GEOSPATIAL SERVICES IN CLOUD INFRASTRUCTURE

Soumadip Biswas

Geospatial Services in Cloud Infrastructure

Thesis submitted in partial fulfillment
for the award of the degree of

Master of Technology(M.Tech)

In

Information Technology

By

Soumadip Biswas, Roll No - 10IT60R12

Under the guidance of

Dr. Soumya K. Ghosh

SCHOOL OF INFORMATION TECHNOLOGY

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721 302, INDIA

April 2012

Certificate

This is to certify that the thesis entitled "**Geospatial Services in Cloud Infrastructure**" submitted by **Soumadip Biswas** (Roll No - 10IT60R12) to the School of Information Technology, Indian Institute of Technology, Kharagpur in partial fulfillment for the award of degree of Master of Technology (Information Technology), is a record of bona fide research work carried out by him under my supervision during the period 2011 - 2012 and I consider it worthy of consideration for the award of the degree of Master of Technology (Information Technology) of the Institute.

Date:

Dr. Soumya K. Ghosh
Associate Professor,
School of Information Technology,
Indian Institute of Technology,
Kharagpur -721 302, India

DECLARATION

I certify that

I, Sri. **Soumadip Biswas**, Roll no **10IT60R12** registered as a student of **M.Tech.** in the **School of Information Technology, Indian Institute of Technology, Kharagpur**, India (hereinafter referred to as the 'Institute')do hereby submit my thesis, title: ***Geospatial Services in Cloud Infrastructure*** (hereinafter referred to as 'my thesis') in a printed as well as in an electronic version for holding in the library of record of the Institute.
I hereby declare that:

1. The electronic version of my thesis submitted herewith on CDROM is in PDF format.

2. My thesis is my original work of which the copyright vest in me and my thesis does not infringe or violate the rights of anyone else.

3. The contents of the electronic version of my thesis submitted herewith are the same as that submitted as final hard copy of my thesis after my viva voce and adjudication of my thesis on 02-05-2012.

4. I agree to abide by the terms and conditions of the Institute Policy on Intellectual Property (hereinafter Policy) currently in effect, as approved by the competent authority of the Institute.

5. I agree to allow the Institute to make available the abstract of my thesis in both hard copy (printed) and electronic form.

6. For the Institute's own, non-commercial, academic use I grant to the Institute the non-exclusive license to make limited copies of my thesis in whole or in part and to loan such copies at the Institute's discretion to academic persons and bodies approved of from time to time by the Institute for non-commercial academic use. All usage under this clause will be governed by the relevant fair use provisions in the Policy and by the Indian Copyright Act in force at the time of submission of the thesis.

7. Furthermore (*strike out whichever is not applicable*)

   (a) I agree / do not agree to allow the Institute to place such copies of the electronic version of my thesis on the private Intranet maintained by the Institute for its own academic community.

   (b) I agree / do not agree to allow the Institute to publish such copies of the electronic version of my thesis on a public access website of the Internet should it so desire.

8. That in keeping with the said Policy of the Institute I agree to assign to the Institute (or its Designee/s) according to the following categories all rights in inventions, discoveries or rights of patent and/or similar property rights derived from my thesis where my thesis has been completed (tick whichever relevant):

   (a) with use of Institute-supported resources as defined by the Policy and revisions thereof,

   (b) with support, in part or whole, from a sponsored project or program, vide clause 6(m) of the Policy.

   I further recognize that:

   (c) All rights in intellectual property described in my thesis where my work does not qualify under sub-clauses 8(a) and/or 8(b) remain with me.

9. The Institute will evaluate my thesis under clause 6(b1) of the Policy. If intellectual property described in my thesis qualifies under clause 6(b1) (ii) as Institute-owned intellectual property, the Institute will proceed for commercialization of the property under clause 6(b4)of the Policy. I agree to maintain confidentiality as per clause 6(b4) of the Policy.

10. If the Institute does not wish to file a patent based on my thesis, and it is my opinion that my thesis describes patentable intellectual property to which I wish to restrict access, I agree to notify the Institute to that effect. In such a case no part of my thesis may be disclosed by the Institute to any person(s) without my written authorization for one year after the date of submission of the thesis or the period necessary for sealing the patent, whichever is earlier.

Soumadip Biswas	**Prof. Soummya K. Ghosh**
(Name of the Student)	(Name of Supervisor)

Department: **School of Information Technology**

Signature of the Head of the Department

Acknowledgments

I feel privileged to acknowledge my sincere gratitude to my adviser, Dr S. K. Ghosh for his invaluable guidance, constant encouragement, time to time help and kind inspiration during the course of this work.

I am grateful to all the faculty members of the School of Information Technology for their continuous support and valuable suggestions. I thank Prof. Indranil Sen Gupta and Prof. Jayanta Mukhopadhyay, the former and present Head, School of Information Technology.

Most importantly I would like to thank my family members, without their support nothing would have been possible. I am grateful for love and support of my father, mother and sister who have always been there for me.

I also express special thanks especially to my senior Arindam Dasgupta(MS). Also I thank all my classmates especially Shrikant Kumar Arya, Vaibhav Neharkar and Darshan Trivedi for their support without any hesitation during the course of my work.

Last of all I am thankful to LaTeX with which this document is edited and formatted.

Soumadip Biswas
School of Information Technology,
Indian Institute Technology, Kharagpur

Abstract

Cloud computing is rapidly emerging as a technology trend which any industry that provides or consumes software, hardware, and infrastructure can leverage. The architecture and technology that cloud service and deployment models offer are main areas of research and development for Geographic Information System (GIS). The rise of cloud computing just might be the most important thing happening in the industry of information technology these days. While not everything will move into the cloud, it's fair to say that nearly every industry will use this new approach in some way or another and so is Geographical Information System. It is unnecessary to mention that GIS technology is one of the most technological advances that human has achieved and lives of the people on the world is effected significantly by the same. Given this reality, cloud computing has the potential to change many aspects of our world as well as for users of geographic information system (GIS). In this Thesis work we will explore our work of creating a *GIS-in-the-cloud* environment by incorporating GIS and Cloud together in a meaningful way. This thesis work gives a model about how to collaborate basic service models of cloud to represent the business architecture completely in the Cloud. In this work *Enterprise–GIS* has been used as the Geographical Information System and an Open Source Cloud system namely Eucalyptus is used as the Cloud structure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing and Geographical Information System are the two keywords for this thesis. It doesn't require mentioning that former one of these two is the most rapidly emerging technology and the other is one of the most important technological advancing that we have achieved. Geographical Information System gives a whole new meaning for the computing world in terms of huge data management and processing the same, on the other hand Cloud Computing is a new door to high performance computing in very convenient way. In this thesis we will see what will be the outcome when these two high-end technological trends merge in a meaningful way. In this chapter we will start the discussion by introducing these two technologies first and then we will see the motivation for this idea to be initiated, following there will be the Objective set for this work.

1.1 Cloud Computing

National Institute of Standards and Technology(NIST) [1] has posted a working definition of cloud computing: Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management of service provider interaction. This cloud model promotes availability and is composed of five essential characteristics (Rapid Elasticity, Measured Service, On-Demand Self-Service, Ubiquitous Network Access, Location-Independent Resource Pooling), three delivery models (Software as a Service, Platform as a Service, and Infrastructure as a Service), and four deployment models (Public Cloud, Private Cloud, Community Cloud and Hybrid Cloud). As per NIST(USA) Cloud Computing is a system which enable on-demand network access to a shared pool of configurable IT resources, which may include networks, storage, applications and services. These resources are managed in such a way so that it can be rapidly provisioned and released with minimum effort of service providers or the system managers. These types of systems now gaining a high popularity in software researcher community because it is new and offers numerous challenges to programmer. However problem

with these systems are most of them are private and not available for research community. Above all as the name suggests for cloud computing, it has no standardized shapes or size similar as a cloud, so we have to define cloud as ourselves as per the requirements of the organizations. In our case we define cloud as an integration of Service Oriented architecture and Virtualization [2, 3, 4, 5].

Conceptually, in Cloud Computing everything is assumed as a service (XaaS) [6], such as TaaS (Testing as a Service), SaaS (Software as a Service), PaaS (Platform as a Service), HaaS (Hard-ware as a Service). To this end, a large number of cloud service providers and middleware suits have emerged, each providing different Cloud Computing services. These providers include Amazon EC2, Google App Engine (GAE), SalesForce.com (SFDC), Microsoft Azure, IBM Blue Cloud, 3Tera, to name a few. As we will discuss throughout the paper, the major challenge of the current Cloud Computing evolution, with multiple offerings and providers, is the lack of standardized APIs and usage model.

1.2 Geographical Information System

Geographical Information Systems (GIS) associates with capturing, storing, analyzing, managing and presentation of geographical features which are spatially referenced to the earth surface [7]. The main objective is to extract information from geospatial data and resolve spatial queries. Geospatial data refers to the abstraction of geographical locations on the earth surface. It may be either natural, such as *road*, *river*, *land-use* etc. or constructed such as *political-boundary*, *census* etc. Geographical information is very useful for effective planning and decision making. As the technology for geographical data collection and storing are advancing, the use of geographical information is increasing within large number of users from various domains.

Geospatial data are collected and maintained by different public and private agencies. These agencies use geospatial data to accomplice their own purposes and also they store these data according to their data storing scheme. With the advent of computer technology, these agencies have been able to digitize the geographical data and store these data with the help of spatial database systems. However, different agencies use several types of proprietary database systems to store and maintain their geospatial data. Moreover, each agency maintains their spatial database according to their own schema and data format along with different data storing policy. Since the geospatial data are collected by many agencies from different areas, the volume geospatial data are becoming too large in volume, highly heterogeneous and scattered over the diverse locations.

