has to wait for the completion of job submitted in job queue.

All the three benchmarks were created using this template. As mentioned before the benchmark manager provides this template download option bundles with the tool. Besides these two files the main program must print a success message on successful completion as mentioned in the template program file.
FFT benchmark was created using an opensource implementation of FFT function in C, ‘fft.c’ [F.2]. In this experiment, this benchmark is used to measure the memory limit of each instance in EC2. The benchmark program, ‘fft_benchmark.c’[F.3] generates a sample input data depending on the input parameters passed to the program. This data is passed on to the function and the entire computation is timed using wall clock time functions to obtain the exact computation time. Makefile was modified to specify the source files that have to be compiled and object files that have to be linked to obtain the executable benchmark file.

The molecular dynamics benchmark code [F.5] implements long range inverse square forces between particles. In this experiment, this benchmark is used to measure the computation performance of the cluster. Input is generated by the benchmark program based on the problem size specified in command line. Execution starts with initialising values to mass, viscosity and position variables for each particle. The program calculates the force acting on each particle due to interaction with other particles and updates the particle position based on only for one time step for very large number of particles. This calculation is timed and published as result.

Ping pong benchmark [F.6], ‘pingpong.c’ creates an array of size specified in command line and sends it back and forth between two processors. This is repeater for the number of iterations specified in command line. The program is derived from a simple MPI ping pong exercise used during the message passing programming course. We are using synchronous send and asynchronous receive to send the array between processors. The array data type is double which is 8 bytes in size. The program reports the computing time which is the wall time taken for all send receive calls, total size of messages passed during the entire communication, time per message also known as latency, and bandwidth of data transfer between processors.

**Running Benchmarks**

There are two ways of running these benchmarks. First one is through the Hawk-i web interface and the second method is using the admin command line tool.

Only admins and managers have the privilege to execute benchmarks on demand using the web interface. The user selects the benchmarks option from the top menu which takes them to benchmark page where they select the benchmark from a list of benchmarks added by the administrator. Users have to select the instance type and number of instances to submit the benchmark. On selecting all three options, they will be provided an option to choose the input parameter from a limited set of inputs which is also set by the administrator. Final step is to set the time limit for execution and submit the benchmark details.

Detailed results will appear few minutes after submission depending on the problem size and benchmark complexity. The summary of results will be available in results tab of Hawk-i web interface and this can be accessed by the user at any time.
In this experiment all the benchmarks created were tested in Hawk-i system to identify how EC2 performs in solving HPC applications. Most of the tests are performed through web interface with the exception of additional tests to verify the reason for unexpected results. These additional tests were performed using admin command line tool.

Optimising and tuning – We compiled the benchmarks using GNU C Compiler version 4.6.1 in paravirtualised AMIs as well as HVM AMIs. We did not use any additional optimisation which is architecture or instance dependent.

We have also run some addition tests to find reasons for the anomalies in results during detailed analysis of results. These tests and their implementation are discussed a part of detailed analysis [4.2].
Chapter 4

Results and Analysis

4.1 Benchmark results

Once the Hawk-i tool was implemented we benchmarked different EC2 instances using the sample benchmarks created. All the executions were done using the web interface provided by Hawk-i and the results were archived in database. Separate users were made to benchmark each application for easy analysis of data provided in user reports. We have measured the memory limit for each EC2 instance and how the execution time varied for each type of instance. We have also measured the parallel performance using a simple ping pong program. From the large set of data generated using Hawk-i tool, we have only listed the interesting data that can be useful to explain the behaviour of EC2 instances.

4.1.1 Spectral Methods (FFT Benchmark)

We measured the execution time in different types of instances for FFT code and we were expecting to get a performance improvement for instances with more ECU per core. This test is divided based on different instance type classes like standard, high memory, high CPU and cluster compute. Through this test we are testing the memory limit of each instance type. Before doing this experiment, we gathered the results for small input sizes to learn how the execution time for this algorithm carries with input size. Since we had inconsistent results in some instances we have chosen the median of three executions to find the execution time scaling.
Since the problem size for FFT increases in powers of two, we expect to get an exponential increase in execution time with increase in problem size [Figure 18].

In memory limit test, the small instance (m1.small which has 600 MB memory) was expected to be the first instance type to fail. The above figure show a clear distinction of execution time only for a problem size above $2^{25}$, so we decided to start with the problem size as $2^{25}$ which was executed in all standard instances and then increase it gradually to find the memory limit. Whenever an instance failed due to memory limit, the benchmark was re-executed two time using admin console to confirm the results.

Figure 19: FFT execution time and memory limit in Standard Instances
In Figure 19, we can see that the small instance was able to execute only up to $2^{25}$ problem size. Extra Large Instance (m1.xlarge) was able to handle bigger problem sizes with a limit of $2^{28}$ time domain inputs. Small instance showed the worst performance since it has only one EC2 compute units per core. All other instances have two compute units per core and hence double performance. The performance difference between instances having same compute units per virtual core is explained later in detailed analysis section.

**High Memory Instances**

We expected high memory instances to have the largest memory limit when compared to other instance types. Due to this reason we have started our benchmark at the point where m1.small failed. Since all the instances

![High Memory Instances Graph](image)

**Figure 20: FFT execution time and memory limit in high memory instances**

The results shown in Figure 20 were as expected with 3.25 EC2 compute units per core, different instance types within high memory instance class gave approximately similar performance for all instance types. Only difference between these instances was the memory limit which was $2^{28}$, $2^{29}$ and $2^{30}$ for m2.xlarge, m2.2xlarge and m2.4xlarge respectively.

