AUTONOMIC RESOURCE ALLOCATION AND DELAY OPTIMIZATION OF DISTRIBUTED APPLICATIONS ON CLOUD

A Project Report

submitted by

PRANAV NAIR [EE09B108]

in partial fulfilment of the requirements

for the award of the degrees of

BACHELOR OF TECHNOLOGY

&

MASTER OF TECHNOLOGY

in

ELECTRICAL ENGINEERING

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2014

THESIS CERTIFICATE

This is to certify that the thesis titled **Autonomic Resource Allocation and Delay Optimization of Distributed Applications on Cloud**, submitted by **Pranav Nair [EE09B108]**, to the Indian Institute of Technology, Madras, for the award of the degrees of **Bachelor of Technology & Master of Technology**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Ramkrishna Pasumarthy
Project Guide
Dept. of Electrical Engineering
IIT Madras, Chennai 600 036

Dr. Harishankar Ramachandran
Head
Dept. of Electrical Engineering
IIT Madras, Chennai 600 036

Place: Chennai

Date: 5th May, 2014

ACKNOWLEDGEMENTS

I would like to express my appreciation and gratitude to my advisor **Dr. Ramkrishna Pasumarthy** for giving me the opportunity to work on this highly exciting and challenging project. Your invaluable guidance and motivation, throughout the project helped me grow as a researcher. I also thank the members of the **Cloud Computing** lab, **Mr. Saikrishna** and **Mr. Sarath Malipeddi** for their valuable help during the study. Lastly I would like to thank my family and friends for their constant motivation and encouragement.

ABSTRACT

Cloud computing is the emerging paradigm of computing service. In this work, we present a resource allocation algorithm for parallelizable scientific computing and data analytics jobs being serviced by a cloud environment. Eucalyptus, an open source cloud framework was studied and implemented as the test bed for conducting the experiments in a distributed manner. Distributed algorithm of k-means clustering of data has been implemented on the Eucalyptus test bed. The parallelization is achieved by distributing the data and the computing on several different virtual machines. Message Passing Interface (MPI), a leading standard for message passing is used for communication between the nodes running on the cloud. The response times of these jobs have been observed to vary with the size of the jobs and the number of resources allocated to these. Hence a model of the response times as a function of size and resources have been developed. This model can be used to predict the response times of incoming jobs. Based on these response times, we propose a resource provisioning algorithm that seeks to minimize the delay expected by the user. The proposed algorithm has been simulated and shown to be superior to a simple sharing algorithm in terms of delay minimization.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

SLA	Service Level Agreement
CLC	Cloud Controller
CC	Cluster Controller
SC	Storage Controller
NC	Node Controller
DHCP	Dynamic Host Configuration Protocol
MAC	Media Access Control
EMI	Eucalyptus Machine Image
MPI	Message Passing Interface

CHAPTER 1

INTRODUCTION

Computing in the cloud lets the user purchase computing resources as a metered service over a network (typically the Internet) and not as a product. This has been made possible due to the advances in Operating System Virtualization and the Internet. Cloud computing allows almost any server environment to be replicated and scaled instantly. Several companies find Cloud Computing much more economical than setting up the hardware infrastructure on their own.

Buyya *et al.* (2009) provides insight into the definition of cloud computing and the various architectures and also talks about the vision and reality about delivering computing as a service. Cloud computing has been identified as the most promising paradigm to provide computing as a service. It is the next generation of enterprise data centers, providing extreme scalability and fast access to users. Cloud computing is still at an early stage, with a motley crew of providers delivering a slew of cloud-based services, from full-blown applications to storage services. Characteristics of Cloud like autonomous and dynamic provisioning, scalability, optimization benefits, networked access and metered servicing have propelled this technology to the forefront Gong *et al.* (2010).

Among the three standard models of deployment of Cloud Gibson *et al.* (2012), SaaS is the the most popular and widely used model of Cloud computing. SaaS uses the Internet to deploy applications on the Cloud that are managed by third party vendors He (2010). Users access the application via the web browser on their side. This eliminates the need to download and install software to run these. SaaS enables users to use applications without being responsible for maintaining the data, O/S, virtualization, servers, storage, and networking. In the SaaS model of deployment, the user is expected to only give the inputs required for the particular applications to get back the outputs without being responsible for computer administration, software installation or system details. System details include properties like the number of nodes to be used, the amount of storage needed and the operating system deployed etc.

Scientific computing is one such SaaS application that can be delivered over the Cloud Yang *et al.* (2011); He *et al.* (2010). Scientific computing involves the construction of mathematical models and numerical solution techniques to solve scientific, social scientific and engineering problems on distributed systems Jakovits and Srirama (2013). Grid Computing gained high popularity in the field of scientific computing through the distributed resource sharing models deployed in academic institutions. Scientific computing is a high-utilization workload requiring huge number of computing resources traditionally employed on Grids.

Ostermann *et al.* (2009) tells us how Cloud Computing can be an attractive alternative to Grid Computing for scientific computing scientists. Since a cloud promotes leasing resources than setting up one's own infrastructure, this is an economic alternative for academic institutions. This also eliminates the burden of constantly updating the hardware due to advances in technology. The cloud eliminates the overhead costs arising due to over-provisioning of occasionally needed resources. The cloud can scale up resources in a fast and cost effective fashion. Despite these scientific computing on the Cloud also presents some challenges like slower interconnects, the pricing model, data management, the resource allocation model, security, etc Kaur and Chana (2010).

The contribution of this work is to be able to provide parallelizable scientific computing applications on a Cloud environment. The setting up of the Cloud environment Eucalyptus has been studied in detail and implemented. K-Means Clustering, a frequently encountered problem in scientific computing has been chosen and a parallelized algorithm has been implemented on the cloud environment. Message Passing Interface, a leading standard for message passing libraries has been studied and has been implemented for parallelizing the jobs. The response times of these jobs are modeled as functions of the size of the jobs and the resources allocated. Furthermore a resource allocation algorithm has been proposed that seeks to reduce the overall delay over the deadline for the jobs.

CHAPTER 2

Cloud Computing - Overview and Eucalyptus Setup

2.1 Definition and Characteristics

Cloud Computing is defined as the model for delivering computing resources as a service over the Internet. The characteristics and benefits of cloud computing are

1. Self-service Provisioning: The cloud allows users to deploy their own sets of computing resources (machines, network, storage, etc.) as needed without the delays and complications typically involved in resource acquisition.
2. Dynamic Provisioning: The provisioning of resources can be done based on the current demand for the services.
3. Elasticity and Scalability: Unlike individual users for whom the usage is typically fluctuating, a cloud can easily accommodate rapid increases or decreases in resource demand. Thus the user can utilize as per usage and avoid the cost of idle infrastructure.
4. Optimization Benefits: The cloud can maximize the usage and increase the efficiency of existing infrastructure resources. This can help reduce capital expenditure and extends infrastructure lifecycle.
5. Network Access and Storage Virtualization: Cloud services can be accessed from anywhere via the Internet and from any type of device. They should also provide storage capability independent of device and location.
6. Metered Services: The usage of resources is metered and the consumers are billed accordingly.

Cloud computing has several advantages. It eliminates the need for the clients to set up and maintain their own physical servers, thus reducing expenses. The clients are billed only as per their usage. Dynamic reallocation of resources ensures that the servers are utilized more efficiently. Also network access ensures that the client can access these services from anywhere. Because of these factors, cloud computing is a fast growing field. Currently Amazon, Google and Microsoft are some of the big names in this field.

2.2 Service Model Classification

Depending on the service models, clouds are classified as

1. Software as a Service (SaaS): In this model the user purchases the ability to use a software application or service on the cloud. Eg: Google Docs
2. Platform as a Service (PaaS): In this model the user purchases access to platforms, enabling them to deploy their own applications on the cloud. Eg: Google App Engine
3. Infrastructure as a Service (IaaS): In this model, the user is delivered infrastructure, namely servers, networks and storage. The user can deploy several Virtual Machines and run specific Operating Systems on them. eg: Amazon EC2, Windows Azure etc.

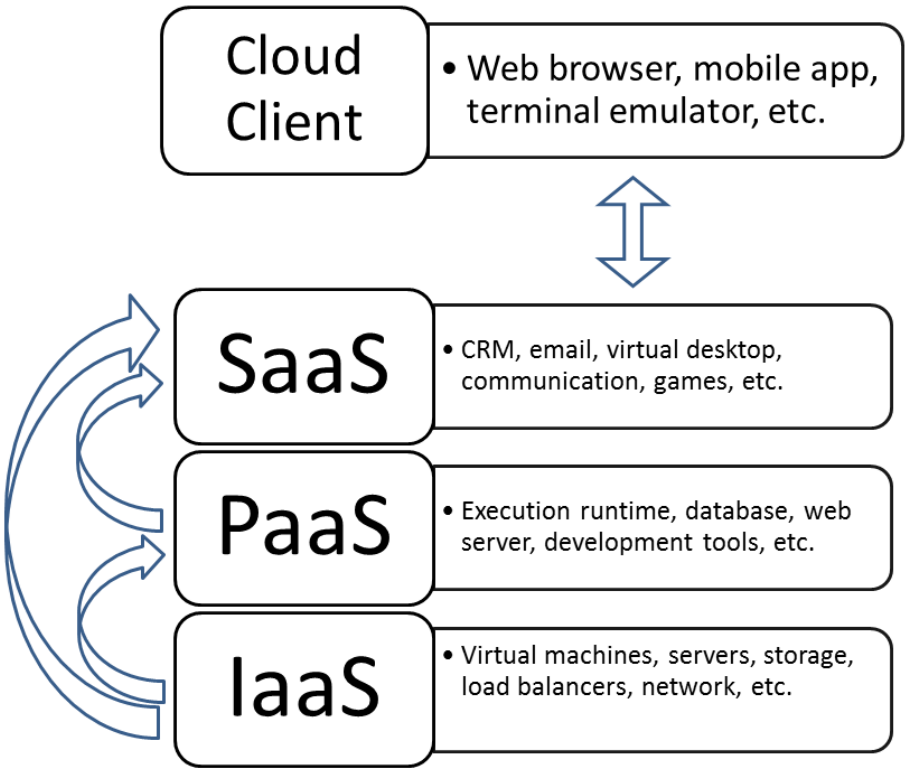

Figure 2.1: Service Models

2.3 Deployment Model Classification

Depending on the deployment models, clouds are classified as

1. Public Cloud: A public cloud can be accessed by any user with an internet connection and is intended for public use.

2. Private Cloud: A private cloud is established and operated solely by a specific group or organization and access is limited to that group.
3. Community Cloud: A community cloud is shared among several organizations with common concerns and similar cloud requirements.
4. Hybrid Cloud: A hybrid cloud is a combination of several clouds, where the clouds are a mixture of public, private or community.