Moreover, in most of these situations, the geospatial information are needed to access from the remote location. This problem can be overcome by providing the geospatial information to the mobile device of the users. Therefore, there is a need to deliver geographical information into the mobile devices by extracting and processing of heterogeneous geospatial data.In recent time, with increase in computing power of mobile devices and easy availability

mobile network, there is an increasing demand for geospatial information in mobile environment. Thus the concept of *Mobile GIS* is emerging which may allow the end-users to resolve geospatial query and visualize maps in resource constrained mobile devices. This has given rise to a new GIS paradigm known as ubiquitous GIS computing, which provides geospatial computing services anywhere and anytime [8, 9].

The geospatial data are usually large in volume and require considerable computing and processing power for resolving spatial queries. However, several issues such as restricted computing resources, inadequate amount of memory and limited processing power, limited display area etc restrict process and store huge amount of geospatial data locally within mobile devices. Further, the bandwidth of mobile network are also limited with respect to conventional network. Thus the geospatial framework and applications/functionalities need to be properly designed for mobile environment.

1.3 Motivation

Geographical Information System is a huge data intensive system. For large geographic area a little information gaining can cause enormous amount of data storage. Another very important issue for a GIS system is that as the data is huge in volume, every time a request comes for processing a service demands significant amount of computation. Clearly GIS is a data as well as computation intensive system. For a popular and important Geospatial applications it can easily cause server overload due to computation or storage. Of course one can have enough memory and computational power beforehand in store which simply cause quite amount of capital investment beforehand and at the same time the statistical analysis over a long time in such a system can show that the use of the available resources is not efficient as the resource consumption in concerned. So it is always preferable that if the computational power can be supplied as per requirement i.e. as a service that can dynamically scale up the storage as required(scaling down for storage in this particular case is merely significant) and also if multiple server can be there to balance the load of the GIS only when it is at its Peak-Usage. Clearly from this discussion two things are clear that a GIS system needs to have (1) Dynamically scale Up/Down of storage and (2) replicating server to balance load (or increase computational power overall). As far as the requirement is concerned these two requirements are the area of specialization of the Cloud Computing Technology. So if we can submerge these two i.e. Cloud and GIS in a meaningful way it is completely convenient to have one dedicated Geospatial Cloud [10].

1.4 Objectives

Having an eye on the services of Geographic Information System to be deployed from the cloud let us first get some idea of the background. To provide these services in the cloud infrastructure it first required that there is indeed an available cloud infrastructure which is

able to provide the computing resources or the infrastructure. Here at this level the Geospatial data services can run and it may be sourced from different organizations (heterogeneous). Next is to provide a platform for deploying application(s), for which purpose any Platform as a Service provider needs to be selected, over which the application can be deployed and to be specific it can be used as a generic platform for developing application in the future which can be successfully deployed on the same.

So to summarize the Objectives are the following:

1. Setting up a Cloud Infrastructure, which is capable of providing Infrastructures(as a Service).
2. Proper modeling of the deployment scenario.
3. Deploying Enterprise GIS framework [9] accordingly over the Infrastructure.
4. Setting up a Platform(as a Service) to provide Infrastructure level independence and custom application deployment.
5. Performance Study.

More about suggested models, the accomplished work and performance are discussed in Chapter 3, Chapter 4, Chapter 5 and Chapter 6.

1.5 Thesis Organization

The findings and achievements of this research work have been organized in to the following chapters. This chapter discusses about the basic technologies. The motivation and objectives of the thesis work have also been presented. The rest of the thesis is organized as follows.

Chapter 2: Background Study

This chapter gives necessary background idea in brief for the understanding the work done which is actually described in the following four chapters.

Chapter 3: Integration of Standard Service Models of Cloud

This chapter consists of the proposed model, problem modeling and their descriptions.

Chapter 4: Cloud Setup and Integrated Model Realization

In this chapter the descriptions of system set up has been described, which gives some idea of the used background structure.

Chapter 5: Geographical Information System on Eucalyptus

This chapter is the heart of the work, the realization of the Geospatial Cloud and an example scenario for the same.

Chapter 6: Performance Evaluation

In this chapter we will see the performance evaluation of the systems used in terms of graphs.

Chapter 7: Conclusion & Future Work

This chapter basically summarize the whole work done in this thesis and then talks about new possibilities that can be achieved based on this work.

Chapter 2

Background Study

This chapter is as the name suggests is a description of available works and technology. First section of this chapter is a little literature survey based on couple of papers. The following section talks about available Open Source Clouds and and their components to give an overview of the same. Next section gives a brief idea about the most popular public cloud Amazon and the last section of this chapter is a little about the Distributed Geospatial Services which are specifically are the lifelines of the Geographical Information System.

2.1 Related Work

To test the utilization of cloud computing for Geosciences applications, the GEOSS clearinghouse was deployed, maintained and tested on the Amazon Elastic Cloud Computing (EC2) platform. The GEOSS Clearinghouse is a web based Geographic Metadata Catalog System, which manages millions of the metadata of the spatially referenced resources for the Global Earth Observations (GEO). Their experiment reveals that the EC2 cloud computing platform facilitates geospatial applications in the aspects of a) scalability, b) reliability, and c) reducing duplicated efforts among Geosciences communities. The test of massive data inquiry by concurrent user requests proves that different applications should be justified and optimized when deploying onto the EC2 platform for a better balance of cost and performance.[11]

A study has taken some first steps towards evaluating the suitability of GAE for hosting GIS services by implementing a Web Map Service in the GAE environment.[12] GAE has limitations for free usage such as No files may be written to the servers local filesystem. All data must be written to Googles persistent store, or to the memory cache (memcache), both of which are distributed over many servers to provide high capacity., many important classes in the standard Java Development Kit are not available, including those in the javax.imageio package and most of the contents of java.awt. These packages contain much of the code that would usually be used for image manipulation, Applications may not spawn background threads. This means that the present versions of many Java GIS libraries, including Geotoolkit (http://www.geotoolkit.org), cannot be run,The amount of random-access

memory available to each service instance is low. Precise figures are not available, but tests suggest that the current limit is around 100MB, applications must therefore make very sparing use of RAM. In spite of the limitations of GAE it was somehow possible to deploy GIS java application over GAE.

2.2 Study of Open Source Cloud

There are quite a few number of open source clouds are available. All of them have their own way of deploying their cloud services. Among them we have selected few popular cloud systems and studies their architectures. In this section the discussions are mainly the case study of some of them e.g. Eucalyptus, OpenStack and OpenNebula.

2.2.1 Eucalyptus

Eucalyptus is a software platform for the implementation of private cloud computing on computer clusters. There is an open-core enterprise edition and an open-source edition. Currently, it exports a user-facing interface that is compatible with the Amazon EC2 and S3 services but the platform is modularized so that it can support a set of different interfaces simultaneously. The development of Eucalyptus software is sponsored by Eucalyptus Systems, a venture-backed start-up.[2] Eucalyptus works with most currently available Linux distributions including Ubuntu, Red Hat Enterprise Linux (RHEL), CentOS, SUSE Linux Enterprise Server (SLES), openSUSE, Debian and Fedora. It can also host Microsoft Windows images. Similarly Eucalyptus can use a variety of virtualization technologies including VMware, Xen and KVM hypervisors to implement the cloud abstractions it supports. Eucalyptus is an acronym for Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems.[13, 14]

Following is a brief description of different components and their functions of Eucalyptus open source Cloud system[15]:

2.2.1.1 Node Controller (NC)

Node controller is installed on each physical resource (node). It controls and administers the life cycle virtual machine (VM) instances running on it. The node controller interface described through WSDL provides functionalities to manage its VMs such as runInstance, terminateInstance, describeInstance, describeResource and startNetwork. The NC interacts with the OS and the hypervisor running on the node on one side and the Cluster Controller (CC) on the other side.
Function :

- Manage the life cycle of Instance.

- NC is also responsible for assisting CC to control VM instances on a node, verifying the authorization, confirming resources availability and executing the request with the hypervisor.

2.2.1.2 Cluster Controller (CC)

Cluster controller runs on one machine per cluster and works as an intermediary between the cloud controller and node controller. CC manages one or more Node Controllers and deploys/manages instances on them. CC is connected to both public and private networks. CC communicates with Cloud Controller (CLC) on one side and NCs on the other side. Its WSDL is similar to that of a node controller and its functions are runInstances, terminate-Instances, describeInstances and describeResource[13].
Function :

- It gets request for deploying instance from CLC and decides NCs on which instance to be deployed.
- CC is responsible to collect/report information about NCs to CLC and to manage virtual instance network overlay.

2.2.1.3 Cloud Controller (CLC)

Cloud controller is the user's entry point into the Eucalyptus system and provides users with a way managing the system. CLC provides an EC2/S3 compliant web services interface to the client tools on one side and interacts with the rest of the components of the Eucalyptus infrastructure on the other side. According to [13], the cloud controller is built using the Enterprise service bus providing decoupling from the services implementation and the users requests which enabled the use of multiple interfaces and different implementations of such services.
Function :

- Monitor the availability of resources on various components and manage the Resource arbitration.
- Handling user's (regular users and administrators) requests, schedules VM instance creation, maintain system and user data and process service level agreements .

2.2.1.4 Walrus Storage Controller (WS3)

WS3 provides a persistent simple storage service, means it provides static storage for VM instance which will be deleted with VM instance is terminated.Its APIs are compatible with Amazons S3 APIs. Walrus should be considered as a simple file storage system.
Function :

- It stores the VM images,users data and snapshots.
- Storing and serving files using S3 API.

2.2.1.5 Storage Controller (SC)

This component provides dynamic block storage facility like Elastic Block storage of Amazon. In this you attach, detach blocks to any instance which are there even instance terminates. Function :

- Create, attach, detach EBS devices.
- Providing the block storage over AoE or iSCSI protocol to the instances.