**High CPU Instances**

We did not expect high CPU instances to have large memory limit. The EC2 specification for high CPU instance gives us only 2.5 compute units per core which is less than all high memory instances.
The results [Figure 21] show that the memory limit of c1.medium is $2^{25}$, c1.xlarge is $2^{27}$. This data proves that EC2 CPU instances are slower than high memory instances, but since they are cheaper, they can be very useful to solve compute bound applications. The interesting result is the execution time at problem size $2^{25}$ where compute units per core is same (2.5) for both High-CPU Medium and High-CPU Extra-large instances, but the performance is better in Extra-Large Instance. This is carefully studied and explained in the detailed analysis section.

**Cluster Compute Instances**

Since Cluster Compute instances are provided specifically for HPC applications, amazon guarantee[62] this class of instances to scale better compared to other EC2 instance types. Large memory (23 GB and 60.5 GB) provided by EC2 cluster compute instances supports this statement. In this experiment we do not check the parallel performance; we only check the performance of execution in a single core. Since the virtualisation technology in both instances is HVM and as mentioned in Amazon Machine Image section, they will provide proportionally higher performance. *Quadruple Extra Large Instance* gives the performance of 2 x Intel Xeon X5570 accounting to a total of 33.5 EC2 compute units whereas *Eight Extra Large Instance* provides performance of 2 x Intel Xeon E5-2670 which is equivalent to 88 EC2 compute units.
The results show the consistent exponential change in computation time for FFT benchmark. Cluster compute eight extra-large instance was able to handle a problem size of $2^{30}$ whereas the Quadruple Extra Large Instance failed to execute a problem size of $2^{29}$.

From these results we found that High-Memory Quadruple Extra Large Instance and the Cluster Compute Eight Extra Large Instance are the instances with high memory and computation power to handle large scientific applications with FFT kernel.

**4.1.2 N-Body Methods (MD Benchmark)**

In order to test the N-Body methods, we used MD benchmark program in Hawk-i. The test was done in different EC2 instance types and the results are recorded. We are looking to measure the performance variation in different instance types for N-Body problem. We have chosen a $2^{16}$ as the fixed memory size which was tested to be sufficient enough to distinguish the performance difference. This is also the problem size that executed in all EC2 instance types.

In this test we were expecting the instance types with larger EC2 compute units per core to perform better than other instance types. In other words, we are expecting the best performance for High-Memory Instances and Cluster compute instances. We ran the experiment in each instance three times and the results displayed are the mean values of all three executions [D.8].

**Figure 22: FFT execution time and memory limit for cluster compute instances**
The results [Figure 23] show performance improvement in high memory instances and Cluster compute instances. High memory instances (m2.xlarge, m2.2xlarge and m2.4xlarge) have 3.5 EC2 compute units which is the highest among all Paravirtualised instance types and Cluster compute instances have HVM parallelised Intel Xeon compute units. It is evident from the graph that the performance change is consistent and directly related to the compute units per core.

The poor performance of small instance is studied in detailed analysis after combining the results from other benchmarks like FFT and ping pong.

4.1.3 Network Test (Ping Pong Benchmark)

Ping pong program was used to test the network latency of clusters built using EC2 instances. In this experiment, we intend to test Amazon’s promise of better network connectivity in EC2 cluster instances. We collected the latency output from Hawk-i tool execution output displayed in browser and compiled in a spread sheet for comparison. Below is the graph [Figure 24] for results obtained after executing ping pong benchmark program. The input array length is $2^{18}$ and iterations is 1000 which converts to 2 MB of message data sent twice between the processors, for thousand times.
The results show expected change in time taken per message (latency). Cluster compute instance have very low latency due to high bandwidth networking provided by Amazon. The non-linear change in performance between instance types with change in compute units per core and the anomaly of instances with same compute units per core giving different performance is discussed in the detailed analysis.
4.2 Detailed Analysis of results

In this section we will discuss the experiment results in detail. We will also discuss some additional tests performed to find the reasons for these unexpected results and how these findings can be useful for users and researchers. This section compares results from all the three benchmarks for each attribute we are measuring using Hawk-i.

4.2.1 Execution Time

**Observation 1:** Multiple instances launched using same EC2 instance type showed different performance.

During analysis of different benchmark results archived in Hawk-i, we were able to identify huge variation in performance for different launches of same instance types. We conducted a detailed study to find the reason for this.

We created a CPU test script executed `lscpu` command and added it in the template format for Hawk-i and named it as ‘cputest’. We used submitted this program in Hawk-i command line with cluster parameters, size=5 and type=m1.small. This created a five node cluster and executed `lscpu` command in each node and returned the result to manager instance.

![Hawk-i shell execution for CPU test](image)

The results obtained displayed the CPU details of each instance spawned in EC2. The comparison of these features was done to reach a conclusion on the anomalies observed before.
<table>
<thead>
<tr>
<th>CPU INDEX</th>
<th>ClockSpeed (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 1</td>
<td>2266.746</td>
</tr>
<tr>
<td>CPU 2</td>
<td>2659.998</td>
</tr>
<tr>
<td>CPU 3</td>
<td>2659.998</td>
</tr>
<tr>
<td>CPU 4</td>
<td>2660</td>
</tr>
<tr>
<td>CPU 5</td>
<td>1999.975</td>
</tr>
</tbody>
</table>

Table 3: CPU Clock speed comparison in m1.small instances

The cache size was also different for each instance tested [Table 4]. This result proved that each Paravirtualised instance have different clock speed and cache size. This is the reason of slower performance in few instances with same instance type. It was also observed that the resources remain same once the virtual machine is launched. The difference in clock speed for same instance types

This can be a motivation to modify Hawk-i to spawn instances, run pilot jobs [63] and discard the slow instances. Then we can add new nodes repeating the test to get a cluster configuration that is cost effective.