2.4 Eucalyptus - Open Source Cloud Computing

Eucalyptus stands for **Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems**. It is a Linux based open source architecture that can be used to implement scalable private and hybrid clouds. The cloud will be deployed across the enterprise's on-premise IT infrastructure and can be accessed over its intranet.

Eucalyptus also supports Amazon Web Service-compatibility allowing our on-premise clouds to interact with these public clouds using a common programming interface. This API-compatibility with Amazon's EC2, S3, ELB, Auto Scaling, and CloudWatch services offers the capability of deploying hybrid clouds. It has support for multiple virtualization platforms like Xen, KVM and VMWare. It is also packaged and supported for multiple distributions like Debian, Ubuntu, Cent-OS, SuSE etc. Eucalyptus Systems (*a*) gives details about Eucalyptus 3.1.2 architecture and setup.

2.5 Architecture of Eucalyptus

Eucalyptus is comprised of six components: Cloud Controller (CLC), Walrus, Cluster Controller (CC), Storage Controller (SC), Node Controller (NC) and an optional VMWare Broker (Broker or VB).

Other than the VMWare Broker, each component is a stand-alone web service. This architecture allows Eucalyptus both to expose each web service as a well-defined, language-agnostic API, and to support existing web service standards for secure communication between its components.

The cloud components, Cloud Controller (CLC) and Walrus, communicate with cluster components, the Cluster Controllers (CCs) and Storage Controllers (SC). The

CCs and SCs, in turn, communicate with the Node Controllers (NCs). The networks between machines hosting these components must be able to allow TCP connections between them. However, if the CCs are on separate network interfaces (one for the network on which the cloud components are hosted and another for the network that NCs use) the CCs will act as software routers between these networks in some networking configurations. So each cluster can use an internal private network for its NCs and the CCs will route traffic from that network to a network shared by the cloud components. Virtual machines (VMs) run on the machines that host NCs. You can use the CCs as software routers for traffic between clients outside Eucalyptus and VMs. Or the VMs can use the routing framework already in place without CC software routers.

2.6 Description of the components

2.6.1 Cloud Controller (CLC)

The CLC is a Java program that acts as the web interface to the outer world. It is the entry point into the cloud for administrators, developers and end users. The CLC acts as the administrative interface for the cloud, querying the other components about resource availability and performing high-level resource scheduling by making requests to the Cluster Controllers (CC). The CLC can be accessed through command line interfaces like euca2ools to manage the compute, storage and network resources. Only one CLC can exist per cloud and it handles all authentication, accounting, reporting etc.

2.6.2 Walrus

Walrus is a Java program which is the Eucalyptus equivalent of the AWS Simple Storage Service (S3). Walrus offers users the ability to store persistent data, organized as buckets and objects, to all the virtual machines and can be used as a simple HTTP put/get storage as a service solution. There are no particular data type restrictions. Users can store application data as well as the images which are the building blocks used to launch the VMs. Volume snapshots, which are point in time copies of data, can also be stored on Walrus. Just like the CLC, only one Walrus per cloud is allowed.

2.6.3 Cluster Controller

Cluster Controller is written in C and acts as the front end of a particular cluster within the Eucalyptus Cloud. It executes on a machine that has connectivity to both the CLC and the Node Controllers (NC) and reports the NCs registered to the CLC. CC also gather the information about a set of NCs and schedules the VM execution on specific NCs. The CC also manages the virtual machine networks and all NCs within a single CC will belong to a single subnet.

2.6.4 Storage Controller

Storage Controller, written in Java, is the Eucalyptus equivalent of the Amazon's Elastic Block Storage. It can interface with various types of storage systems. It communicates with the Cluster Controller and Node Controller and manages the Eucalyptus block volumes and snapshots to the instances within its specific cluster. EBS volumes persist even after VM termination but cannot be shared between VMs and can only be accessed within the same availability zone in which the VM is running.

2.6.5 Node Controller

The Node Controller (NC) is written in C and it runs on the machine that hosts the VM instances. It runs on each node and interacts with the CC on one hand and the OS and the hypervisor on the other side. It controls VM activities like the execution, inspection and termination of VM instances. It downloads and creates caches of images and snapshots from the Walrus. It is also responsible for the management of the virtual network endpoint. There is no theoretical limit to the number of NCs per cluster but performance limits do exist.

2.7 Benefits of Eucalyptus

1. Eucalyptus has a modular and easy design which enables a variety of user interfaces and thus brings the benefits of virtualization to a broad spectrum of users like administrators, developers, managers and hosting customers.

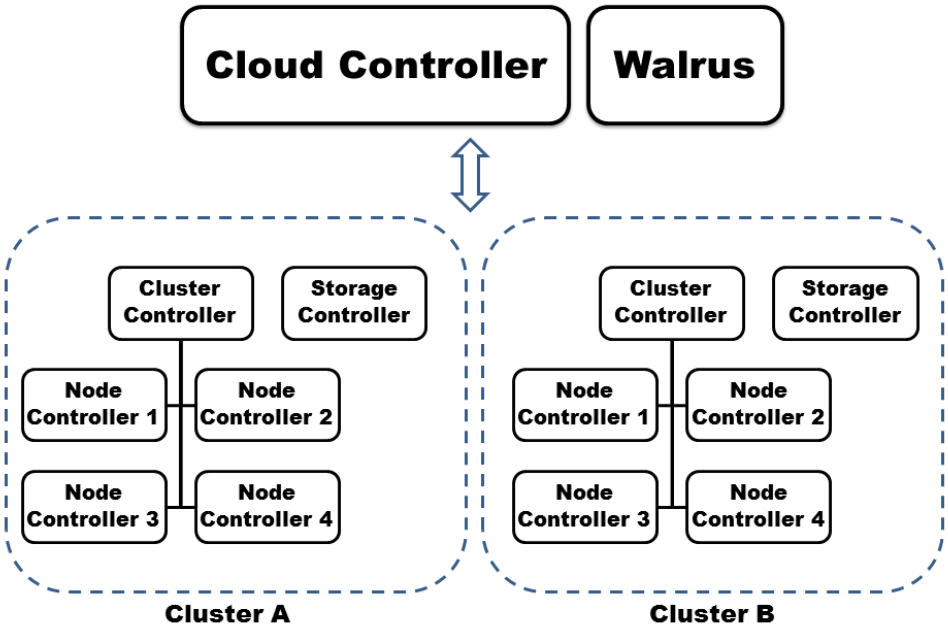

Figure 2.2: Eucalyptus Architecture

2. The snapshot feature provides opportunities to improve the reliability and automation of the cloud making it very easy to use and reduce average learning time for users, thus improving turnaround time for projects.

3. Supports existing virtualization technologies, Linux-based operating systems, and supports multiple hypervisors.

4. Since the core of the Eucalyptus project will continue to remain open-source, users can access the source code and leverage the contributions from a worldwide community of developers.

2.8 Eucalyptus Framework Setup

2.8.1 System Requirements

Compute Requirements

1. Physical Machines: All Eucalyptus components must be installed on physical machines, not virtual machines.

2. Central Processing Units (CPUs): It is recommended that each machine in your Eucalyptus cloud contain either an Intel or AMD processor with a minimum of two, $2GHz$ cores.

3. Operating Systems: Eucalyptus supports Ubuntu 12.04 LTS and some other Linux distributions.

4. Machine Clocks: Each Eucalyptus component machine and any client machine clocks must be synchronized (for example, using NTP) at all the time, not just at installation.

5. Hypervisor: Ubuntu 12.04 LTS installations must have KVM installed and configured on NC host machines. VMWare-based installations do not include NCs, but must have a VMWare hypervisor pool installed and configured.

6. Machine Access: Verify that all machines in your network allow root or sudo access and SSH login.

Storage and Memory Requirements

The following are recommended:

1. Each machine in your network needs a minimum of 30 GB of storage.

2. At least 100 GB for Walrus and SC hosts running Linux VMs.

3. 50-100 GB per NC host running Linux VMs.

4. Each machine in your network should have at least 4 GB RAM or above for improved caching.

Network Requirements

1. All NCs must have access to a minimum of 1 GB Ethernet network connectivity.

2. All Eucalyptus components must have at least one Network Interface Card (NIC) for a base-line deployment. For better network isolation and scale, the CC should have two NICs each with a minimum of 1 GB Ethernet (one facing the CLC/user network and one facing the NC/VM network).