2.2.1.6 Networking

In cloud one of the most attractive feature is virtual networking, in which we can configure virtual network as per our need in-spite of underlying complex and restrictive physical network. In Eucalyptus VM instance network address the issues of connectivity and isolation.

- Virtual Network Connectivity : Each instance in eucalyptus is assigned two virtual network interfaces, one is private which is connected to private network and other is public connected to public network. The Public interface is responsible for communication outside the given set of VM instances or between instances within the same availability zone. While private interface takes care of communication between instances which are disperse over two different availability zone.
- Virtual Network Isolation : To provide isolation for two instances which are same physical resource owned by two users, instances are assigned tags known as VLAN tags. VLAN are used by tagging each packet with its VLAN name which would make the VDE switch send the packet to that VLAN only. Figure 2 Shows the use of private and public interfaces [16].

2.2.2 OpenStack

OpenStack is a collection of open source technologies delivering a massively scalable cloud operating system.[17] Lets see how OpenStack Compute is logically architect-ed. Since Cactus is the newest release, I will concentrate there (which means if you are viewing this after around July 2011, this will be out of date). There are several logical components of OpenStack Compute architecture but the majority of these components are custom written python daemons of two varieties:

- WSGI applications to receive and mediate API calls (nova-api, glance-api, etc.)
- Worker daemons to carry out orchestration tasks (nova-compute, nova-network, nova-schedule, etc.)

2.2.2.1 Overview of the Components

Now here is an overview of all the components, how and what do they do[17] as following:

The nova-api daemon is the heart of the OpenStack Nova.It provides an endpoint for all API queries (either OpenStack API or EC2 API), initiates most of the orchestration activities (such as running an instance) and also enforces some policy (mostly quota checks).

The nova-schedule process is conceptually the simplest piece of code in OpenStack Nova: take a virtual machine instance request from the queue and determines where it should run , specifically, which compute server host it should run on.

The nova-compute process is primarily a worker daemon that creates and terminates virtual machine instances. The process by which it does so is fairly complex but the basics are simple: accept actions from the queue and then perform a series of system commands (like launching a KVM instance) to carry them out while updating state in the database.

The nova-volume manages the creation, attaching and detaching of persistent volumes to compute instances It can use volumes from a variety of providers such as iSCSI or AoE.

The nova-network worker daemon is very similar to nova-compute and nova-volume. It accepts networking tasks, the queue and then performs tasks to manipulate the network .

The queue provides a central hub for passing messages between daemons. This is currently implemented with RabbitMQ today, but theoretically could be any AMPQ message queue supported by the python ampqlib.

The SQL database stores most of the build-time and run-time state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, networks available and projects.

OpenStack Glance is a separate project from OpenStack Nova, but is is complementary.There are three pieces to Glance: glance-api, glance-registry and the image store.glance-api accepts API calls, much like nova-api, and the actual image blobs are placed in the image store. The glance-registry stores and retrieves metadata about images.

The OpenStack Dashboard provides a web interface into OpenStack Nova to give application developers and devops staff similar functionality to the API. This is also an optional component.

2.2.3 OpenNebula

OpenNebula is a fully open-source management toolkit for on-premise Infrastructure as a Service (IaaS) cloud computing. OpenNebula can be primarily used as a virtualization tool to manage your virtual infrastructure in the data-center or cluster, which is usually referred as Private Cloud. OpenNebula supports Hybrid Cloud to combine local infrastructure with public cloud-based infrastructure, enabling highly scalable hosting environments. OpenNebula also supports Public Clouds by providing Cloud interfaces to expose its functionality for virtual machine, storage and network management. [18] The aim of a Private Cloud is not to expose to the world a cloud interface to sell capacity over the Internet, but to provide local

cloud users and administrators with a flexible and agile private infrastructure to run virtualized service workloads within the administrative domain. OpenNebula virtual infrastructure interfaces expose user and administrator functionality for virtualization, networking, image and physical resource configuration, management, monitoring and accounting. An OpenNebula Private Cloud provides infrastructure users with an elastic platform for fast delivery and scalability of services to meet dynamic demands of service end-users. Services are hosted in VMs, and then submitted, monitored and controlled in the Cloud by using the OpenNebula Operations Center or any of the OpenNebula interfaces (1) Command Line Interface (CLI), (2) XML-RPC API, (3) OpenNebula Cloud APIs and (4) Libvirt virtualization API or any of its management tools.

OpenNebula was designed to address the requirements of business use cases from leading companies and across multiple industries, such as Hosting, Telecom, eGovernment, Utility Computing. The principles that have guided the design of OpenNebula are:

- Openness of the architecture, interfaces, and code
- Adaptability to manage any hardware and software combination, and to integrate with any product and service in the cloud and virtualization ecosystem
- Interoperability and portability to prevent vendor lock-in
- Stability for use in production enterprise-class environments
- Scalability for large scale infrastructures
- Standardization by leveraging and implementing standards

2.2.3.1 Tasks Managed by OpenNebula

The tasks manged by OpenNebula are at a glance[18] are the following:

Management of the Network, Computing and Storage Capacity :
Orchestration of storage, network and virtualization technologies to enable the dynamic placement of the multi-tier services on distributed infrastructures

Management of VM Life-cycle :
Smooth execution of VMs by allocating the resources required for them to operate and by offering the functionality required to implement VM placement policies

Management of Workload Placement :
Support for the definition of workload and resource-aware allocation policies such as consolidation for energy efficiency, load balancing, affinity-aware, capacity reservation

Management of Virtual Networks :
Support for the definition of virtual networks to interconnect VMs

Management of VM Images :
Exposing of general mechanisms to transfer and clone VM images. Images can be registered

before execution. When submitted, VM images are transferred to the host and swap disk images are created. After execution, VM images may be copied back to the repository.

Management of Information and Accounting :
Provision of indicators that can be used to diagnose the correct operation of the servers and VMs and to support the implementation of the dynamic VM placement policies

Management of Security :
Definition of security policy on the users of the system, guaranteeing that the resources are used only by users with the relevant authorizations and isolation between workloads

Management of Remote Cloud Capacity :
Dynamic extension of local capacity with resources from remote providers to build hybrid or federated cloud deployments

Management of Public Cloud Servers :
Exposing of most common cloud interfaces to support public cloud computing deployments

2.3 Study of Public Cloud – Amazon Cloud Service

Amazon Elastic Compute Cloud (EC2)[19, 20] is a central part of Amazon.com's cloud computing platform, Amazon Web Services (AWS). EC2 allows users to rent virtual computers on which to run their own computer applications. EC2 allows scalable deployment of applications by providing a Web service through which a user can boot an Amazon Machine Image to create a virtual machine, which Amazon calls an "instance", containing any software desired. A user can create, launch, and terminate server instances as needed, paying by the hour for active servers, hence the term "elastic". EC2 provides users with control over the geographical location of instances that allows for latency optimization and high levels of redundancy.

2.3.1 Operating Systems

When it launched in August 2006, the EC2 service offered Linux and later Sun Microsystem's OpenSolaris and Solaris Express Community Edition. In October 2008, EC2 added the Windows Server 2003 and Windows Server 2008 operating systems to the list of available operating systems. As of December 2010 it has also been reported to run FreeBSD; in March 2011, NetBSD AMIs became available.

2.3.2 Persistent Storage

An EC2 instance may be launched with a choice of two types of storage for its boot disk or "root device". The first option is a local "instance-store" disk as a root device (originally the only choice). The second option is to use an EBS volume as a root device.

Instance-store volumes are temporary storage, which survive rebooting an EC2 instance, but when the instance is terminated (e.g., by an API call, or due to a failure), this store is lost.

EBS volumes provide persistent storage independent of the lifetime of the EC2 instance, and act much like hard drives on a real server. More accurately, they appear as block devices to the operating system that are backed by Amazon's disk arrays. The OS is free to use the device however it wants. In the most common case, a file system is loaded and the volume acts as a hard drive. Another possible use is the creation of RAID arrays by combining two or more EBS volumes. RAID allows increases of speed and/or reliability of EBS. Users can set up and manage storage volumes of sizes from 1GB to 1TB. The volumes support snapshots, which can be taken from a GUI tool or the API. EBS volumes can be attached or detached from instances while they are running, and moved from one instance to another.

Simple Storage Service (S3)[21] is a storage system in which data is accessible to EC2 instances, or directly over the network to suitably authenticated callers -all communication is over HTTP. Amazon does not charge for the bandwidth for communications between EC2 instances and S3 storage in the same region. Accessing S3 data stored in a different region (for example, data stored in Europe from a US East Coast EC2 instance) will be billed at Amazon's normal rates.

S3-based storage is priced per gigabyte per month. Applications access S3 through an API. For example, Apache Hadoop supports a special s3: file-system to support reading from and writing to S3 storage during a MapReduce job. There are also S3 file-systems for Linux, which mount a remote S3 file-store on an EC2 image, as if it were local storage. As S3 is not a full POSIX file system, things may not behave the same as on a local disk (e.g., no locking support).

2.3.3 Elastic IP Addresses

Amazon's Elastic IP Address feature is similar to static IP address in traditional data centers, with one key difference. A user can programmatically map an Elastic IP Address to any virtual machine instance without a network administrator's help and without having to wait for DNS to propagate the new binding. In this sense an Elastic IP Address belongs to the account and not to a virtual machine instance. It exists until it is explicitly removed. It can also be used as unused IP address[clarification needed]. IPv6 is provided in the US East (Northern Virginia) and EU (Ireland) regions.