This difference was not observed in Cluster compute instances since the virtualisation is HVM. The paravirtualised instances get the clock speed and cache size of the actual CPU executing the instructions.

Reasoning: The difference in performance of paravirtualised machines can be the due to the difference in cache size and clock speed of the different virtual CPU model allocated for each launch.

**Observation 2**: Significant performance difference between standard instances having same compute unit.

We got the cache size of each instance type using `lscpu` command and the results are show below in Table 4. We verified the results and all instances get the same CPU model except m1.small and m1.large. The results shown here for these two instances may change when it is verified for a different launch.
Standard instances have different cache size

<table>
<thead>
<tr>
<th>Instance</th>
<th>CPU MHz</th>
<th>Virtualization type</th>
<th>L1d cache</th>
<th>L1i cache</th>
<th>L2 cache</th>
<th>L3 cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>2666.774</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>6144K</td>
<td>--</td>
</tr>
<tr>
<td>m1.medium</td>
<td>2000.07</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>12288K</td>
</tr>
<tr>
<td>m1.large</td>
<td>2659.998</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>12288K</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>2000.072</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>8192K</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>2666.76</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>8192K</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>2666.76</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>8192K</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>2666.76</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>8192K</td>
</tr>
<tr>
<td>c1.medium</td>
<td>2327.502</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>6144K</td>
<td>--</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>2327.508</td>
<td>Para</td>
<td>32K</td>
<td>32K</td>
<td>6144K</td>
<td>--</td>
</tr>
<tr>
<td>cc1.4xlarge</td>
<td>2933.039</td>
<td>Full</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>8192K</td>
</tr>
<tr>
<td>cc2.8xlarge</td>
<td>2594.14</td>
<td>Full</td>
<td>32K</td>
<td>32K</td>
<td>256K</td>
<td>20480K</td>
</tr>
</tbody>
</table>

Table 4: CPU Cache Size comparison of EC2 Instances

From these results we were able to identify the difference in clock speed and cache size in Standard instance types. All standard instances have fixed CPU clock speed except small instance and large instance. The paravirtualised CPU model was changing for m1.large and m1.small instance during each launch. This variation can explain the difference in performance between cores having same compute units per core.

We were unable to explain the reason for performance difference observed between m1.medium and m1.xlarge. Both these instances had the same CPU model hence the same clock speed and cache size. Only difference in CPU is m1.xlarge had more threads per core and this doesn’t explain the performance improvement in a serial program.

**Reasoning:** Difference in clock speed and cache size can explain the performance difference between instances having same compute units per core.

### 4.2.2 Memory scaling

**Observation 3:** Memory usage of FFT was calculated mathematically to compare the actual and calculated memory limit.

The memory usage for each problem size was calculated mathematically by considering the size of main arrays used in the program. There are three arrays of size N in FFT program, one for time domain samples, other one for frequency domain results and one as scratch space. Each element is a complex number containing two double Datatypes.

So, total memory = 3 * 2 * N * size of double; where N is the number of samples. The results shown below [Figure 26] is the problem size and approximate amount of
memory required to execute the problem for different values of \( N \) from \( 2^{25} \) to \( 2^{29} \), along with an indication of how much memory each standard EC2 instance provide.

Figure 26: FFT memory requirement comparison to EC2 Standard Instance memory

The results obtained [Figure 19] during memory limit test show that the problem size executed by each instance is under the expected limit. For example m1.small was able to execute \( 2^{25} \) problem size but not \( 2^{26} \). The Figure 26 also gives us the same inference. First problem size of \( 2^{25} \) is below the m1.small line (instance memory) whereas second problem size of \( 2^{26} \) if beyond the memory limit. The same memory behaviour holds true for other EC2 instances [D.10] [D.11].

**Reasoning:** Memory limit identified through benchmarking is the same as the memory limit calculated mathematically.

**Observation 4:** Very low performance of small instance type when compared to other instances type.

Small instance had only one EC2 compute unit per CPU, whereas medium and large instances had two compute units per core. So, it is expected of small instance to give poor performance in all benchmarks. The results for FFT and MD proved this true, but the variation in performance was not linear and small instance was very slow.
When we verified the swap memory allocated for each instance in `fstab` [64] using `swapon` command, we realised that the swap is enabled by default in m1.small EC2 instances[Figure 27].

```
root@master:/home/ajgeadmin# swapon -a
Filename      Type  Size   Used  Priority
/dev/mvds3    partition 917500 0       -1
```

**Figure 27: Swap Memory in m1.small**

To verify the difference we ran the FFT benchmark with $2^{26}$ input size in m1.small with and without swap memory (using `swapon` and `swapoff` command) and we observed that the application break immediately on reaching the memory limit when swap is turned off whereas it took more than 400 seconds to break when the swap memory is turned on.

This shows that, the swap memory helps to execute large problem size but it affects the performance due to page swapping into disk. We intend to perform detailed study in the effects of swap space in future.

In this case the additional swap memory allocated was not sufficient enough to scale for larger problem sizes. So the limit measured earlier will remain the same even after changing removing the swap in small instance. No other EC2 instances have swap memory turned on by default.