3. Depending on some configurations, Eucalyptus requires that you make available two sets of IP addresses. The first range is private, to be used only within the Eucalyptus system itself. The second range is public, to be routable to and from end-users and VM instances. Both sets must be unique to Eucalyptus, not in use by other components or applications within your network.

2.8.2 Networking Mode

There different networking modes available for configuring Eucalyptus cloud are System, Static, Managed, Managed No VLAN. We use static mode of networking since Eucalyptus runs its own DHCP server and offers better control of VM instance IP assignment.

Static Mode

Static mode offers you control over instance IP address assignment. In Static mode, you configure Eucalyptus with a map of MAC address/IP Address pairs. When a VM is instantiated, Eucalyptus sets up a static entry within a Eucalyptus controlled DHCP server, takes the next free MAC/IP pair, assigns it to an instance, and attaches the instance's ethernet device to the physical ethernet through the bridge on the NCs.

2.8.3 Configuring Dependencies

Install Hypervisors on NC

1. Pre-installation checklist: To run KVM, you need to check if your processor supports hardware virtualization. The following command can be used to check.

   ```
   $ engrep -c '(vmx|svm)' /proc/cpuinfo
   ```

   If the Output is 0, it means that your CPU doesn't support hardware virtualization. If not it will show how many processors support virtualization.

2. We need to install necessary packages like `qemu-kvm`, `libvirt-bin`, `ubuntu-vm-builder`

3. Verify Installation: You can test if your install has been successful with the following command.

   ```
   $ virsh -c qemu:///system list
   ```

   Output must look like this:

   ```
   Id Name                 State
   ----------------------------------
   $
   ```

4. Optional: Install virt-manager (graphical user interface) if you are using a desktop computer to manage virtual machines using the following command.

   ```
   $ sudo apt-get install virt-manager
   ```

Configure Bridges

For Static mode, we must configure a Linux ethernet bridge on all NC machines. This bridge connects your local ethernet adapter to the cluster network. NCs will attach

virtual machine instances to this bridge when the instances are booted. To configure a bridge on Ubuntu:

1. Install the bridge-utils package.
2. Modify the following file according to manual:

   ```
   /etc/network/interfaces
   ```

3. Restart Networking

Configure NTP

Eucalyptus requires that each machine have the Network Time Protocol (NTP) daemon started and configured to run automatically on reboot. The following has to be done on CLC and NCs

1. Install NTP on machines

   ```
   $ sudo apt-get install ntp
   ```

2. Edit the following NTP configuration files on machines according to manual:

   ```
   /etc/ntp.conf
   ```

3. Restart the NTP service on machines

2.8.4 Installing Eucalyptus

1. Download the Eucalyptus release key from

   ```
   www.eucalyptus.com/eucalyptus-cloud/security/keys
   ```

2. Add the public key to the list of trusted keys.

   ```
   apt-key add c1240596-eucalyptus-release-key.pub
   ```

3. Edit the file `/etc/apt/sources.list` and add the following content:

   ```
   deb http://downloads.eucalyptus.com/software/
   eucalyptus/3.1/ubuntu precise main

   deb http://downloads.eucalyptus.com/software/
   euca2ools/2.1/ubuntu precise main
   ```

4. Update source list on all machines.

5. Install Eucalyptus packages and dependencies.

   ```
   eucalyptus-cloud
   eucalyptus-cc
   eucalyptus-sc
   eucalyptus-walrus
   ```

6. On each planned NC server, install the NC package

   ```
   eucalyptus-nc
   ```

2.8.5 Configuring Eucalyptus

Configure Network Modes - Static Mode

Static mode requires you to specify the network configuration each VM should receive from the Eucalyptus DHCP server running on the same physical server as the CC component. Configure each CC to use an Ethernet device that lies within the same broadcast domain as all of its NCs. To configure for Static mode:

1. CLC Configuration - No network configuration required.
2. CC Configuration - For each CC, log in and edit the following file according to the manual for static mode.

   ```
   /etc/eucalyptus/eucalyptus.conf
   ```

3. NC Configuration - For each NC, log in and NC machine and edit the following file according to the manual.

   ```
   /etc/eucalyptus/eucalyptus.conf
   ```

2.8.6 Registering Eucalyptus

Eucalyptus implements a secure protocol for registering separate components. You only need to register components the first time Eucalyptus is started after it was installed. Most registration commands run on the CLC server. NCs, however, are registered on each CC. Note that each registration command will attempt an SSH as root to the remote physical host where the registering component is assumed to be running. The registration command also contacts the component so it must be running at the time of the command is issued.

NCs need only two pieces of information: component name and IP address. Other component require four pieces of information:

1. The component (–register-XYZ) you are registering, because this affects where the commands must be executed.
2. The partition (–partition) the component will belong to. The partition is the same thing as availability zone in AWS.
3. The name (–component) ascribed to the component. This name is also used when reporting system state changes which require administrator attention. This name must be globally-unique with respect to other component registrations.
4. The IP address (–host) of the service being registered.

2.8.7 Starting Eucalyptus

1. **Start the CLC Walrus** : In our case since Walrus, CLC, and SC are on the same physical machine. Hence all 3 can be initialized and started by logging into CLC and using the following commands:

   ```
   /usr/sbin/euca_conf --initialize
   service eucalyptus-cloud start
   ```

2. **Start the CC** : Log in to the CC server and enter the following commands:

   ```
   service eucalyptus-cc start
   ```

3. **Start the CC** : Log in to the CC server and enter the following commands:

   ```
   service eucalyptus-nc start
   ```

Verify the Startup according to the manual.

2.8.8 Configuring the Runtime Environment

After Eucalyptus is installed and registered, perform the tasks in this section to configure the runtime environment and generate Administrator Credentials.

1. Generate administrator credentials.

   ```
   /usr/sbin/euca_conf --get-credentials admin.zip
   unzip admin.zip
   ```

2. Source the eucarc file.

   ```
   source eucarc
   ```

2.8.9 Eucalyptus Testbed Setup

We have implemented a Eucalyptus cloud platform on Linux with 2 physical machines. The components directly accessible to the external world are CLC and Walrus and the controlling components (CC/SC) are installed on the first physical machine. The CLC and WALRUS components are not designed for networking over wide area and hence we prefer to install and run these components on the same physical machine i.e. called the front end machine. The node controller (NC) is installed on the second machine of 24 cores. This machine is virtualization enabled and has the capability of virtual machine instantiation on demand.

The cloud installation described here is done using the Eucalyptus 3.1.2 version and euca2tools 2.1.1 with Ubuntu 12.04 LTS 64-bit Desktop operating system on both front-end machine as well as the Node controller machine.

Table 2.1 shows the configuration details of the two machines and the client.

Table 2.1: Machine Roles and Configurations

Machine Name	Machine Role	CPU (Cores)	RAM (GB)	Disk (GB)	eth0	eth1
Cloud Controller	CLC/WALRUS /CC/SC	8	16	500	Private	Public
Node Controller	Host Virtual Machines	24	48	1000	Private	Private
Client	Requests Jobs	4	6	1000	Public	...

2.9 Customized Image Creation

An image is a snapshot of a system's root file system and it provides the basis for instances. When you run a new virtual machine, you choose a machine image to use as a template. The new virtual machine is then an instance of that machine image that contains its own copy of everything in the image. The instance keeps running until you stop or terminate it, or until it fails. If an instance fails, you can launch a new one from the same image. You can create multiple instances of a single machine image. Each instance will be independent of the others. Once the Eucalyptus framework has been installed Eucalyptus Systems (*b*) details the user level functionalities.

In Eucalyptus there are different types of Virtual Machines which can be instan-

tiated, ranging from small size to extra large size VMs depending up on the amount of resources allocated on the node controller machine. 2.2 shows the number of VMs available for each type.

Table 2.2: VM sizes

VM type	Max No.	CPU	RAM (MB)	Disk (GB)
m1.small	24	1	128	2
c1.medium	24	1	256	5
m1.large	12	2	512	10
m1.xlarge	12	2	1024	20
c1.xlarge	6	4	2048	20

You can use a single image or multiple images, depending on your needs. From a single image, you can launch different types of instances. An instance type defines what hardware the instance has, including the amount of memory, disk space, and CPU power.

A machine image contains all the information needed to boot instances of your software. For example, a machine image might contain software to act as an application server or Hadoop node.

An image is a file having the entire state of a guest instance which would run on the Node controller machine. An image may contain any of the Linux distributions like, Ubuntu, Cent-OS, Debian or Fedora. An image which is bundled, uploaded and registered with Eucalyptus is called Eucalyptus machine image (EMI). Other types of images supporting EMI are the kernel (EKI) and ramdisk (ERI). They contain kernel modules for proper functioning of the image.

The following are the tasks that you can perform on images in Eucalyptus:

2.9.1 Add an Image

To enable an image as an executable entity, you do the following:

1. Bundle a root disk image and kernel/ramdisk pair.
2. Upload the bundled image to Walrus bucket storage.
3. Register the data with Eucalyptus.

Add a Kernel/Ramdisk

When you add a kernel/ramdisk to Walrus, you bundle the kernel/ramdisk file, upload the file to a bucket in Walrus that you name, and then register the kernel with Eucalyptus. Use these commands to add a kernel/ramdisk to Walrus (X stands for kernel/ramdisk):

```
$ euca-bundle-image -i <X_file> --X true
$ euca-upload-bundle -b <X_bucket> -m
/tmp/<X_file>.manifest.xml
$ euca-register <X_bucket>/<X_file>.manifest.xml
```

Where the returned value eki-XXXXXXXX is the unique ID of the registered kernel image and eri-XXXXXXXX is the unique ID of the registered ramdisk image.