2.3.4 Amazon CloudWatch

Amazon CloudWatch is a Web service that provides real-time monitoring to Amazon's EC2 customers on their resource utilization such as CPU, disk and network. Cloudwatch does not

provide any memory, disk space, or load average metrics. An Amazon engineer has stated that this is due to the requirement to install software in the VM - something they wish to avoid. The data is aggregated and provided through AWS management console. It can also be accessed through command line tools and Web API's, if the customer desires to monitor their EC2 resources through their enterprise monitoring software.

The metrics collected by Amazon CloudWatch enables Auto Scaling feature to dynamically add or remove EC2 instances. The customers are charged by the number of monitoring instances.

Since May 2011, Amazon CloudWatch accepts custom metrics that can be submitted programmatically via Web Services API and then monitored the same way as all other internal metrics, including setting up the alarms for them.

2.3.5 Automated Scaling

Amazon's Auto Scaling allows you to scale one's Amazon EC2 capacity up or down automatically according to conditions one define. With Auto Scaling, one can ensure that the number of Amazon EC2 instances one using increases seamlessly during demand spikes to maintain performance, and decreases automatically during demand lulls to minimize costs. Auto Scaling is enabled by Amazon CloudWatch.

2.4 Distributed Geospatial Services

The main objective of Distributed Geospatial Services is to provide a platform for sharing geospatial information from the distributed source in an interoperable manner. It is the services that handle the geospatial for shearing and distribution. The SOA is the most promising architecture for the implementation of interoperability principles. Theses services are type of Web services which deal with geographical information and can provide access to geographical information stored in a database, perform simple and complex geospatial analysis and return messages that contain geographical information [22].

2.4.1 Encoding of Geospatial Data

The roles of the data provider and the service provider are distinct. However, for sharing and transferring of data, it is required to encode geospatial data in such a way, so that it can be understandable between both parties. Moreover, the use of widely varying client display facilities,from the high-resolution screens of desktop computers to the miniature displays available on modern tiny mobile devices, makes it necessary to separate the presentation characteristics of the service from the information content it is based on. A well-designed service architecture should facilitate the geospatial services in supporting varying user needs with most current data. The introduction of the *Semantic Web* concept is an important step in this evolution.

According to initial Semantic Web ideas, the semantics is to be attached to the Web content by means of Extensible Markup Language (XML) technologies. XML specification provides the logical structure of data, so that the data can be understandable between the service provider and consumer. Considering XML technology, OGC have proposed a standard for encoding geospatial data, called Geography Markup Language (GML).

2.4.2 Geospatial Data Services

Geospatial data service is the essential component of the proposed architecture. It provides access and manipulation operations on geographic features using HTTP as the underlying protocol. The de facto data interchange format of data service is GML. To achieve interoperable geospatial data services, OGC provides a set of specifications, called Web Feature Services (WFS) for developing standard data service interface [23]. It is a widely adopted standard in most of the GIS applications to access spatial data in different data sources over the Internet. WFS provides two classes of services as following.

Basic Web Feature Service:

The basic WFS is considered a read-only web feature service and solely providing data output facilities. It offers three types of operations

- ***GetCapabilities*** – The purpose of this operation is to describe capabilities data server. Specifically, it must indicate which feature types it can service and what operations are supported on each feature type. It provides an XML-encoded capabilities document which contains names of feature types, the spatial reference system, the spatial extent of the data and information about the operations that are supported.

- ***DescribeFeatureType*** – This operation provides the overall data structure a geospatial feature which is supplied by the service. This operation is used to retrieve an XML Schema document with a description of the structure of the feature types which are served by that WFS service.

- ***GetFeature*** – This operation is used to access the actual feature instance in GML format. In addition, the client could be able to specify the constraint to filter the feature instance based on the feature properties.

2.4.3 Geospatial Map Services

The geospatial map service produces map of geo-referenced data for its client applications in the form of an image rather than actual data. Since the visual representation of data is more comprehensible than the data itself. Clients request maps from the map service in terms of named layers and provide parameters such as the size of the returned image as well as the spatial reference system for rendering the map. Each layer of map corresponds to a specific

type of geographical features offered by data service. So each map service should depend on the geospatial data services [24].

- ***GetCapabilities*** – It is a mandatory operation to provide service-level metadata. It is a machine-readable as well as human-readable description about server information content along with acceptable request parameters.
- ***GetMap*** – It is also a mandatory for accessing geospatial map. The service request should contain a well defined geographical and dimensional parameters. The content of the map may contain one or more themes, or map layers. The client application must indicate which layers to include in the map image. For each map layer, the server may provide a list of styles in which the corresponding layer can be visualized.
- ***GetFeatureInfo*** – This service may optionally allow the GetFeatureInfo operation. When a client needs any further information about the feature on the requested map According to this operation, a Client can request information about features on a map by adding to the map URL additional parameters specifying a location and the number of nearby features about which to return information.

2.4.4 Geospatial Registry Services

A registry service acts an open service directory in a distributed geospatial network systems. Within service registry, the service providers publishes the availability of their resources using metadata, and the service consumers can then query the descriptive information abou data or metadata in the registry to discover relevant resources and bind that resources to access required information. Metadata within service registry represents resource characteristics that can be queried and presented for evaluation and further processing. OGC Catalog Service for Web (CSW) provides service interface standards for geospatial registry services. It provides common interface standards for enabling the ability to publish and search collections of metadata for data, services, and related information objects. It specifies a formal structure representing geospatial metadata resources and their interrelationships and a conceptual schema constraining the kinds of objects stored in the registry and how these registry objects and the relevant descriptive information are organized and interpreted.

2.5 Summary

In this chapter we have seen some related works at first, then lot of case studies of Open source cloud systems & Amazon public cloud service and in the end small description of the important Geospatial Services. One point has to be clear at this point is that Amazon public cloud is now considered as the standard of for the cloud systems. Among all the open source cloud we have discussed earlier in this chapter, Eucalyptus is the one that is the most

similar in working to the Amazon's. So at this point it is clear that using Eucalyptus has its own advantages of being considered as proper cloud very easily. We will see why we choose Eucalyptus over others in much more detail in Chapter 4. But before let us have a look at the models for the system design in the following Chapter.

Chapter 3

Integration of Standard Service Models of Cloud

From this chapter onward the work has been discussed. This particular Chapter is about the models that can be considered as the basics of the work and further discussion. First section in this chapter gives a brief idea about the standard service models of the cloud namely IaaS, PaaS and SaaS. Afterwards the proposed model which is constructed by the integration of these three models is discussed and in the last section the problem modeling is discussed.

3.1 Standard Service Models of Typical Cloud

For a typical system to be called a cloud it needed to deploy itself to provide at least one of the standard service models namely Infrastructure as a Service, Platform as a Service and Software as a Service. Following is a small description on each.

3.1.1 Infrastructure as a Service [IaaS]

In this most basic cloud service model, cloud providers offer computers as physical or more often as virtual machines , raw (block) storage, firewalls, load balancers, and networks. IaaS providers supply these resources on demand from their large pools installed in data centers. Local area networks including IP addresses are part of the offer. For the wide area connectivity, the Internet can be used or - in carrier clouds - dedicated virtual private networks can be configured.

To deploy their applications, cloud users then install operating system images on the machines as well as their application software. In this model, it is the cloud user who is responsible for patching and maintaining the operating systems and application software. Cloud providers typically bill IaaS services on a utility computing basis, that is, cost will reflect the amount of resources allocated and consumed [25].

3.1.2 Platform as a Service [PaaS]

In the PaaS model, cloud providers deliver a computing platform and/or solution stack typically including operating system, programming language execution environment, database, and web server. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers. With some PaaS offers, the underlying compute and storage resources scale automatically to match application demand such that the cloud user does not have to allocate resources manually [25].

3.1.3 Software as a Service [SaaS]

In this model, cloud providers install and operate application software in the cloud and cloud users access the software from cloud clients. The cloud users do not manage the cloud infrastructure and platform on which the application is running. This eliminates the need to install and run the application on the cloud user's own computers simplifying maintenance and support. What makes a cloud application different from other applications is its elasticity. This can be achieved by cloning tasks onto multiple virtual machines at run-time to meet the changing work demand. Load balancers distribute the work over the set of virtual machines. This process is transparent to the cloud user who sees only a single access point. To accomodate a large number of cloud users, cloud applications can be multitenant, that is, any machine serves more than one cloud user organization. It is common to refer to special types of cloud based application software with a similar naming convention: desktop as a service, business process as a service, Test Environment as a Service, communication as a service. The pricing model for SaaS applications is typically a monthly or yearly flat fee per user [25].

3.2 Proposed Model for Geospatial Cloud

Before going to the discussion of the model let us consider one example, basically its the idea of coding style. In conventional codes we first visualize the problem and solve the same by applying logic with the code snippets. As time goes, we improved the our coding style by dividing the problem into sub-problems and code for each of them, these sub-problems are called modules. Later we tried to make these modules and tried to make them as independent to main problem as possible. Once this is done now we can see these modules to be independent code snippets which can be used with some other problem as per similar requirements. We now call the collection of the code snippets library and the idea is called Modularization. In this particular discussion we will be linking to the similar concept of modularization, but this time the idea is to be realized into the physical world entities. The big challenge here is whether it is possible if so then how. That is where the idea of cloud comes into the picture. In computer science, general users stick to the most abstract end of the whole system i.e.

they are concerned with the jobs to be done, all they care about is the output. But for the output to come properly there are certain background phenomenon that needs to be taken care of by the providers. In the background first to serve a request from an user first thing that is needed is an application. Now to run an application it needs an environment typically called the run-time (e.g. jvm, jre, Operating Systems, Database services). Now to run such an run-time or operating system it needs some physical infrastructure to run on. So broadly it needs three main modules to serve a request, they are in reverse order infrastructure, platform and the software. Thus if we can separates out each of the modules and serve each of them independently, conversely we can achieve the modularization in the physical level. Cloud computing has three main modes of deployment which are respectively Infrastructure as a Service, Platform as a Service and Software as a service, which are as the name suggests can give us the three level of independent modular services as discussed earlier.