**Reasoning:** The low performance in small instance can be due to the swap memory used.

### 4.2.3 Stability of Performance

**Observation 5:** Good stability of results in Cluster Compute and High Memory instances compared to other instances

From the results of MD benchmark we analysed the variation in three consecutive executions in same instance type with same input parameters. We calculated the standard deviation of these results to summarise the stability of performance in each Instance types.
The results show a significant variation of results obtained during multiple execution of MD benchmark. It is observed that there is less deviation in Cluster Compute and High Memory instances.

### 4.2.4 Cost effective solution

With the results obtained from the molecular dynamics benchmark, we did an analysis to find the most cost effective EC2 instance for a sample scenario of 1000 steps measured by total cost. The benchmark calculates time for one step for interaction between particles, so we estimated the time and cost required for 1000 steps.

![Figure 28: Standard deviation of execution for 2^16 problem size](image)

![Figure 29: Cost Efficiency Calculation for MD benchmark Amazon EC2](image)
This results in Figure 29 show that the cost effective solution for our Molecular Dynamics algorithm with a problem size of $2^{26}$ executing 1000 steps is High CPU Medium Instance (c1.medium).

We also found that the fastest instance for this scenario is the Eight Extra Large cluster compute instance, which also the most expensive instance is provided by Amazon EC2. This calculation can be rerun for different problem sizes.

4.3 Analysis of Hawk-i

Hawk-i tool can be considered as a good prototype for live benchmark tool. There is significant amount of design and coding required for creating a tool that provides full functionality and usability. In addition to that feedback from someone who is a potential end user like a mathematician or scientist would have been very beneficial in deciding the features for a live benchmark tool. After all a system should be built for the user taking into account of his/her needs.

The computation time and memory limit certainly gives an idea of how big of a problem can be executed and how long it will take to execute a particular problem type. The classification into 13 motifs is very useful in standardising the benchmarks added into Hawk-i.

It is a good feature that users have the flexibility to upload their own code for benchmarking and this is a simple task for a programmer, but we can’t be certain of this simplicity until we get it tested by a scientific user who is less skilled in programming or who just know how to run an existing scientific package. A product survey should be done to get feedback on this product from the scientific community.

The development approach and programming model used for this project end-up somewhere between iterative and code like hell. It is more towards iterative model since there were new features added during each phase of development but there was no documentation for these changes which is a feature of code like hell.

If this tool is delivered as a service, the cost efficiency is low due to the hourly billing policies of AWS. Most of the existing benchmarks execute only for few minutes but the cost of execution is metered for an hour. There is a scope of improvement by holding an instance (delay termination) for an hour to reuse it for simultaneous execution. This can also reduce the start-up which is not significant to the benchmarking results.

Careless benchmarking by end users can result in huge credit loss. (For example, a user can accidently choose to create a cluster with 16 instances using web interface or even thousands of instances using admin command line), and there should be some mechanism to limit this kind of execution. Another good approach can be to allow users to bear the execution cost. Users should be prompted for user’s amazon account details for registration. The cluster can easily be created and benchmarked with the user’s account details and this also makes the users more responsible during benchmarking, but it can be modified to as a system that accepts user credentials.
All these benchmark executions were made in the default region (us-west). There should be an option for users to select the region before executing the benchmark since the actual application will be launched in a region of user’s preference and not in the default region.

Terminating the cluster only after confirmation from the user can be beneficial in many ways. For example, if the benchmarking is done in the user’s expense and the user is satisfied with the performance, he should be able to use the same cluster from his account. Another case can be an advanced user who needs to debug the cluster has to create another cluster from admin command line tool. With the current design, an option to choose whether to terminate the cluster after benchmarking can be added with the existing command line options as well as in web interface.

From the large amount of results obtained, it is quite evident how easily numerous benchmarks can be executed. Each benchmark execution is independent and exclusive and the user doesn’t have to wait for the results. There should be an option to log the detailed execution results to verify or debug the benchmarks used.

Hawk-i should provide an extensive admin panel to edit or delete existing benchmarks and inputs in the system. The results recorded should also be configurable by the admin. For example, if Hawk-i is to be used by a technical user who is planning to measure hardware performance, instead of storing the calculation time, the user should be able to choose what data (clock speed, I/O Bandwidth or any other metric) is to be stored.

Hawk-i can be modified to be used for a different cloud provider or a private cloud which provide EC2 APIs. The concept of creating a similar tool in other cloud services is the same. By adding a wide variety of cloud services and enabling comparison between different vendors, it can be a very powerful benchmarking tool or service.

The benchmarks created were useful to measure the performance of each kind of application as described by the 13 motifs classification. Some evidence to test the relation between a motif and a real application would have made the discussion more interesting.

Addition of parallel FFT and MD benchmarks would have helped in identifying the issues with communication pattern described for the motifs. This would have provided us with interesting data which may give us more details on parallel performance of a cluster built in vendor cloud.

Even though there are many missing features required for a full scale live benchmarking tool, Hawk-i was successful in proving how useful a web tool can be to execute benchmarks a cloud infrastructure.
Chapter 5
Conclusions and Future Work

5.1 Conclusions

We have successfully created a working prototype of a live benchmark tool and investigated how different scientific algorithms scale in cloud instances. We were able to prove how easily virtual clusters can be built, benchmarked and terminated in Amazon EC2 cloud using Hawk-i tool. With the large amount of execution data collected using Hawk-i, we were able to extend our study to behaviours of different instance types and how these features can be used to select a suitable cluster for an application.