Add a Root Filesystem

When you add a root filesystem to Walrus, you bundle the root filesystem file, upload the file to a bucket in Walrus that you name, and then register the root filesystem with Eucalyptus. The bundle operation can include a registered ramdisk (ERI ID) and a registered kernel (EKI ID). The resulting image will associate the three images. Use the following commands to add a root filesystem to Walrus:

```
$ euca-bundle-image -i <root_filesystem_file>
 --kernel <eki-XXXXXXXX> --ramdisk <eri-XXXXXXXX>

$ euca-upload-bundle -b <root_filesystem_file_bucket>
 -m /tmp/<root_filesystem_file>.manifest.xml

$ euca-register <root_filesystem_file_bucket>
/<root_filesystem_file>.manifest.xml
```

Where the returned value emi-XXXXXXXX is the unique ID of the registered machine image.

2.9.2 Verify the Image

Use the following command to see the list of available images and verify your image.

```
$euca-describe-images
```

2.9.3 Modify an Image

To modify an existing image to meet your needs:

1. Create a mount point for your image.
2. Associate a loop block device to the image.
3. Mount the image.
4. Make procfs, dev and sysfs available in your chroot environment.
5. If you want to install packages into the image, you can `chroot` into the mounted directory and use `apt-get` to install the packages.
6. Unmount the drive.

You now have an image with your modifications. You are ready to add the image to Eucalyptus. Refer to the example of MATLAB enabled Image creation.

2.10 Using Instances

After an virtual image is launched, the resulting running system is called an instance. An instance is a virtual machine. A virtual machine is essentially an operational private computer that contains an operating system, applications, network accessibility, and disk drives. The following are the instance related tasks.

2.10.1 Find an Image

Enter the following command to display available images:

```
$ euca-describe-images
```

2.10.2 Create Keypairs

Eucalyptus uses cryptographic keypairs to verify access to instances. Before you can run an instance, you must create a keypair using the euca-add-keypair command. Creating a keypair generates two keys: a public key (saved within Eucalyptus) and a corresponding private key (output to the user as a character string). To enable this private key you must save it to a file and set appropriate access permissions (using the chmod command), as shown in the example below.

When you create a VM instance, the public key is then injected into the VM. Later, when attempting to login to the VM instance using SSH, the public key is checked against your private key to verify access.

1. To create a keypair enter the following command which will save the private key to a file in your local directory:

   ```
   $ euca-add-keypair <keypair_name>
    <keypair_name>.private
   ```

2. Change file permissions to enable access to the private key file in the local directory.

3. Query the system to view the public key.

   ```
   $ euca-describe-keypairs
   ```

2.10.3 Authorize Security Groups

Before you can log in to an instance, you must authorize access to that instance. This done by configuring a security group for that instance.

A security group is a set of networking rules applied to instances associated with a group. When you first create an instance, it is assigned to a default security group that denies incoming network traffic from all sources. To allow login and usage of a new instance, you must authorize network access to the default security group with the following command.

```
$ euca-authorize <security_group>
```

2.10.4 Launch an Instance

1. Use the following command to launch an instance.

   ```
   $ euca-run-instances <image_id> -k <mykey>
   ```

2. Enter the following command to get the launch status of the instance:

   ```
   $ euca-describe-instances <instance_id>
   ```

2.10.5 Log in to an Instance

When you create an instance, Eucalyptus assigns the instance two IP addresses: a public IP address and a private IP address. The public IP address provides access to the instance from external network sources; the private IP address provides access to the instance from within the Eucalyptus cloud environment. To use an instance you must log into it via ssh using one of the IP addresses assigned to it.

Use SSH to log into the instance, using your private key and the external IP address.

```
$ ssh -i <keypair.private> root@<IP Address>
```

2.10.6 Reboot an Instance

Rebooting preserves the root filesystem of an instance across restarts. To reboot an instance:

```
$ euca-reboot-instances <instance_id>
```

2.10.7 Terminate an Instance

Terminating an instance can cause the instance and all items associated with the instance (data, packages installed, etc.) to be lost. To terminate VM instances:

```
euca-terminate-instances <ID string>$
```

CHAPTER 3

Scientific Computing on the Cloud - A Distributed Computing Approach

3.1 Definition

Scientific Computing is one of the leading disciplines in information technology with varied application in fields such as economics, science and engineering. It is the practice of aggregating the computing resources in such a way that it delivers much higher performance than computations on a personal desktop or workstation. Due to specific performance requirements, it is common to operate high performance computing resource in private and thus the access to these are often restricted. Also jobs have to wait in a queue for resource allocation and execution. Also it may happen that these physical machines are underutilized because of the fluctuating demands within the particular organization.

3.2 Advantages

Thus HPC or scientific computing on the cloud can be alternate solution to the computing needs of the organization. A cloud computing approach can be both cost effective and efficient alternative to traditional HPC approaches. There are several advantages to using a cloud computing approach:

1. Large providers can set up the required infrastructure bring down the overall running cost and thus computing resources can be made available at lower costs.
2. The use of VMs can allow users to secure administrative privileges and thus customize usage according to their requirements, from choosing operating systems down to the various libraries.
3. The continuous availability and scalability of cloud ensure virtually infinite pool of resources to choose from at any point of time. Thus jobs need not wait in queue for resources.

4. Clouds can provide isolation of multiple workloads on the same physical resources and networking infrastructure.
5. Dynamic provisioning ensures that the computing resources can be scaled up and down according to the users' workload demand fluctuation.
6. Service Level Agreements can guarantee network performances and other Quality of Service (QoS) constraints.
7. Live Migration of VMs is a concept that allows seamless transfer of VMs over physical resources during operations. The advantage is that maintenance work can be run on the physical resources without interrupting the jobs or processes.

3.3 Distributed Computing - Definition and Frameworks

Distributed computing refers to the parallelization of a large computational job into several smaller computational tasks and executing these tasks on different nodes. Nodes are autonomous computational resources with its own local memory that communicate with each other by passing messages on the network. First a MATLAB Distributing Computing Server was considered but was found to be inadequate for the work due to system constraints. Then a Python based approach was considered. For communication purposes the two most prevalent approaches have been MPI and MapReduce. Finally MPI was chosen for the simplicity and flexibility provided.

3.4 MATLAB Distributed Computing Server

This framework can run computationally intensive MATLAB programs and Simulink models on computer clusters, clouds and grids. The program or model is first developed on a multicore desktop environment using the Parallel Computing Toolbox and then scaled up to many computers by running it on this framework. This framework can support batch jobs, parallel communication and distributed large data.

3.4.1 MATLAB enabled Image creation for Eucalyptus

The following section explains how to setup MATLAB enabled EMI.

1. Make a temp directory

```
$ mkdir /newubun
```

2. Mount the image.

```
$  mount -o loop /img_name.img /newubun
```

3. Find out the loop block device attached to the mounted Image and increase the associated space by using the following commands (X is the number of MiB you want to add):

```
$ sudo losetup /dev/loop0
$ sudo dd if=/dev/zero bs=1 MiB of=/path/to/file
conv=notrunc oflag=append count=X
$ sudo losetup -c /dev/loop0
$ sudo resize2fs /dev/loop0
```

4. First change root to `/newubun` and copy MATLAB ISO file to it. Mount the ISO File. Give GUI permissions as follows.

```
$ xhost
$ echo $DISPLAY
$ export DISPLAY=:0
$ xhost local:root
$ export XAUTHORITY=/home/sysid/.Xauthority
```

5. Install MATLAB by following the MATLAB Installation Guide.

6. Unmount MATLAB and exit.

3.4.2 Advantages

1. It has a very large and growing database of functions and built-in algorithms.

2. It is very user friendly, especially for beginners.

3. It already has a large scientific community base; used extensively in several universities.

4. Simulink is a unique product.

3.4.3 Disadvantages

1. The smallest VMs cannot be used to run MATLAB due to disk and ram constraints.

2. All the algorithms are proprietary. Hence we cannot see, verify or modify the code of the algorithm. It also makes it difficult for third parties to extend the tools of MATLAB.

3. It is expensive to get a MATLAB license.

3.5 MapReduce

MapReduce is a programming framework that lets the user process large data sets with a parallel, distributed algorithm on a distributed system. A MapReduce implementation consists of 2 main phases: Map() phase and Reduce() phase.

In the Map() phase, the master node takes the input, divides it into smaller sub-problems, and distributes them to worker nodes. Each mapping operation is independent and thus can be performed in parallel. A worker node may do this again in turn, leading to a multi-level tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

In the Reduce() phase, the master node then collects the answers to all the sub-problems and combines them in some way to form the output which is the answer to the problem it was originally trying to solve. We can have a set of reducers that perform the reduction phase provided that all outputs of the map phase which share the same key are given to the reducers at the same time.

Apache Hadoop is an open source software framework built on the MapReduce framework for storage and large scale distributed data processing. A Hadoop framework has a single master and multiple worker nodes. The master node is responsible for the tracking jobs submitted, sub-tasks created and the data being dealt with. The worker node is also responsible for sub-task tracking and data. It is also possible to have data-only worker nodes and compute-only worker nodes. The Hadoop framework is comprised of the following modules:

Hadoop Common: This module contains all the libraries and utilities needed by other Hadoop modules.

Hadoop Distributed File System (HDFS): This module is a distributed and scalable file-system written in JAVA for the Hadoop framework. Each data node serves

up blocks of data over the network using a protocol specific to HDFS. The file system uses the TCP/IP layer for communication and use remote procedure calls (RPC) to pass messages and communicate between each other.

Hadoop YARN: This module is a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.

Hadoop MapReduce: a programming model for large scale data processing.