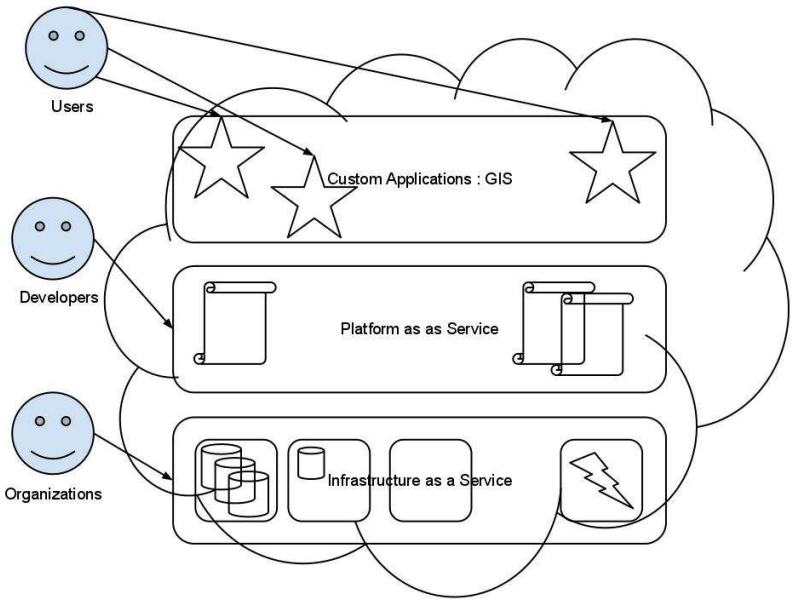


Figure 3.1: Actors at Different Levels in the Cloud

Now let us put together all these models. This particular case is shown in the Figure 3.1 when we put one model over another one complete business models can be represented and the figure shows the positions of different level of users of the system.

In a cloud all different kind of actors are having its place,[Figure 3.1] such as an organization normally tend to buy computing infrastructures which can be obtained from the IaaS service paradigm of the cloud. Now having a company established it will have workers who will be working on those resources of the company but obviously been provided a platform to work on, which again can be fetched as PaaS service over a cloud. Finally one has to deploy products/Web applications more specifically, to achieve the goal of productivity, it is now as simple as deploying on the SaaS environment. So all the requirement an organization is

having can be met with one service – The Cloud Service.

3.3 Modeling a Scenario for Geospatial Cloud

In case of GIS there can be many back-end data services available for providing the different spatial data layers for the applications to be fetched. For instance let us consider there are 3 such services there. Now for each application it may not be necessary to contact all the services so required services (or a caching server for each service) and another host can be bind together to complete the system. Once this is done each custom application can be made available to the public for using.

In case of GIS Technology, as it demands huge amount of data processing in a efficient time and it is an on-demand service, so that the requirement of the resource can differ in different times. Clearly Cloud in this kind of scenario comes as quite promising infrastructure for organization that has the service. Now that they don't have to care much about the management of the infrastructure they can just concentrate on the data only. At the same time when an organization wants to provide web services developer shouldn't need to worry about the infrastructure either, here the organization can provide them only with the platform to use, similarly cloud can provide that with the PaaS service, so that they can concentrate only on the developing. Following is general case to understand the problem. Figure 3.2 illustrates the problem.

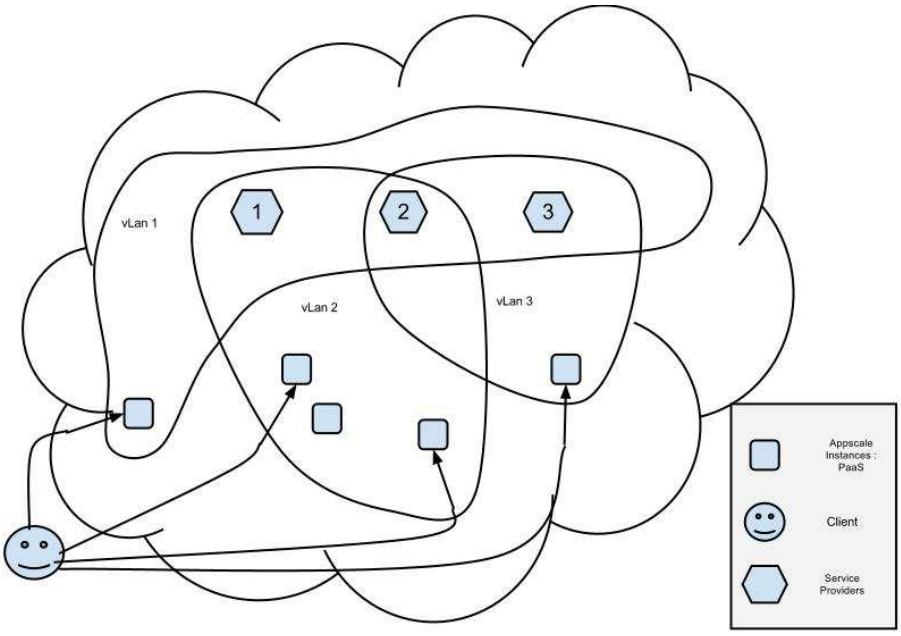


Figure 3.2: Problem Illustration for Geospatial Cloud

Now that an organization can have their services running over instances inside the cloud, by using grouping facilities they can provide layer 2 or layer 3 security to the instances. Thus

only required services can be connected together and can be bind with the desired services. Again the services can be run through the PaaS service model, so that load balancing, distributed replica and dynamic scaling can be done in a transparent way to the developers who are the client for the platform service. In the end deployed application/services can now be viewed to the client as a software as a service. Thus it can be integrated as a SaaS-PaaS-IaaS module altogether(as discussed in Section 3.2), and providing a complete industry structure in the cloud itself. Now in addition to that if a service needs to be fetched from outside the cloud it can also be done easily by simple network accesses. Moreover a dedicated Catalog service can be incorporated for searching required data server or any other service, so that depending on the requirement of the customer at different level of the model can be entertained easily.

Chapter 4

Cloud Setup and Integrated Model Realization

As the proposed model is discussed in the Chapter 3 the first thing that is needed to accomplish is provision of Infrastructure as a Service. In this case the open source project Eucalyptus has been selected for the role. In this chapter first we will see what are the upsides of selecting Eucalyptus and then an overview of the setup, configurations and the resources used for the same. Following is a section based on the user-system interaction in the Cloud. Once the IaaS has been provided successfully, now the next task is to provide Platform as a Service and then the Software as a Service on top of that to fulfill the model requirement. In this Chapter the next discussion section is on what a platform really stands for and one way of demonstrating the same. Afterwards how application(SaaS) can be deployed on the platform is shown.

4.1 Advantages of Eucalyptus

Eucalyptus Open-Source Cloud Computing Infrastructure. Eucalyptus implements IaaS (Infrastructure as a Service) style private and hybrid clouds. The platform provides a single interface that lets users access computing infrastructure resources (machines, network, and storage) available in private cloudsimplemented by Eucalyptus inside an organization's existing data centerand resources available externally in public cloud services. The software is designed with a modular and extensible Web services-based architecture that enables Eucalyptus to export a variety of APIs towards users via client tools. Currently, Eucalyptus implements the industry-standard Amazon Web Services (AWS) API, which allows the interoperability of Eucalyptus with existing AWS services and tools. Eucalyptus provides its own set of command line tools called Euca2ools, which can be used internally to interact with Eucalyptus private cloud installations or externally to interact with public cloud offerings, including Amazon EC2.

Eucalyptus includes these features [13] [26]:

- Compatibility with Amazon Web Services API.
- Installation and deployment from source or DEB and RPM packages.
- Secure communication between internal processes via SOAP and WS-Security.
- Support for Linux and Windows(Experimental) virtual machines (VMs).
- Support for multiple clusters as a single cloud.
- Users and Groups Management.
- Accounting reports.
- Configurable scheduling policies and SLAs.

4.2 Eucalyptus Cloud: Setup

Setting up Eucalyptus is primarily about the installation of the packages but the main concern is to configure the setup accordingly to the requirements of the organization. This part concerns about the resources available for the setup and the network configuration accordingly. Following in this section we will see about these two concerns separately.

4.2.1 Device Configurations

For the project we have used three Dell rack servers with a total 56 CPU modules, 80 GB of RAM and 1.5 TB of hard drive as the background resource/infrastructure to the GIS cloud. Apart from that there is another machine running the Cloud Controller service. The detailed configuration of the system used is given in the Table 4.1

4.2.2 Network Configurations

Our Eucalyptus configuration uses two different network for its configuration. First is the enterprise network which can be public for users of the system. Second, is the internal network of the eucalyptus components with which components are privately connected to each other. There is another network is used for the configuration which another private network to be assigned to the instances internally, and a set of public ip to be assigned to the instances for public access. Thus each instance running under the cloud infrastructure will be having assigned two IPs.
Table 4.2 shows the network configuration list. And the allocation of Ip address and the system services descriptions can be found in Table 4.3.