Hawk-i implementation was possible within this short period of time due to the effort spent in perfecting the design of web interface, benchmarks and manager system. Choosing the right environment and tools was also important for this project. Since we lacked previous experience in cloud computing projects, we took an iterative approach. Once we had a proper foundation (environment, tools, manager and 13 motifs), rest of the construction work could be done with little deviation from the design. Even though new features like benchmark submission and threshold were added later on the core system remained the same.

We found Hawk-i very useful when we put the system to test by creating benchmarks like FFT, MD and Ping pong. The integration of new benchmarks was hassle free and the execution methods were even better through the web interface. Even though the project was successful in meeting the basic requirement of collecting execution time and memory limit, a provision to measure and archive few more performance and hardware statistics from the virtual cluster would have been interesting for detailed analysis of EC2 features.

During detailed analysis of results we made the following observations and conclusions:

- Multiple instances launched using same EC2 instance type showed different performance due to difference in CPU model allocated for each launch.
- Significant performance difference between standard instances having same compute unit was observed due to difference in clock speed cache size.
- Memory limit identified through benchmarking is the same as the memory limit calculated mathematically.
- Very low performance of small instance type when compared to other instances type can be due to memory swap enabled in small instance.
- Good stability of results in Cluster Compute and High Memory instances compared to other instances

We were also successful in estimating the actual cost a molecular dynamics simulation scenario. Through this study we were able to identify a cost effective solution for that particular problem size.

We identified areas in which we need more research and analysis to provide a clear reasoning. These are discussed in the next section.

5.2 Future Work

By creating this live benchmark tool, we have proved how easy it is to execute an application in an HPC virtual cluster built in a public cloud. There are many possibilities of improving Hawk-i and providing it as an online cloud benchmarking service or even a cluster computing platform.

We have identified the following modifications that can improve our Hawk-i implementation.

- We have only created two serial benchmarks (Spectral Methods and N-Body Methods) out of the thirteen possible motifs. These existing benchmarks can be parallelised and new benchmarks can be added to Hawk-i system.
- Cost of execution was a major challenge faced during live benchmarking. Adding features to create an Amazon EC2 sub account for newly registered users can be a practical approach, especially when we are providing this tool to a group of trusted users. A separate billing system can also be implemented for a wide range of users who access the tool via internet.
- There is a possibility to conduct further research to find a cost effective method to add a 24x7 benchmark so that the stability of cloud instances can be measured continuously.
- Collecting more statistics like CPU and Memory and I/O performance can help in detailed analysis of the cloud instances for advanced users.

Besides these modifications, we have also identified the scope of a few related research works which are discussed below.

Whenever we found a discussion on cloud computing and HPC, we saw a common question: “Can clouds replace the existing HPC systems?” From our primary observation, we found that the cloud cannot provide consistent stability of performance like a cluster. A detailed study using the same type benchmarks based on 13 motifs to compare cluster computing and cloud computing can give a clear answer to these questions.
Creating a micro benchmark suite based on the 13 motifs can be useful not only to Hawk-i, but also to actual clusters. Similar work is being done for GPU computing [65].

In our detailed analysis, we discussed how running pilot jobs to acquire the best virtual machines in a public cloud can be an efficient way to get the best performance for the least cost. Research work on creating such an application can be useful to cloud users. The basic idea we suggest is to create a cluster with small or medium instances, verify the performance of each node and terminate the poorly performing nodes. New nodes can be added to the existing cluster and tested repeatedly to get the required configuration of instances with good performance.

Hawk-i benchmark source is available to public in Github Repository [Appendix F]. We welcome collaborators to contribute to the development of Hawk-i tool and submission of new parallel benchmarks.
Appendix A

Hawk-i User Manual

1. GETTING STARTED
   1.1 Logging On

   Public URL: ec2-23-20-179-14.compute-1.amazonaws.com/hawki

   Access details are available in project wiki.

   A hawk-i system has 3 levels of users: Administrator, Manager and normal user. Administrators and manager interact with the system and they require authentication to access the tools.

   They can login to Hawk-i using a web form on the Hawk-i home page. Username and password details have to be provided by the administrator. A set of demo user credentials is available in the project wiki page.

   1.2 System Menu

   The Navigation menu items provided by the Hawk-i Web interface are different for different users

   Administrators have access to all menu items which include:

   a) Home: Hawk-i home page
   b) Benchmarks : Allowing access to benchmark functionality
   c) Results: Allow Users to view benchmark history
   d) Benchmark Manager: Access to adding new benchmarks and Inputs

   Now the managers can only run benchmarks added by the administrator.

   1.3 Logout

   Users can logout from the system using the logout button in the left side of all pages.

2. USING THE SYSTEM IN WEB INTERFACE
   2.1 Run Benchmarks
In the first input form under benchmarks tab, the user selects benchmark name, instance type and number of instances and clicks the “Select input size” button.

On submitting this form a new form appears below to enter the input size. Select an input size and submit the form by clicking the “Run Benchmark” button and wait for few minutes to retrieve the entire execution results. Alternatively they can view the execution results summary in Results tab.

2.2 Results

Under this tab users can view the history of their benchmark executions in a sortable table. You can sort each column by clicking the column heading.

2.3 Benchmark Manager

Only the administrators have access to this module. This module is used to submit new benchmarks and input parameters.

The first form in this page is to add a new benchmark.

You can download the benchmark template file from this page by clicking the template link.

untar the template and update the contents of this file following the Instructions provided in the template.

Once you have the new benchmark created according to the specification, tar the benchmark folder and upload tar ball in the form provided in benchmark manager. You can add a small description about the benchmark as well.

The next important thing is to enter the different command line parameters in comma separated format. Each set of parameters will appear as a select option in benchmark tool for this new benchmark.