3.6 Message Passing Interface

The Message Passing Interface is a standardized and portable message passing system designed to function on a wide variety of parallel systems. The standard itself is not a library, but defines the syntax and semantics of the library routines for a language independent communications protocol. MPI primary addresses the message passing parallel programming model in which data is moved across the address spaces of processes through cooperative operations. Since the release of MPI, it has become the leading standard for message passing libraries for parallel systems and has achieved widespread implementation.

3.6.1 Advantages

1. Standardization: MPI is the only message passing library that can be considered a standard. It is supported on all types of HPC platforms.
2. Portability: The user can seamlessly port the applications across platforms (that are MPI compliant) by making minimal changes to source code.
3. Functionality and Performance Opportunities: There are over 440 routines defined in the latest MPI release. Also vendor specific implementations can exploit hardware features for maximal performance.
4. Availability of rich documentation and support.

3.6.2 Functionality of MPI

MPI interface is meant to provide essential virtual topology, synchronization and communication functionality between a set of processes that have been mapped to nodes/instances. Typically each core in a multicore machine will be assigned a single process. This assignment happens at runtime through the agent that starts the MPI routine.

Functionalities include point-to-point rendezvous type send/receive operations, choosing between Cartesian or graph-like logical process topology, exchanging data between process pairs, combining the partial results of computations (gather and reduce), synchronization of nodes. Point-to-point communication can be of synchronous, asynchronous or buffered type as well as blocking/non-blocking type. Collective communications like broadcast and scatter are also supported. Latest implementations of MPI also support dynamic process management allowing addition of new processes during program execution.

3.6.3 Programming Model of MPI

The MPI was first developed as a communications protocol for distributed systems. As developments were made in architectural trends, shared memory systems were also combined into MPI creating hybrid distributed and shared memory systems. The libraries were adapted to handle both types of memory architectures seamlessly.

In most MPI implementations, a fixed set of processes are created and each process is assigned to a processor. However each process may execute a different program on a different set of data. Hence MPI model conforms to the 'multiple program multiple data' (MPMD) model.

3.6.4 Communicators and Groups

A group is an ordered set of processes. Each process in a group is associated with a unique integer rank. Rank values range from 0 to N-1, where N is the number of processes in the group. A process can belong to multiple groups. Group is a dynamic object in MPI and can be created or destroyed during the execution of the program. For any message being passed, the source and destination are identified by the process rank

of the process within that group.

The communicator is an object that defines the 'communication universe' within the MPI framework. It is a logical unit that defines which processes are allowed to send and receive messages. Intracommunicator is the communicator that is used for communication within a single group of processes. Intercommunicator is the communicator that is used for communication across different non overlapping groups. Communicators are also dynamic, i.e., they can be created or destroyed during the execution of the program. Each communicator gives each contained process an independent identifier and arranges its contained processes in an ordered topology.

MPI was chosen over MapReduce because

1. MapReduce setup and running costs are significant. MPI is much lighter and simpler.
2. MPI supports asynchronous communication.
3. Code refactoring is easier in MPI as compared to MapReduce.
4. Much more efficient algorithms can be written in MPI for certain applications like learning algorithms.

3.7 Python

Python is a widely used general-purpose, high-level programming language. It has an efficient high-level data structures and a simple but effective approach to object-oriented programming with dynamic typing and dynamic binding. It has a holistic language design with emphasis on readability and concise coding. It has the perfect balance of high level and low level programming. It is an open source programming languages with an impressive standard library and external libraries being developed by the enthusiastic Python community. Python has good language interoperability.

Python has an impressive support for scientific computing. SciPy is a computing environment and open source ecosystem of software for Python programming language that is used by scientists, analysts and engineers doing scientific computing and technical computing. SciPy also refers to the open source Python library of algorithms and mathematical tools that are at the crux of the SciPy environment.

NumPy is an extension to the Python programming language that adds support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these. This pair of libraries (SciPy/NumPy) provide array and matrix structures, linear algebra routines, numerical optimization, random number generation, statistics routines, differential equation modeling, Fourier transforms and signal processing, image processing, sparse and masked arrays, spatial computation, and numerous other mathematical routines. Most of the MATLAB's basic functionality and support for reading and writing MATLAB files are included in these.

matplotlib is a plotting library for the Python programming language and its NumPy numerical mathematics extension. SymPy is a Python library for symbolic computation. There is great documentation and a very active community for these libraries.

3.7.1 MPI for Python

There are several Python packages available that wrap MPI as a library and allow MPI functions to be called from Python. These include pyMPI, maroonmpi, mpi4py, myMPI, Pypar etc.

The mpi4py library Dalcin (2012) provides an interface very similar to the MPI-2 standard C++ Interface. It supports point-to-point (sends, receives) and collective (broadcasts, scatters, gathers) communications of Python objects, as well as optimized communications of certain Python object like NumPy arrays. We have chosen mpi4py for our implementation.

Since we will be using Python's scientific programming environment for developing applications, let us briefly go through the important routines in this MPI implementation for *Python*.

Environmental Management Routines

1. `MPI_Init()` : This function initializes the MPI execution environment and must be called only once in every MPI program before any other MPI functions.
2. `MPI_Finalize()` : The function terminates the MPI execution environment and there should be no other MPI routine after this.

These two functions are called when you import the MPI module from mpi4py

package, but only if mpi4py is not initialized.

Communicators

In mpi4py `Comm` is the basic class of communicators. The `Intracomm` and `Intercomm` classes are subclasses of the `Comm` class. `COMM_SELF` and `COMM_WORLD` are the two predefined intracommunicator instances available. From them, new communicators can be created as needed.

The number of processes in a communicator and the calling process rank can be respectively obtained with methods `Get_size()` and `Get_rank()`. The associated process group can be retrieved from a communicator by calling the `Get_group()` method, which returns an instance of the Group class.

Point-to-Point Communication

This is the fundamental type of communication with one side sending and the other side receiving. MPI provides a set of send and receive functions that can be used to communicate specific typed data with a tag associated for identification.

The basic methods for point-to-point communication are `Send()` and `Recv()` for communicator objects. These are of blocking type, i.e., the function will block the caller until the data buffers used in communication can be safely reused by the program. MPI also provides support for non-blocking communication so as to overlap communication with computation.

Collective Communications

Collective Communication allows communication between multiple processes of a group simultaneously. These communicate typed data, but without a specific tag. The following are the common routines:

1. `Barrier()`: It is the function for synchronization operation. Each task, when reaching the `Barrier()` call, blocks until all tasks in the group reach the same `Barrier()` call after which all tasks are free to proceed.
2. `Bcast()`: It is a data movement operation. It broadcasts (sends) a message from the process with rank "root" to all other processes in the group simultaneously.

3. `Scatter()`: It is a data movement operation. It distributes distinct messages from a single source process to other destination processes in the group simultaneously.

4. `Gather()`: It is a data movement operation. It gathers distinct messages from each process in the group to a single destination process. This routine is the reverse operation of `Scatter()`.

5. `Reduce()`: It is a collective computation operation. It applies a reduction operation on all processes in the group and places the result in a single defined process.

Topologies

Virtual Topologies We know that in the basic assignment, each process is assigned a rank in a linear way. This kind of linear ranking method may not reflect the actual logical communication pattern of the processes (which is dependent on the geometry of the problem and the algorithm used).

A virtual topology is the mapping of MPI processes, built upon MPI communicators and groups, into a geometric shape based on a certain connectivity. The two main types of topologies supported by MPI are Cartesian(grid) and Graph. Since they are virtual, there won't be any relationship between the physical structure of the parallel system and the process topology. The topology needs to be programmed into the code by the application developer. MPI provides routines for letting you arrange these processes in a specific grid like structure.

Virtual topologies are useful when the application demands a specific communication pattern. Also the efficiency of the code can be improved in cases where the hardware architecture of the system imposes penalties on communication between specific nodes. Besides possible performance benefits, virtual topology can function as a convenient naming structure greatly benefiting the program readability.

CHAPTER 4

Application Parallelization and Modeling

For the experiments we chose the K-Means Clustering problem, a commonly resource intensive problem. The main motivation was to be able to setup parallelized application for K-Means clustering, so that the user can run this algorithm on their datasets within the time constraints. Let us briefly look at the algorithm.

4.1 K-means Clustering Problem

Clustering is the process of partitioning or grouping a given set of data points into disjoint clusters. This is done such that the certain attributes of data points in the same cluster are alike and the same attributes of data points within different clusters are different. Clustering has been a widely studied problem in variety of domains like neural networks, data mining and statistics, data compression and vector quantization etc. The k-means method of clustering has been proven to be an effective clustering technique for many applications.

4.1.1 Problem Description

Given a set of observations $(x_1, x_2, , x_n)$, where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k sets ($k \leq n$) $S = (S_1, S_2, , S_k)$ so as to minimize the within-cluster sum of squares given as

$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$

where μ_i is the mean of points in S_i.

4.1.2 Sequential Approach

The most common sequential approach uses an iterative refinement technique. The algorithmis given in given in Algorithm 4.1.