Table 4.1: Device Configurations

System	Processor	Other Information
Node Controller NC1 and NC2	Intel Processor Xeon X5570	Form Factor: 1U Rack Mount Server Processor: Intel Xeon processor 5500-based series 2.93 GHz; 2-way x 4 cores = 8 cores(x2 Memory: 16 GB RAM Storage: 280 GB HDD
Node Controller NC3	Intel Processor Xeon X5570	Form Factor: 1U Rack Mount Server Processor: Intel Xeon processor 5500-based series 2.93 GHz; 2-way x 6 cores = 12 cores(x2) Memory: 48 GB RAM Storage: 1.2 TB HDD
Cloud Controller, Cluster Controller, Walrus Storage Service,and Block Storage Controller	Intel Core2Duo	Form Factor: PC; Processor: Intel Core2Duo Memory: 2 GB RAM Storage: 320 GB HDD

Table 4.2: Network Configurations

Network	Network Address
Cloud Component Network	192.168.10.0/24
Enterprise Network	192.168.20.0/24
Private Network for Instances	172.1.0.0/25
Public IP Range	192.168.10.128 - 223

Table 4.3: Systems, Services and Allocation of IPs

Component(s)	IP Address	Operating System	Running Service(s)
CLC-SC-Walrus Front-end	192.168.10.121 192.168.20.1	UBUNTU server 11.04	Eucalyptus cloud controller, storage controller, walrus service
NC1	192.168.20.2	UBUNTU server 11.04	Eucalyptus node controller service
NC2	192.168.20.3	Fedora 13	Eucalyptus node controller service
NC3	192.168.20.4	UBUNTU server 11.04	Eucalyptus node controller service

4.3 Users in Eucalyptus

Once the Cloud has been set up next most important thing is to give access to the users of the system. For each user running one or more instance in the cloud will have an access key, in this case called a *keypair*, by which users can securely log in to the instances of theirs. Following is a step-by-step description of how the user will interact with Eucalyptus.

4.3.1 System-User Interaction

The way a user interacts with the system i.e. the cloud is as follows

- A client/user in the eucalyptus public/enterprise network who has a login information in the cloud requests to the cloud controller to start a new instance of an image with his resource requirements and a keypair (which allows the user to login later) if available with the cloud.
- On receive of the request from the client/user cloud controller module checks if the image the user is requesting is available and if the resource requirements can be met at that time, if both conditions can be met then cloud controller contacts the cluster controller and proceeds further.
- Cluster controller then selects a node from its pool of available nodes which can supply on the required resource and notifies it about the task i.e. requirements and the image information
- In the mean time the Cloud controller decides the Public IP and Private IP to be assigned to the instance and put the entry for mapping of the Private and Public IP in the firewall rules (basically SNAT and DNAT).
- now as the node controller knows all the information including the Private IP it contacts the Storage Controller for the Image to be run.
- Once the image is there with the Node controller it starts the instance for the image and assignes the Private Ip to the instance.
- In the process of booting the instance should contact the metadata service of the cloud so that it can get in login information corresponding to the keypair for the user who has started it.
- Now the cloud controller is notified about the status of the instance i.e. running state, which indeed get notified to the user.
- Now as the user has the status of the instance as running and the Public IP (as well as the Private IP) and the user is sitting in the public network, can get access to the instance by ssh and with the keypair with which the instance is been associated.

From the routing point of view the interaction with the cloud is done in the following

- Incoming
  - A request comes to the Public IP must come through the Cloud Controller which is configured with the firewall rules.

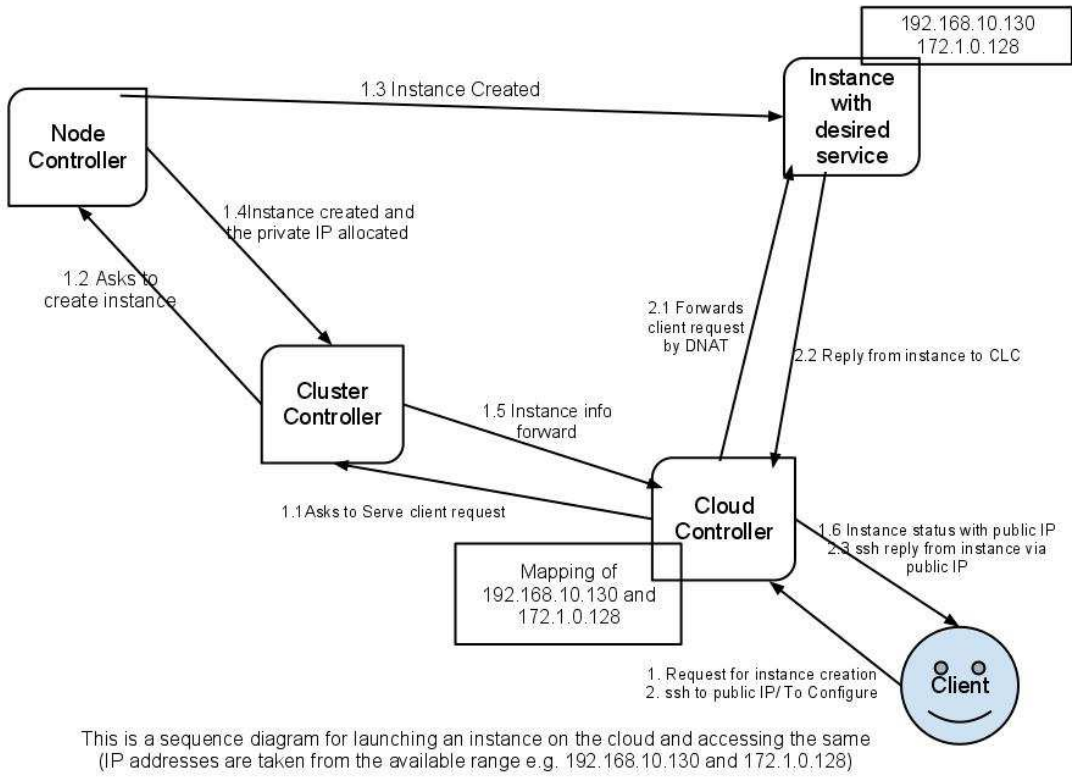


Figure 4.1: Block Diagram of an User Accessing an Instance in the Cloud

- When a incoming request is triggered it uses the DNAT rules defined in the firewall PREROUTING chain of the 'nat' table.
- Now the packet can directly goes to the instance it supposed to go and returns with the reply to the cloud controller which in turn again forwarded as the DNAT table to the actual requesting IP.

- Outgoing
  - A request comes from the Instance(Private IP) for some Public network IP must go through the Cloud Controller which is configured with the firewall rules.
  - When a outgoing request is triggered it uses the SNAT rules defined in the firewall POSTROUTING chain of the 'nat' table.
  - Now the packet can directly goes to the Public network IP it supposed to go and returns with the reply to the cloud controller which in turn again forwarded as the SNAT table to the actual requesting Instance.

A simpler version of the procedure is shown in the Figure 4.1.

4.3.2 User Commands

Eucalyptus can be accessible over the web-interface and also via the command line using commands provided by Euca2ools. The commands to used for accessing Eucalyptus are listed in Table 4.3.2.

Table 4.4: Euca2ools Commands

euca-add-group	euca-create-volume	euca-describe-image-attribute
euca-get-password-data	euca-add-keypair	euca-delete-bundle
euca-describe-images	euca-modify-image-attribute	euca-allocate-address
euca-delete-group	euca-describe-instances	euca-reboot-instances
euca-associate-address	euca-delete-keypair	euca-describe-keypairs
euca-register	euca-attach-volume	euca-delete-snapshot
euca-release-address	euca-authorize	euca-reset-image-attribute
euca-describe-snapshots	euca-delete-volume	euca-bundle-image
euca-deregister	euca-describe-volumes	euca-revoke
euca-bundle-instance	euca-run-instances	euca-describe-addresses
euca-detach-volume	euca-bundle-vol	euca-disassociate-address
euca-describe-availability-zones	euca-terminate-instances	euca-cancel-bundle-task
euca-describe-bundle-tasks	euca-download-bundle	euca-unbundle
euca-confirm-product-instance	euca-get-console-output	euca-upload-bundle
euca-describe-group	euca-create-snapshot	euca-describe-groups
euca-get-password	euca-version	euca_conf

4.4 Custom Image Creation

As previously faced the difficulty for the non-persistent instances, every time one user had to configure the instances to make the things run. Now an image has been created and configured (Ubuntu 11.04 server ed.) in such a way that instances that are running this image will be completely configured to go as they launched. Now the only thing required is that to run it put your app inside the instance and its all done.

As other running instances will be providing different services and their APIs which can be accessed by the valid owners/users of other applications to use and run their web application successfully. And by this model the billing can now be done as per the usage of the data too.

4.5 Deploying Applications

This is second module of the proposed model in Figure 3.1 i.e. Platform as a Service. In fair understanding of the platform as a service architecture the platform can actually be as vast as an Operating system or as simple as a run-time environment. In our case to run a Geospatial Web application it basically requires required resources to compute that is data (WFS service to be specific) and a runtime environment (in our case it is apache2). Now runtime can be run on some operating system which can be running on a machine(in our

case virtual machine/instances). So clearly on the IaaS architecture that we have the PaaS service is provided. A web portal is there which can show you the current status of the cloud. and one can deploy their web application suitable for the environment just by some clicks and the application will be running. Until now the basic framework has been built which can deploy web applications.