Once you have added all these details, submit the form by clicking the “Submit” button in first form.

The second form allows you to add new input for any benchmark uploaded.

In this form, you have to select the benchmark name and enter the input parameters in comma separated format and then submit the form using the “Submit” button in second form.

3. USING THE SYSTEM IN COMMAND LINE MODE
You have to get access to the manager instance using the pem file saved in project wiki with username ‘ubuntu’. The details of gaining this access are explained in Appendix C.

Once you have SSH access to manager instance, you can run the manager script with different input options

Usage: manager.py [options]

Options:

- **-h, --help** show help message and exit
- **-N CLUSTERNAME** Name of cluster (string)
- **-i IMAGEID** Image id (string) [eg: ami-999d49f0]
- **-t INSTANCETYPE** Type of instance (string) [eg: m1.small]
- **-s SIZE** Cluster Size (int)
- **-b FILE** Path to benchmark code and scripts (folder)
- **-u USERID** Web userid (for web interface)
- **-r FILE** Path to benchmark results folder
- **-a ARCHIVE** Archive results to database (0/1)
- **-d INPUTDATA** Input parameters (to change input files/parameters)
- **-T THRESHOLD** Threshold (seconds)

Example: **Hawk-i $ ./manager.py -t cc1.4xlarge -s 2 -i ami-4583572c -b ./benchmark/pingpong -d "1024 1000"**

This executed the pingpong benchmark in cc1.4xlarge cluster with two nodes for a command line input of “1024 1000”
Appendix B
AMI Details

1. Name: ami-999d49f0 (x86_64)
   Supported Instances:
   
   m1.small, m1.medium, m1.large, m1.xlarge
   m2.xlarge, m2.2xlarge, m2.4xlarge
   c1.medium, c1.xlarge
   
   Operating system: Ubuntu 11.10

2. Name: ami-4583572c (HVM)
   Supported Instances:
   
   cc1.4xlarge, cc2.8xlarge
   cg1.4xlarge
   
   Operating system: Ubuntu 11.10
Appendix C
Manager Instance Configuration

You must have registered with Amazon Web Services for doing the following steps.

Launch an amazon instance (t1.micro):


User: ubuntu

Private Key for ssh authentication will be provided during instance creation. Keep it safe since it is not possible to get another key from amazon.

In you terminal:
$ chmod 400 hawki.pem
$ ssh -i hawki.pem ubuntu@ec2-23-20-179-14.compute-1.amazonaws.com

If everything goes well, we will get remote access to the running instance.
Note: The username for each type of instance is different and it is provided in manual for each instance.

I am using 64 bit Ubuntu instance and I have installed LAMP (Linux Apache MySQL and PHP) in this instance for publishing Live Benchmark Data. Accessible through http://ec2-23-20-179-14.compute-1.amazonaws.com

Steps:
$ sudo su

apt-get install tasksel

tasksel install lamp-server

service apache restart

pem file is stored

Now we have EC2 manager instance with web server. Now we have to install Starcluster using easy install

$ sudo easy_install starcluster
Starcluster configuration has to be modified in ~/.starcluster/config file. Instructions to modify the configuration is defined in this file is provided within the config file. We have to change only the access credentials. Remaining settings are accepted by manager script in command line.

Now you have the environment setup and ready to deploy the application that is stored in Github.

https://github.com/visakhmr/hawki-cloud-benchmark.git

Setup GIT: https://help.github.com/articles/set-up-git

And pull the entire project to a directory called Hawk-i in ‘ubuntu’ home directory.

The documentation to create new Joomla modules for additional functionality is available in: http://docs.joomla.org/Creating_a_simple_module
Appendix D
Benchmark data

Note: Blank data in all tables represents a failed execution.

D.1 Micro Instance consecutive benchmark execution time (secs)

<table>
<thead>
<tr>
<th>Execution count</th>
<th>m1.small</th>
<th>t1.micro</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15.828069</td>
<td>6.665506</td>
</tr>
<tr>
<td>2</td>
<td>15.811795</td>
<td>69.393741</td>
</tr>
<tr>
<td>3</td>
<td>15.829731</td>
<td>26.371656</td>
</tr>
<tr>
<td>4</td>
<td>15.852331</td>
<td>69.442591</td>
</tr>
<tr>
<td>5</td>
<td>15.717949</td>
<td>7.168759</td>
</tr>
</tbody>
</table>

Table 5: t1.micro consecutive benchmark comparison with m1.small

D.2 EC2 Instances Start-up time (secs)

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Start-up time</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>137</td>
</tr>
<tr>
<td>m1.medium</td>
<td>113</td>
</tr>
<tr>
<td>m1.large</td>
<td>109</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>106</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>118</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>142</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>128</td>
</tr>
<tr>
<td>c1.medium</td>
<td>114</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>107</td>
</tr>
</tbody>
</table>

Table 6: EC2 Instances Start-up time
D.3 Time scaling for memory benchmark (secs)