Algorithm 4.1 K Means Sequential Algorithm

Require: Data Points $X = (x_1, x_2, \ldots, x_n)$ to be clustered
k (no of clusters)
$MaxIter$ (no of iterations)
Ensure: Set of Clusters $C = (C_1, C_2, \ldots, C_k)$
Cluster Centroids $c = (c_1, c_2, \ldots, c_k)$
$c = RandomCentroid(k)$ {Initialize the centroid list}
while $iter \leq MaxIter$ **do**
  for all $x_i \in X$ and $c_j \in c$ **do**
    $c_p = \arg\min_{c_i} Dist(x_i, c_j)$
    $UpdateCluster(c_p, x_i)$
  for all $C_i \in C$ **do**
    $c_i = RecomputeClusterCentre(C_i)$
    $UpdateCentroidList(c, c_i)$

4.1.3 Distributed Approach

In the distributed approach we divide the initial dataset into m parts where m is the number of the processes over which the code is parallelized. After this we follow a similar iterative refinement technique. Only the centroid list in every iteration is being passed on by the processes. The pseudocode for the Distributed Approach is given in Algorithm 4.2

4.1.4 Experiment

The experiment was done using randomly generated set of data points of sizes varying from 1000 to 13000. Each run of the k- means clustering was distributed over a set of VMs. The number of VMs were varied from 1 to 10 VMs for each data set clustering. It is assumed that no other job is running on the VMs and that there is no waiting time for any job. The response times of each job were measured and plotted. The graphs 4.1 and 4.2 were obtained.

4.1.5 Modeling

Based on the exponentially changing trend observed in 4.2 and the almost linear trend in 4.1, the equation (4.1) has been considered. Here z stands for response time, $x \in (1000, 13000)$ stands for size of the dataset, $y \in \{1, 10\} \subset \mathbb{Z}$ stands for number of

Algorithm 4.2 K Means Distributed Algorithm

Require: Data Points $X = (x_1, x_2, \ldots, x_n)$ to be clustered
k (no of clusters) , $MaxIter$ (no of iterations), p (no of processes)
Ensure: Set of Clusters $C = (C_1, C_2, \ldots, C_k)$
Cluster Centroids $c = (c_1, c_2, \ldots, c_k)$
if $rank = 0$ **then** // Initialization
$c = RandomCentroid(k)$ // Initialize the centroid list
$SplitData(X, p)$ // Split data X into p parts $(X_1, X_2, \ldots, X_p)$
for $i = 0$ to p **do**
$Send(X_i, rank = i, tag = i)$
else
$Receive(X_i, rank = 0, tag = i)$
while $iter \leq MaxIter$ **do** // Iterations
if $rank = 0$ **then**
$Broadcast(c)$ // Broadcast Centroid List
for $i = 0$ to p **do**
$Receive(l_i, rank = i)$ // Receive Cluster Assignments from each process
$L = Aggregate(l_i)$
$C = GetClusters(X, L)$
for all $C_i \in C$ **do**
$c_i = RecomputeClusterCentre(C_i)$
$UpdateCentroidList(c, c_i)$
else
$i = rank$
$Receive(c, rank = 0)$ // Receive Centroid List
$l_i = GetClusterAssignment(X_i, c)$ // Cluster assignments computed
$Send(l_i, rank = 0)$

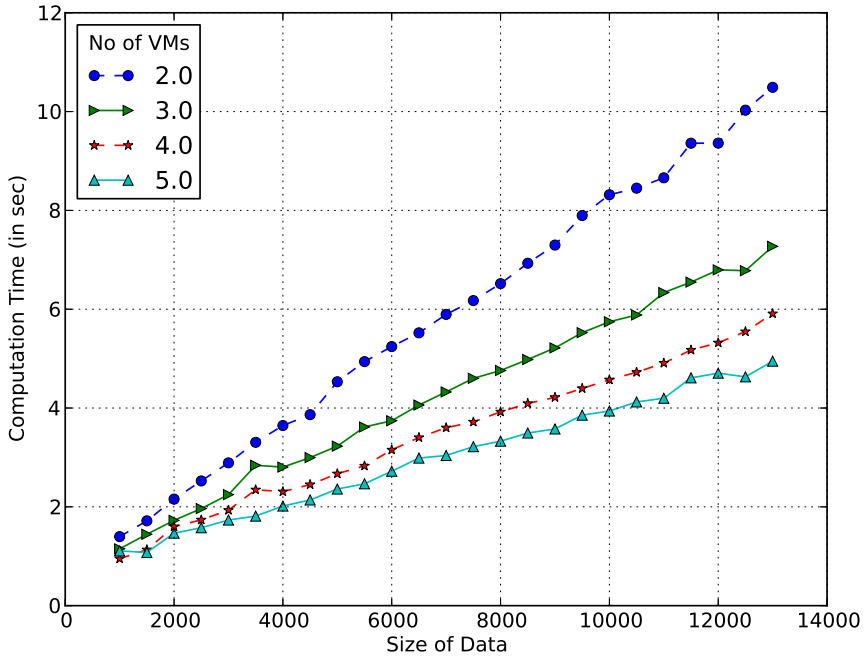

Figure 4.1: Response Times vs Size of Dataset for different no of VMs

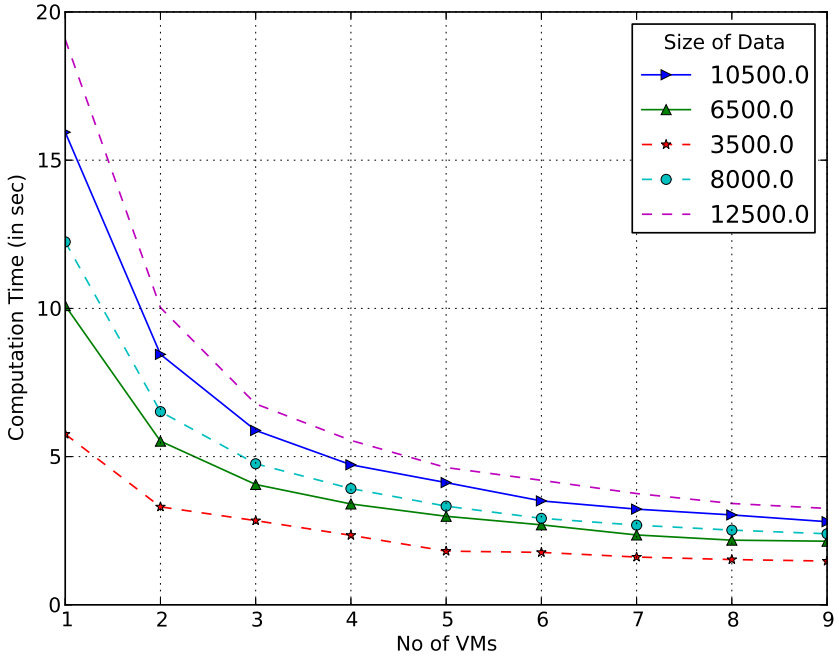

Figure 4.2: Response Times vs VMs for different sizes of Dataset

virtual machines/processors.

$$z = ke^{(a_1 y + a_2 y^2 + \dots)} x^b \tag{4.1}$$

Taking logarithm on both sides

$$z = 0.029e^{(-0.77y + 0.10y^2 - 0.004y^3)} x^{0.74} \tag{4.2}$$

If we write this equation for each data point (total m data points) the equations can be combined together and written in the following matrix form:

$$\begin{bmatrix} \ln(z_1) \\ \ln(z_2) \\ \vdots \\ ln(z_m) \end{bmatrix} = \begin{bmatrix} 1 & y_1 & y_1^2 & \dots & \ln x_1 \\ 1 & y_2 & y_2^2 & \dots & lnx_2 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & y_m & y_m^2 & \dots & lnx_m \end{bmatrix} \begin{bmatrix} \ln(k) \\ a_1 \\ a_2 \\ \vdots \\ b \end{bmatrix} \tag{4.3}$$

It can be simplified as $Y = AX$. Now we find $X = \hat{X}$ such that it minimizes $\|r\|$ where $r = A\hat{X} - Y$. Then $\hat{X}$ is called the least squares fit of $Y = AX$. We assuming that A is full rank. The norm of r is given as

$$\|r\|^2 = X^T A^T AX - 2Y^T AX + Y^T Y \tag{4.4}$$

Setting the gradient to zero to obtain the minimum value.

$$\nabla_X \|r\|^2 = 2A^T AX - 2A^T Y = 0 \tag{4.5}$$

This yields the normal equations : $A^T AX = A^T Y$ Assuming $A^T A$ is invertible gives us the solution to the least squares fit problem as:

$$\hat{X} = (A^T A)^{-1} A^T Y \tag{4.6}$$

These curve fits were done for various number of coefficients $(a_1, a_2, \dots)$ and b. The Percentage Fit and the Root Mean Square Error were computed in each case and

the following graphs were obtained.

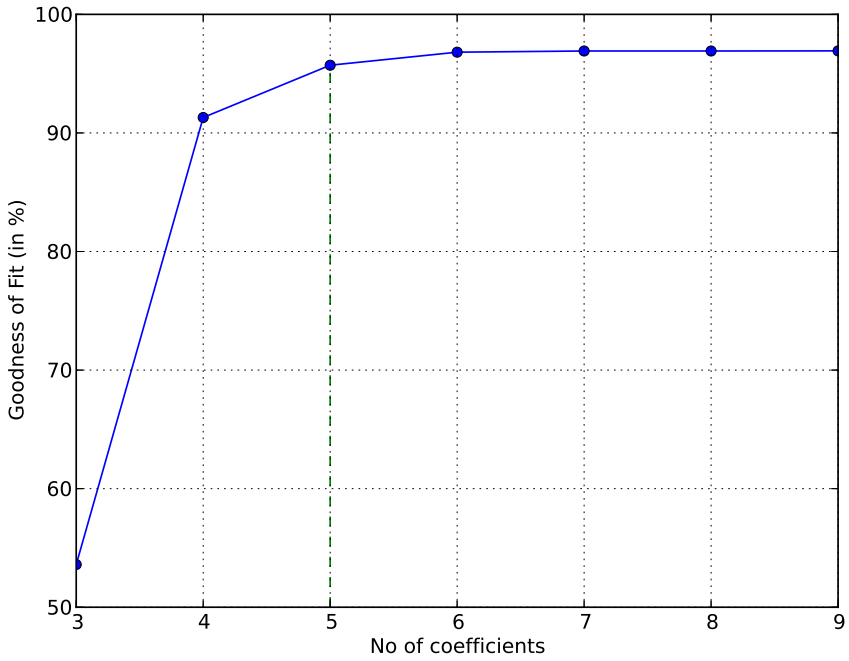

Figure 4.3: Percentage fit vs no. of coefficients

Form the graphs it is evident that a 5 coefficient model is a good fit for the model. Hence the model and the parameters are given as:

$$z = 0.029e^{(-0.77y+0.10y^2-0.004y^3)}x^{0.74} \tag{4.7}$$

- RMSE = 0.645
- Percentage fit = 95.71 %

The 4.5 compares the real values and the estimated values by the model.