To illustrate the idea here are two screen shots of the system to provide PaaS Figure 4.2 and Figure 4.3

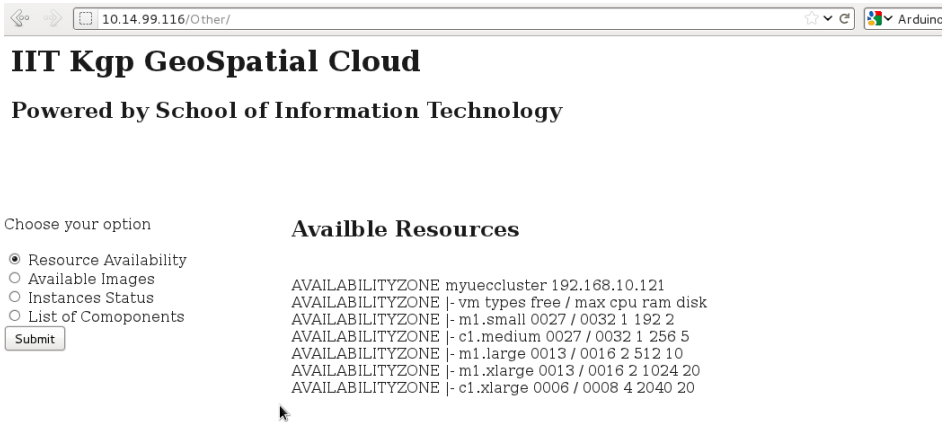


Figure 4.2: Screen Capture: Geospatial Cloud service Homepage

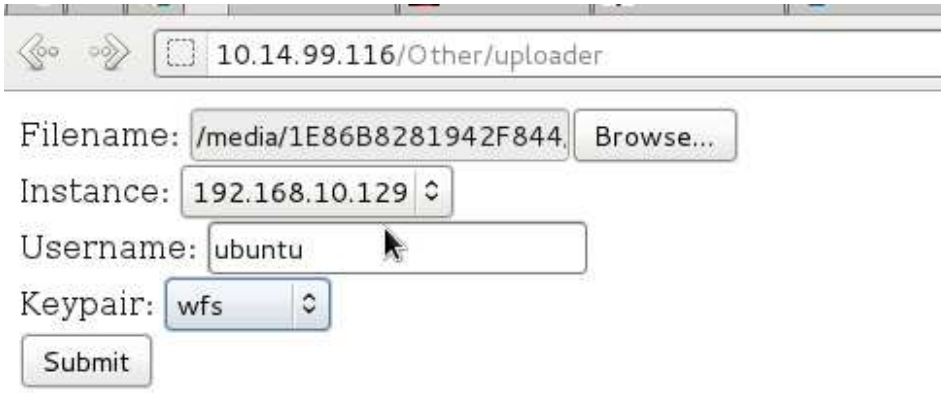


Figure 4.3: Screen Capture: Platform as a Service Portal

4.6 Summary

In this chapter all about the set up the proposed model has been discussed. To summarize we choose Eucalyptus to be the lowest level of the model i.e. for the provision of Infrastructure as a Service, then on top of it we choose JAVA to be the runtime environment for the developers and selected apache-tomcat webserver for deployung java applications and created a portal for the deployment of the application over the web and it has been thoroughly tested with sample web applications. The only things required now is to deploy Geospatial services and the goal of this work will be achieved. In the following chapter we will see how the Geographical Information System in terms of Enterprise–GIS Framework can be integrated with this set up.

Chapter 5

Geographical Information System on Eucalyptus

After all the previous discussions about the different models of cloud and their proposed integration scenario and in the next chapter the Cloud set up using open source cloud system Eucalyptus, now in this chapter we will see how Geographical Information System has been integrated with Eucalyptus and how are the different scenarios in this particular situation. In the end an example scenario has been discussed and some screenshots from the system show the system activity.

5.1 Deployment of Enterprise GIS Framework over Eucalyptus

In this let us have a look at the Enterprise GIS Framework and following the methods of deploying the same.

5.1.1 Enterprise GIS Framework

The enterprise GIS is a platform which is able to integrate enterprise wise geospatial data sources and delivers these data in a uniform format or in form of map. In order to generate information from heterogeneous datasets, Web service is the key technology to integrate different geospatial data repositories. To develop enterprise GIS framework, the Web Service technology is utilized as a platform of the work. In this work, an enterprise GIS platform has been developed. Through this platform, the mobile devices are able to access geospatial data in a uniform format from the diverse and heterogeneous data repositories. The mobile devices are also able to access the geospatial data in form of map [9].

5.2 System Behavior Models

As the problem stated earlier, it is required to deploy the geospatial services onto the cloud, which as we encountered can be done primarily in two different ways. Both the cases is described below.

5.2.1 Model to illustrate 'Everything in the Cloud'

Here we consider that we need to deploy the whole geospatial system in the cloud. In this case that means we need to provide the data services(WFS) as well as any other services that are intended to provide by the cloud(e.g. WMS- Web Map Service, CWS- Catalog Service, WPS- Web Processing Service etc.). For this we deployed the instances of Ubuntu Desktop Edition in the cloud and configure that image with required parameters such as java, apache-tomcat etc. After that the image is ready to go once we deployed the service over there. In the following figure the idea is illustrated.

Deployment Scenario 1 is shown in the Figure 5.1.

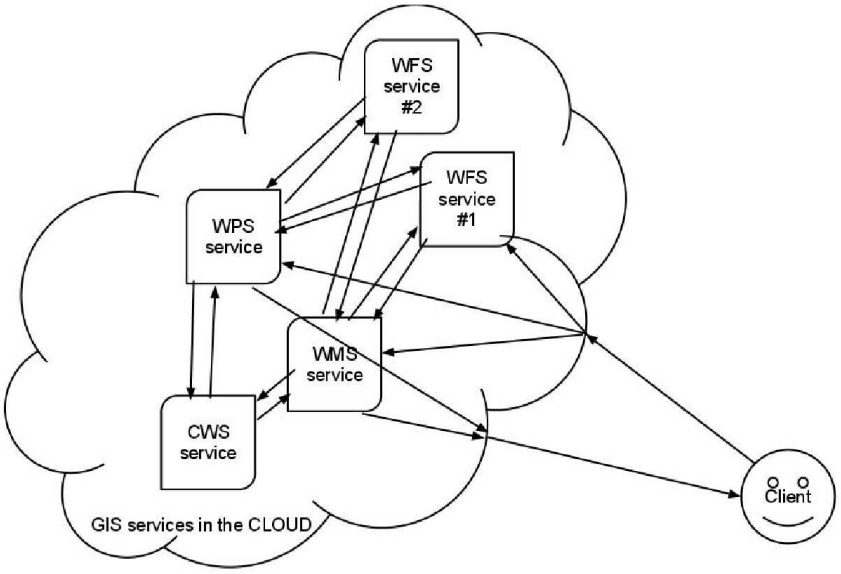


Figure 5.1: Deployment Scenario 1: Everything in the Cloud

This is a typical case for a Cloud System. All the services such as WFS, WMS, CWS etc. are provided through the cloud itself. All those services are deployed over one or more individual instance which runs from inside the cloud. In this case binding different services are very easy to handle, thus just by providing security from inside the cloud image isolation can be done and for each different Web Service provided to the public network can be made secure at the infrastructure level. But sometimes it is not possible for the service providers to

give up their data solely to the cloud without hesitation, for such cases the following model has been prepared.

5.2.2 Model to illustrate 'Data Providers are Outside the Cloud'

In contrast to the previous case here we consider that not all the basic sources are available with the cloud. As the Enterprise GIS Application uses many different sources of data from different providers and make integrate all those data by sitting on top of them, it has been seen that the data providers are not willing to giving up all the raw data to the organization but as a service. Now this problem can be solved by remodeling the problem where we have considered that the WFS of the data services are outside at their places and the cloud needs to take services from them when required by the other geospatial services which are still provided from inside the cloud. Thus the client side still dosen't know any more than contacting the cloud but receiving the reply for its request.

Deployment Scenario 2 is shown in the Figure 5.2.

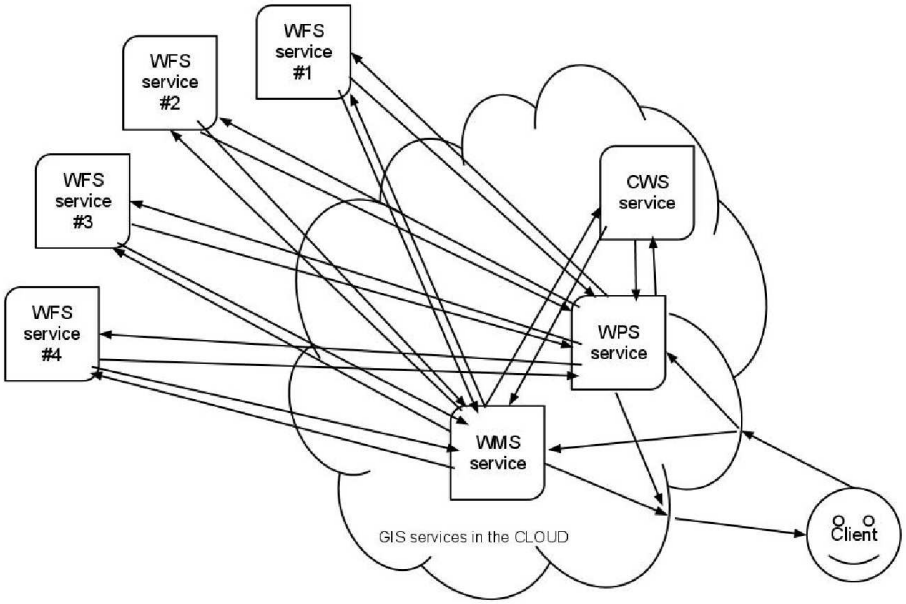


Figure 5.2: Deployment Scenario 2: Situation when Cloud has to take Other Services

In this figure as we can see that all the data services(which can be any other service that is required to stay outside) are outside of the cloud. In this case the computation such as creating maps or other computationaly expensive processing of the spatial data that is fetched from the data providers will be done in the cloud by the applications running inside the cloud (which can be done efficiently given that they use as much resource as required within the availability from the cloud) and then the results can be forwarded to the client.

Again here also the transparency for the user is preserved. But as cloud doesn't have any other access to the service providers outside than only fetching the data the security model for this case has to made in the application level.