<table>
<thead>
<tr>
<th>Size</th>
<th>m1.small</th>
<th>m1.medium</th>
<th>m1.large</th>
<th>m1.xlarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^19$</td>
<td>3.18</td>
<td>2.32</td>
<td>2.02</td>
<td>2.03</td>
</tr>
<tr>
<td>$2^20$</td>
<td>3.17</td>
<td>2.94</td>
<td>9.27</td>
<td>4.56</td>
</tr>
<tr>
<td>$2^21$</td>
<td>8.79</td>
<td>5.42</td>
<td>4.79</td>
<td>5.31</td>
</tr>
<tr>
<td>$2^22$</td>
<td>12.63</td>
<td>10.86</td>
<td>17.63</td>
<td>10.92</td>
</tr>
<tr>
<td>$2^23$</td>
<td>48.75</td>
<td>23.38</td>
<td>39.59</td>
<td>39.67</td>
</tr>
<tr>
<td>$2^24$</td>
<td>123.25</td>
<td>48.91</td>
<td>69.35</td>
<td>76.75</td>
</tr>
<tr>
<td>$2^25$</td>
<td>276.64</td>
<td>134.05</td>
<td>147.6</td>
<td>102</td>
</tr>
</tbody>
</table>

Table 7: Time scaling for FFT benchmark

D.4 Standard Instance memory limit and execution time (secs)

<table>
<thead>
<tr>
<th>Size</th>
<th>m1.small</th>
<th>m1.medium</th>
<th>m1.large</th>
<th>m1.xlarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^25$</td>
<td>276.64</td>
<td>134.05</td>
<td>147.6</td>
<td>102</td>
</tr>
<tr>
<td>$2^26$</td>
<td>319.1</td>
<td>39.46</td>
<td>212.4</td>
<td></td>
</tr>
<tr>
<td>$2^27$</td>
<td></td>
<td>453.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^28$</td>
<td></td>
<td>1055.76</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Standard instances memory limit and execution time

D.5 High Memory Instance memory limit and execution time (secs)

<table>
<thead>
<tr>
<th>Size</th>
<th>m1.xlarge</th>
<th>m2.xlarge</th>
<th>m2.2xlarge</th>
<th>m2.4xlarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^25$</td>
<td>102</td>
<td>90.61</td>
<td>86.29</td>
<td>77.77</td>
</tr>
<tr>
<td>$2^26$</td>
<td>212.4</td>
<td>169.65</td>
<td>169.19</td>
<td>169.06</td>
</tr>
<tr>
<td>$2^27$</td>
<td>453.38</td>
<td>406.59</td>
<td>381.71</td>
<td>376.79</td>
</tr>
<tr>
<td>$2^28$</td>
<td>1055.76</td>
<td>880.53</td>
<td>891.71</td>
<td>896.96</td>
</tr>
<tr>
<td>$2^29$</td>
<td></td>
<td>1911.01</td>
<td>1899.86</td>
<td></td>
</tr>
<tr>
<td>$2^30$</td>
<td></td>
<td></td>
<td>3939.56</td>
<td></td>
</tr>
</tbody>
</table>

Table 9: High memory instances memory limit and execution time

D.6 High CPU Instance memory limit and execution time (secs)

<table>
<thead>
<tr>
<th>Size</th>
<th>m1.medium</th>
<th>m1.xlarge</th>
<th>c1.medium</th>
<th>c1.xlarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^25$</td>
<td>134.05</td>
<td>102</td>
<td>186.46</td>
<td>110.77</td>
</tr>
<tr>
<td>$2^26$</td>
<td>319.1</td>
<td>212.4</td>
<td>242.01</td>
<td></td>
</tr>
<tr>
<td>$2^27$</td>
<td>453.38</td>
<td></td>
<td>714.73</td>
<td></td>
</tr>
</tbody>
</table>

Table 10: High CPU instances memory limit and execution time
D.7 Cluster Compute Instance memory limit and execution time (secs)

<table>
<thead>
<tr>
<th>Size</th>
<th>m2.4xlarge</th>
<th>cc1.4xlarge</th>
<th>cc2.8xlarge</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{25}$</td>
<td>77.77</td>
<td>83.46</td>
<td>85.67</td>
</tr>
<tr>
<td>$2^{26}$</td>
<td>169.06</td>
<td>163.97</td>
<td>179.95</td>
</tr>
<tr>
<td>$2^{27}$</td>
<td>376.79</td>
<td>372.13</td>
<td>384.36</td>
</tr>
<tr>
<td>$2^{28}$</td>
<td>896.96</td>
<td>875.19</td>
<td>850.35</td>
</tr>
<tr>
<td>$2^{29}$</td>
<td>1899.86</td>
<td></td>
<td>2084.23</td>
</tr>
<tr>
<td>$2^{30}$</td>
<td>3939.56</td>
<td></td>
<td>4499.04</td>
</tr>
</tbody>
</table>

Table 11: Cluster Compute instances memory limit and execution time

D.8 MD benchmark results for $2^{16}$ problem size

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Execution time (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>637.32</td>
</tr>
<tr>
<td>m1.medium</td>
<td>315.38</td>
</tr>
<tr>
<td>m1.large</td>
<td>300.44</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>303.41</td>
</tr>
<tr>
<td>c1.medium</td>
<td>283.10</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>278.00</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>214.49</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>215.25</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>212.57</td>
</tr>
<tr>
<td>cc1.4xlarge</td>
<td>198.26</td>
</tr>
<tr>
<td>cc2.8xlarge</td>
<td>186.42</td>
</tr>
</tbody>
</table>

Table 12: MD benchmark results
D.9 Ping pong benchmark results array size $2^{18}$ iterations 1000

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Latency (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>0.106728</td>
</tr>
<tr>
<td>m1.medium</td>
<td>0.037796</td>
</tr>
<tr>
<td>m1.large</td>
<td>0.028307</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>0.012192</td>
</tr>
<tr>
<td>c1.medium</td>
<td>0.022473</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>0.020348</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>0.021864</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>0.018857</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>0.018504</td>
</tr>
<tr>
<td>cc1.4xlarge</td>
<td>0.004929</td>
</tr>
<tr>
<td>cc2.8xlarge</td>
<td>0.00432</td>
</tr>
</tbody>
</table>