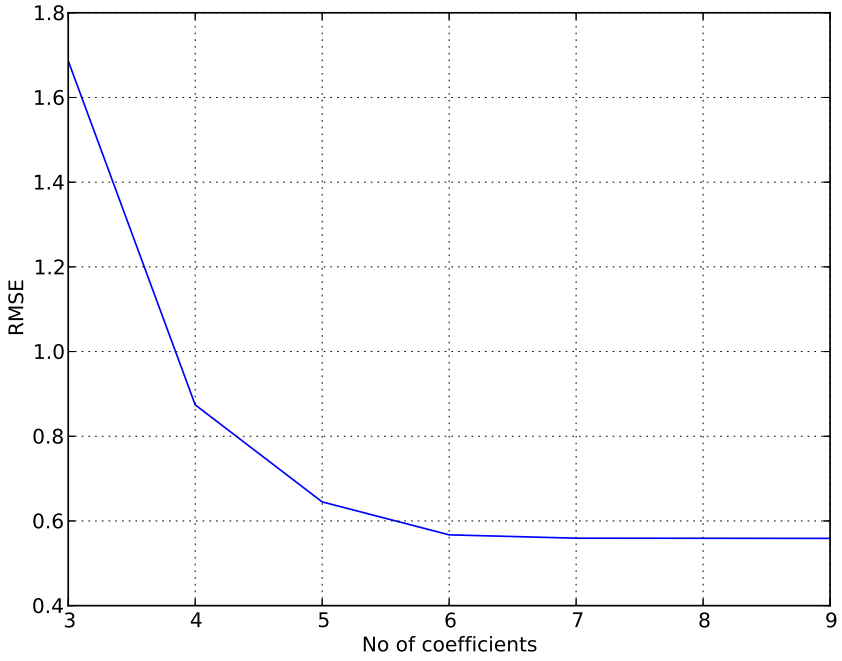

Figure 4.4: RMSE vs no. of coefficients

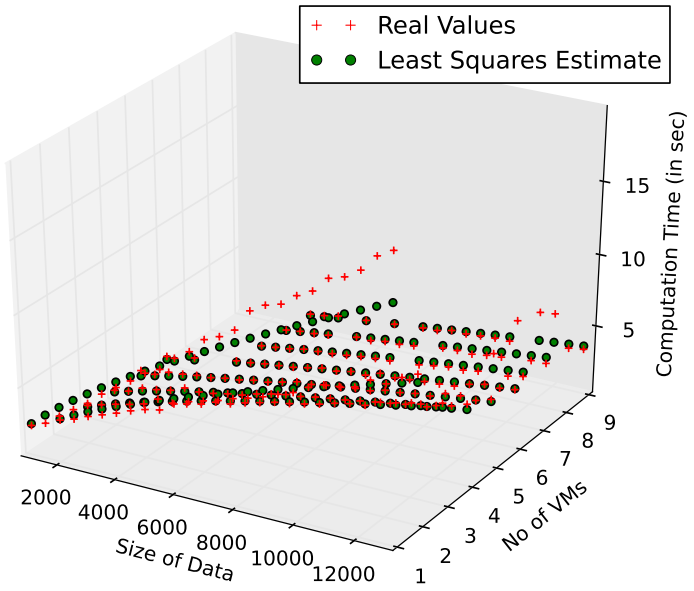

Figure 4.5: Real Values vs Estimated Model

CHAPTER 5

Virtual Machine Allocation - Delay Optimization and Algorithm

5.1 Optimization Problem Formulation

In our analysis we consider the cloud model wherein parallelizable applications like the one described in the previous chapter arrive at the loadbalancer. The loadbalancer creates batches of several jobs and allocates the resources at regular intervals. At every resource allocation trigger, the loadbalancer computes the number of resources to be provided based upon the algorithm proposed.

Consider the batches as a function of time as $Batch(t)$. These are considered at every resource allocation trigger to be composed of several jobs $(A_1(t), A_2(t), \ldots, A_m(t))$. Each of these jobs specify the following parameters:

1. Job Size $(size)$: This is the parameter that decides the size of the job. Input data in the case of K-Means Clustering.
2. Due Time (d) : This is a parameter that specifies the due time expected by the user. It is decide based on the priority of the job assigned by the user.

Let $(x_1, x_2, \ldots, x_m)$ refer to the number of VMs to be assigned for each of the jobs. Then the expected worst case running time of each of the jobs can be computed according to the models described in the previous section. Let us denote the expected worst case run time for job A_i with size s_i and due time d_i run on j VMs be given as $t_i(j) = F(s_i, j)$. Our aim is to minimize the total delay over the expected due time experienced by the user. Hence the the cost function to be minimized is given as $\sum_{i=1}^{m} \delta_{ij}$ where δ_{ij} represents the excess delay over the expected due time when run on j resources and is given as

$$\delta_{ij} = \begin{cases} t_i(j) - d_i & : t_i(j) > d_i \\ 0 & : t_i(j) < d_i \end{cases}$$

At every resource allocation trigger the loadbalancer runs a check on the current utilization of the system and returns the number of free resources (VMs) that can be allocated. Let us assume that at time t, a maximum of n resources are available. Hence our optimization problem can be written as

$$\min \sum_{i=1}^{m} \delta_{ij}$$

subject to the constraint

$$\sum_{i=1}^{m} x_i \leq n$$

$$x_i \in \mathbf{Z}$$

5.2 Proposed Algorithm

The jobs arriving are grouped into batches at regular trigger intervals as showin in 5.1. At each trigger instant, we have a batch of m jobs and n resources need to allocated to minimize the overall $\sum_{i=1}^{m} \delta_{ij}$. We create a matrix C where the rows corresponds to each job and the columns correspond to the number of VMs allocated. The proposed algorithm first computes the total number of resources required for the jobs n_{init} for minimizing the delay without imposing the constraint on total number of available resources. Then it iterates from n_{init} to the actual value n. In each iteration it looks for the job that will add the least amount of penalty into the cost function $\sum_{i=1}^{m} \delta_{ij}$ and reduces 1 VM from it. The detailed algorithm is given in 5.1

5.3 Simulation and Results

As proof of concept, we have simulated the proposed algorithm and compared it with a simple shared allocation algorithm. In the simple shared allocation alogrithm, we divide the total resources available equally among the jobs in the batch. We simulate the two algorithms using the multiprocessing package from python. This package lets us spawn multiple processes that can work on a queue structure. We create 3 processes for simulating each of the algorithm. The first process generates batches of jobs at regular

Algorithm 5.1 Resource Allocation Algorithm

Require: Model equations, n number of VMs, Job numbers and details
Ensure: x_i number of virtual machines to be allocated for each job i

for $i = 1$ to m **do**
  Compute j_i such that $t_{ij} = d_i$
  $J_i = \lceil j_i \rceil$
  $\delta_{ij} = t_{iJ_i} - d_i$
if $\sum(J_i) \leq n$ **then**
  Allocate J_i VMs to job i
else
  $k = 0$
  for $i = 1$ to m **do**
    if $J_i = 1$ **then**
      Allocate J_i = 1 VM to job i
      $extra\_delay_i = 0$
      $k = k + 1$
    else
      $extra\_delay_i = \delta_{iJ_i-1} - \delta_{iJ_i}$
  $m = m - k$
  $n = n - k$
  $iter = \sum(J_i) - n$
  while $iter > 0$ **do**
    for $J_I > 1$ **do**
      $I = \arg\min_i extra\_delay_i$
    $J_I = J_I - 1$
    if $J_I > 1$ **then**
      $extra\_delay_I = \delta_{iJ_I-1} - \delta_{iJ_I}$
    else
      $extra\_delay_I = 0$
    $iter = iter - 1$
  Allocate J_i VMs to job I
  Total Excess Delay is $\sum_{i=1}^{m} x_i$ where $x_i = \delta_{iJ_i}$

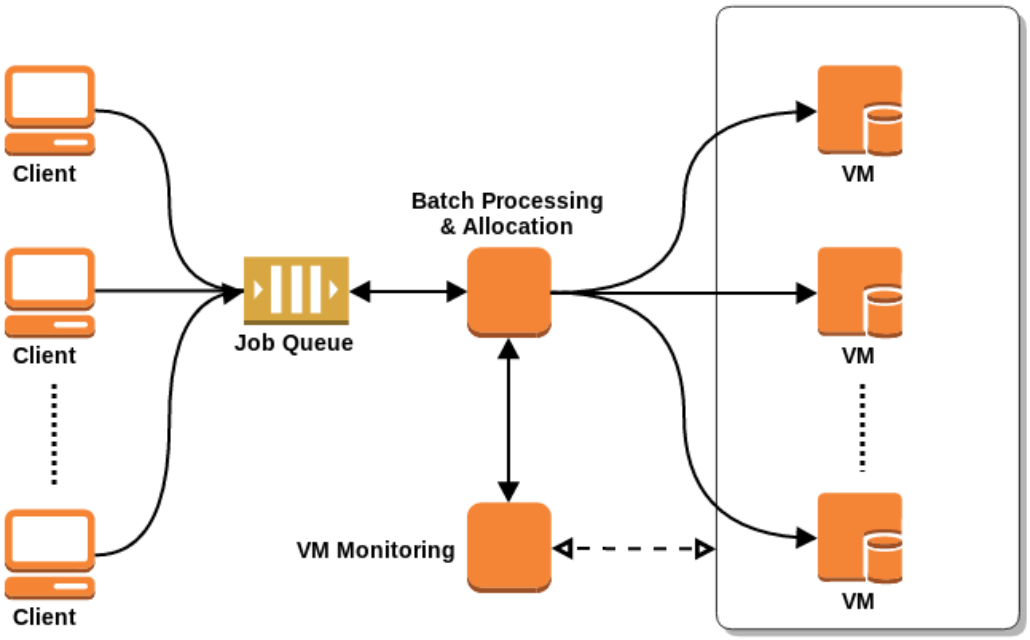

Figure 5.1: Job Queueing and Allocation

interval and pushes it into the queue. The second process pops these batches from the queue, executes the algorithms, starts the jobs on the simulated VMs and computes the delay. The third process monitors and updates the execution of the jobs on the simulated VMs with time.

The jobs were sent in the form of 20 batches at regular intervals of 5 seconds. The no of virtual machines initially alloted were 5. They were scaled up to 10 whenever the no of jobs were more than available free VMs.

We can compare the the total delays experienced by batches of jobs from the graph 5.2. It is evident that the proposed minimum delay algorithm is superior to a simple shared algorithm. Also a comparison of the the VMs utilization 5.3 reveals that the proposed algorithm completes the jobs faster than the shared algorithm.

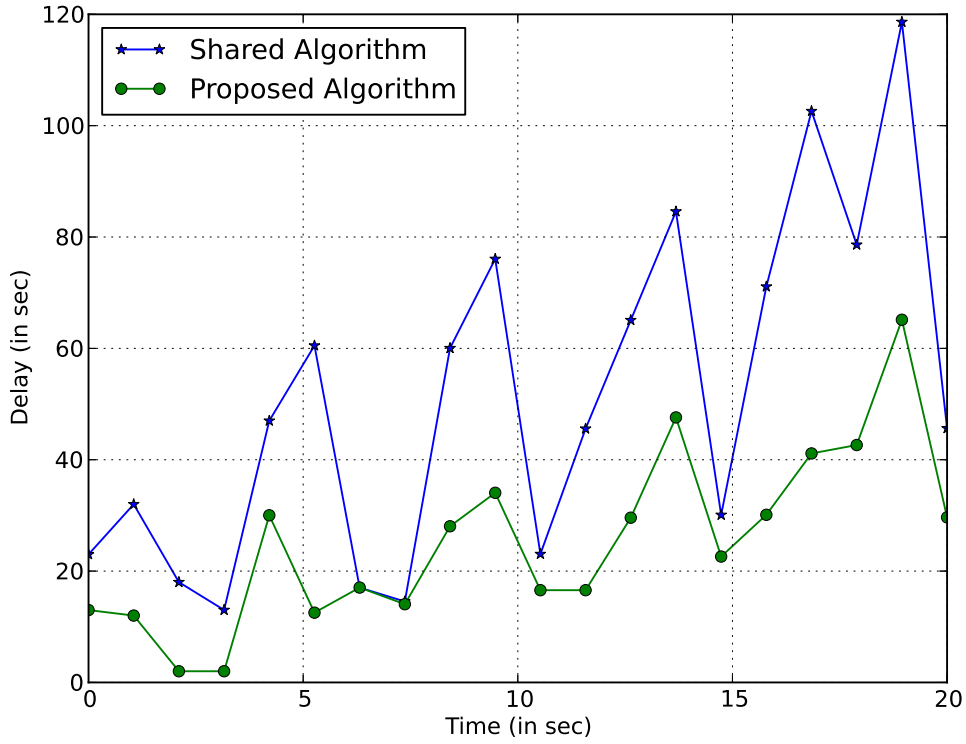

Figure 5.2: Comparison of Delay

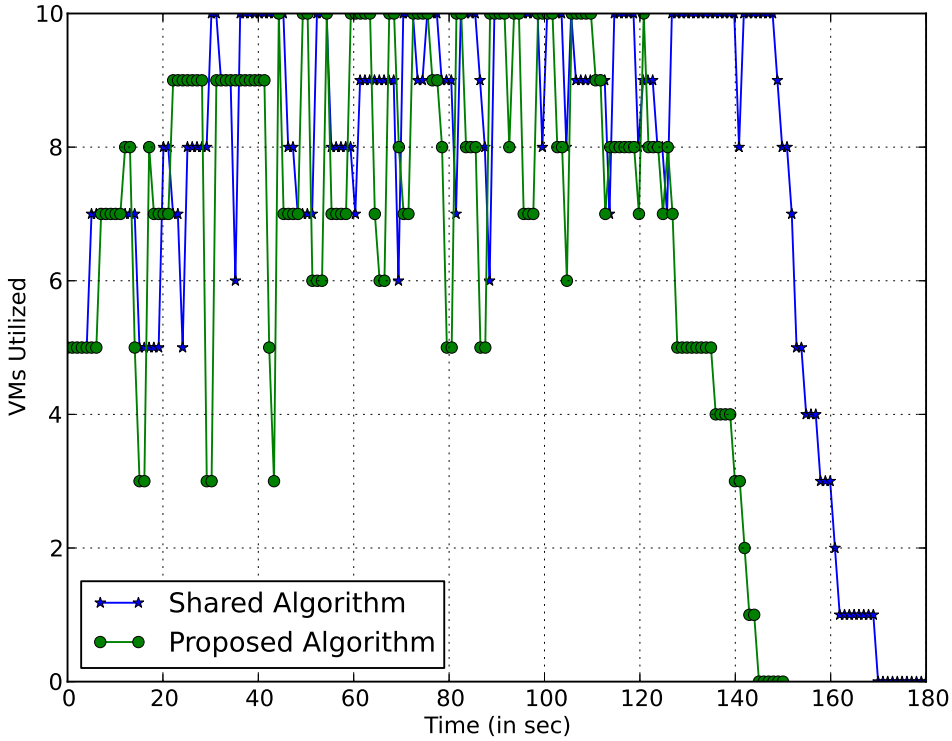

Figure 5.3: Comparison of VM utilization

CHAPTER 6

Conclusion

In this work we have successfully set up the Eucalyptus Cloud framework on the local physical systems to function as a Private Cloud. Using the Message Passing Interface library, we were able to set up parallel applications of KMeans Clustering in the distributed environment to reduce time. This application can thus be deployed over the cloud for use by the users in the scientific community who would like to take advantage of the benefits offered by clouds. Thus the user can get results without bothering about the underlying resource management or implementation. The profile of the response times for the application has been modeled as a function of the job sizes and the resources allocated. Based on these benchmarking tests, a new resource allocation algorithm has been proposed to reduce overall delay over expected due time for jobs in a batch-wise fashion. The proposed algorithm has been simulated and shown to be superior to a simple sharing algorithm in terms of delay minimization.

REFERENCES

1. **Buyya, R.**, **C. S. Yeo**, **S. Venugopal**, **J. Broberg**, and **I. Brandic** (2009). Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, **25**(6), 599 – 616. ISSN 0167-739X. URL `http://www.sciencedirect.com/science/article/pii/S0167739X08001957`.

2. **Dalcin, L.** (2012). *MPI for Python, Release 1.3*. URL `http://mpi4py.scipy.org//`.

3. **Eucalyptus Systems, I.** (*a*). *Eucalyptus 3.1.2 Installation Guide*. URL `https://www.eucalyptus.com/`.

4. **Eucalyptus Systems, I.** (*b*). *Eucalyptus 3.1.2 User Guide*. URL `https://www.eucalyptus.com/`.

5. **Gibson, J.**, **R. Rondeau**, **D. Eveleigh**, and **Q. Tan**, Benefits and challenges of three cloud computing service models. *In Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*. 2012.

6. **Gong, C.**, **J. Liu**, **Q. Zhang**, **H. Chen**, and **Z. Gong**, The characteristics of cloud computing. *In Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*. 2010. ISSN 1530-2016.

7. **He, H.**, Applications deployment on the saas platform. *In Pervasive Computing and Applications (ICPCA), 2010 5th International Conference on*. 2010.

8. **He, Q.**, **S. Zhou**, **B. Kobler**, **D. Duffy**, and **T. McGlynn**, Case study for running hpc applications in public clouds. *In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-942-8. URL `http://doi.acm.org/10.1145/1851476.1851535`.

9. **Jakovits, P.** and **S. Srirama**, Adapting scientific applications to cloud by using distributed computing frameworks. *In Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. 2013.

10. **Kaur, P.** and **I. Chana**, Unfolding the distributed computing paradigms. *In Advances in Computer Engineering (ACE), 2010 International Conference on*. 2010.

11. **Ostermann, S.**, **R. Prodan**, and **T. Fahringer**, Extending grids with cloud resource management for scientific computing. *In Grid Computing, 2009 10th IEEE/ACM International Conference on*. 2009.

12. **Yang, G.**, **Z. Zhu**, and **F. Zhuo**, The application of saas-based cloud computing in the university research and teaching platform. *In Intelligence Science and Information Engineering (ISIE), 2011 International Conference on*. 2011.