Table 13: Ping pong benchmark results

D.10 FFT problem size and memory required

<table>
<thead>
<tr>
<th>Size</th>
<th>Memory Required (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^{25}$</td>
<td>1.5</td>
</tr>
<tr>
<td>$2^{26}$</td>
<td>3</td>
</tr>
<tr>
<td>$2^{27}$</td>
<td>6</td>
</tr>
<tr>
<td>$2^{28}$</td>
<td>12</td>
</tr>
<tr>
<td>$2^{29}$</td>
<td>24</td>
</tr>
<tr>
<td>$2^{30}$</td>
<td>48</td>
</tr>
<tr>
<td>$2^{31}$</td>
<td>96</td>
</tr>
</tbody>
</table>

Table 14: FFT problem size and memory required
D.11 Available Memory in EC2 Instances

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Memory Available(GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>1.7</td>
</tr>
<tr>
<td>m1.medium</td>
<td>3.7</td>
</tr>
<tr>
<td>m1.large</td>
<td>7.5</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>15</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>17.1</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>34.2</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>68.4</td>
</tr>
<tr>
<td>c1.medium</td>
<td>1.7</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>7</td>
</tr>
<tr>
<td>cc1.4xlarge</td>
<td>23</td>
</tr>
<tr>
<td>cc2.8xlarge</td>
<td>60.5</td>
</tr>
<tr>
<td>cg1.4xlarge</td>
<td>22</td>
</tr>
<tr>
<td>hi1.4xlarge</td>
<td>60.5</td>
</tr>
</tbody>
</table>

Table 15: Available memory in EC2 Instances

D.12 Stability of results

<table>
<thead>
<tr>
<th>Instance Type</th>
<th>Mean(Sec)</th>
<th>Standard Deviation(Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1.small</td>
<td>637.3233</td>
<td>3.870662</td>
</tr>
<tr>
<td>m1.medium</td>
<td>315.3767</td>
<td>17.66945</td>
</tr>
<tr>
<td>m1.large</td>
<td>300.44</td>
<td>20.74568</td>
</tr>
<tr>
<td>m1.xlarge</td>
<td>303.41</td>
<td>24.13985</td>
</tr>
<tr>
<td>m2.xlarge</td>
<td>214.4867</td>
<td>1.062837</td>
</tr>
<tr>
<td>m2.2xlarge</td>
<td>215.25</td>
<td>2.086864</td>
</tr>
<tr>
<td>m2.4xlarge</td>
<td>212.5667</td>
<td>0.787923</td>
</tr>
<tr>
<td>c1.medium</td>
<td>283.1033</td>
<td>14.24597</td>
</tr>
<tr>
<td>c1.xlarge</td>
<td>278.0033</td>
<td>5.31747</td>
</tr>
<tr>
<td>cc1.4xlarge</td>
<td>198.2633</td>
<td>1.050598</td>
</tr>
<tr>
<td>cc2.8xlarge</td>
<td>186.42</td>
<td>0.500067</td>
</tr>
</tbody>
</table>

Table 16: Stability of results
Appendix E
EC2 API Tools Configuration

EC2 instance can be controlled from a local machine using command line tools.
Steps to install command line tool :

Install Amazon EC2 API Tools

Instantiate your images(s)

Configure your instance

# Configurations done as root in Manager instance
apt-get install gnupg
gpg --gen-key

copy https://aws.amazon.com/security/ec2-pkgs-public-key/ to ec2-packages-public.key
gpg --import ec2-packages-public.key

note "pg: key 0349E66A: 

nopt --fingerprint <key-value>
gpg --edit-key <key-value>
  > trust
  > 4
  > sign
  > y
  > save

egoryID=88

Create ec2-api-tools.zip.asc and copy http://s3.amazonaws.com/ec2-downloads/ec2-api-tools.zip.asc to new file
gpg --verify ec2-api-tools.zip.asc ec2-api-tools.zip
# if good signature unzip and install ec2 tools
#Tell shell where tools are: (include in .bashrc)
export EC2_HOME=<path-to-tools>
export PATH=$PATH:$EC2_HOME/bin

#Tell tools who you are
#-get access
get Access keys

#create a folder .ec2/ and place .pem files there

export EC2_PRIVATE_KEY=<path>/ec2/pk-HKZYKTAIG2ECMXYIBH3HXV4ZBEXAMPLE.pem
export EC2_CERT=<path>/ec2/cert-HKZYKTAIG2ECMXYIBH3HXV4ZBEXAMPLE.pem

#-----Notes:
#If Java is not present,
apt-get install openjdk-jre-7
export JAVA_HOME=/usr

#create bashrc(root) file for path to be persistant

YES now we have CLT installed in manager instance

Appendix F
Source Code and Repository

The entire project source code is available in a Github public repository:

https://github.com/visakhmr/hawki-cloud-benchmark.git

Relative path from repository folder is provided for all source files.

F.1 manager.py
environment/manager.py

F.2 fft.c
benchmark/fft/fft.c

F.3 fft_benchmark.c
benchmark/fft/fft_benchmark.c

F.4 SGE file
benchmark/template/benchmark.sge

F.5 MD.c
benchmark/md/MD.c

F.6 pingpong.c
benchmark/pingpong/pingpong.c
References


[30] K. Asanovic, R. Bodik, B. C. Catanzaro et al., The landscape of parallel computing research: A view from berkeley, EECS Department, University of
California, Berkeley, 2006.