5.3 An Example Scenario of the Geospatial Cloud

To understand the solution lets take a generic example as shown in the following figure. This particular example shows a Cloud as a wrapper to the Geospatial Information System and the effect of the same to the existing system.

Geospatial data are huge in size and collected & maintained by different organizations depending on the attributes of the data e.g. layers, type, region. All these vendor/ organizations provide Data as a Service to the customers of them, which also known as WFS or Web Feature Service. To view the same information in a view-able format GIS has a Map Service known as WMS and another important service is the Web Processing Service (WPS) which deals with large amount of computation on the spatial data and produce desired information out of it.

Given the situation as above in general scenario all these organization are scattered over the web in different physical locations, which essentially means whenever any service is triggered a huge amount of data needs to be transferred from one host to another over the network . It can be easily seen by the problem statement now that the more it needs cascading of services the more bitter it will go to produce the end result and essentially that effects the end user. Moreover it gets worse with the increment in service requests in terms of computation capability and the network bandwidth limitation.

The model in the Figure 5.3 shows a scenario that fits the Cloud to the GIS and creates a solution to the above mentioned problems known as GIS Cloud. As discussed earlier the model SaaS over PaaS over IaaS is the exact solution in this particular situation. For a GIS central organization it needs to have the data repository as the central repository in the figure, but not all organization will surrender their data to the same so all together these organization includes with the central repository integrated with a cloud service creates the desires GIS cloud service. At this level this particular Cloud will provide the Infrastructure as a Service which can be availed by all the basic service repositories that means mainly the WFS providers. once this is set up, other business organizations like to offer WPS, WMS of other services to their end user customers will be availed infrastructure as their requirement from the cloud itself and can be coupled with their desired WFS or other requirements and will be ready to go.

Now from each business organization point of view they have the resources those are elastic as per the computation requirements, at this point only thing they need is to run their applications. So they need developers to create their web apps and deploy the same on the cloud. For this the Organization can provide the Platform as a Service on the infrastructure they have acquired to the developers to develop and run their applications. Now developers

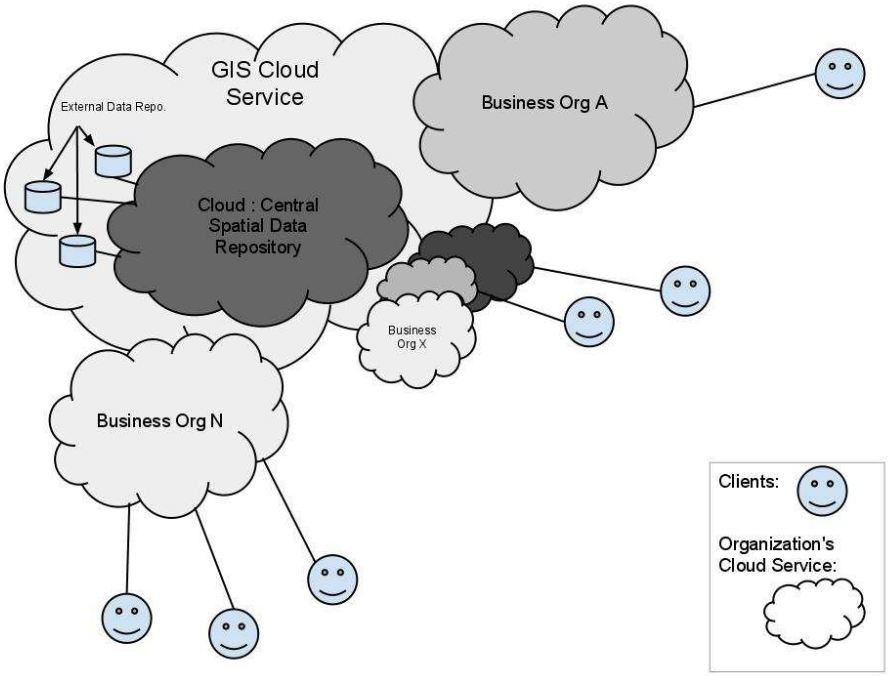


Figure 5.3: A Geospatial Cloud Representation

can now write their application in the form of Software as a Service and can deploy in there. At this point all the three models of deployment has been used one over another and the end result is each organization having their business running with no difference as the existing infrastructure but with lesser headache such as maintenance, similarly developers doesn't need to care about the infrastructure behind but to concentrate on the application development and of course the end users do not feel a difference at that level. So to make a gist of the situation following is the list of providers and consumers at different stage of this arrangement.

- IaaS (Generic Hardware)
  - Vendor: Cloud Service Provider(IITkgp Geospatial Cloud)
  - Client: Business Organization
- PaaS (Operating System/ Runtimes)
  - Vendor: Business Organization
  - Client: Developers
- SaaS (Application-web apps/geospatial apps)
  - Vendor: Developers
  - Client: Customer Base

Now let's have a look at the bright sides of this arrangement:

- All the data are maintained by a central maintenance
- All end user request are streaming through the front end and actually served at the central Cloud
- As all the requests are served at a single point so not much moving of large data is required
- Computation Power as a Service gives the flexibility to large computation whatsoever
- All similar requests are streaming at a single point leads to better routing

5.4 Some Screenshots of the GIS CLoud

Following are some three screen captures from the configured system. Figure 5.4 shows the system behavior model discussed in subsection 5.2.1 by running a WFS server in side the Geospatial Cloud. The Figure 5.5 shows system behavior model discussed in subsection 5.2.2 by getting a map of the saltlake blocks from an outside WFS server and generating the map in the cloud. The last Figure 5.6 shows a web service client side interface for a WMS service. Other Screenshots for the system can be Found in the Appendix B

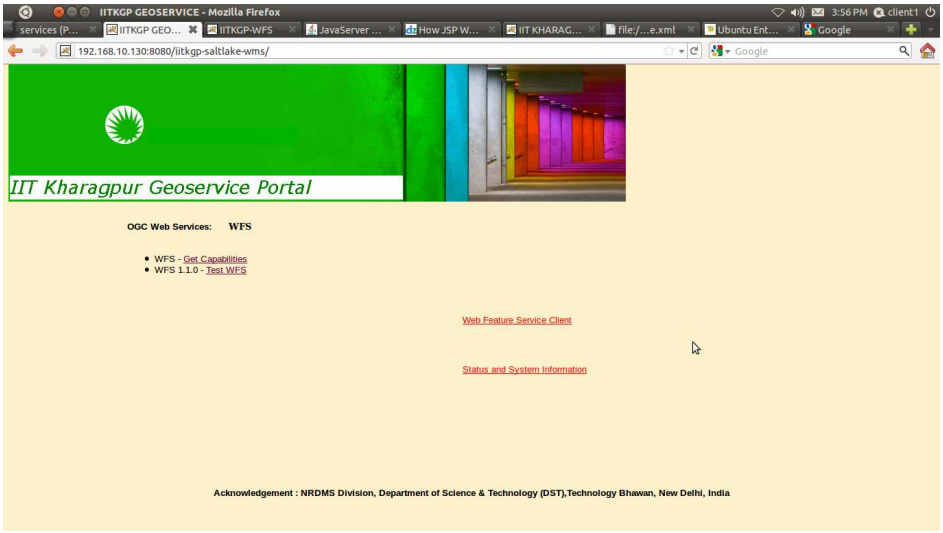


Figure 5.4: A WFS server running in the Cloud

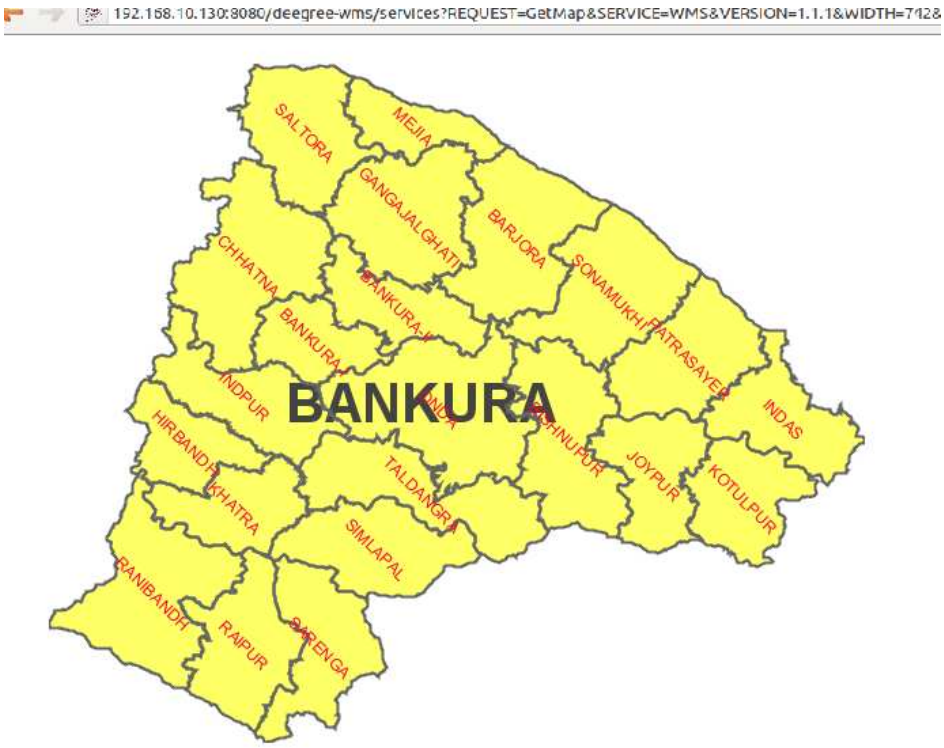


Figure 5.5: A Map Generated in the Cloud from Spatial Data fetched from an Outside WFS

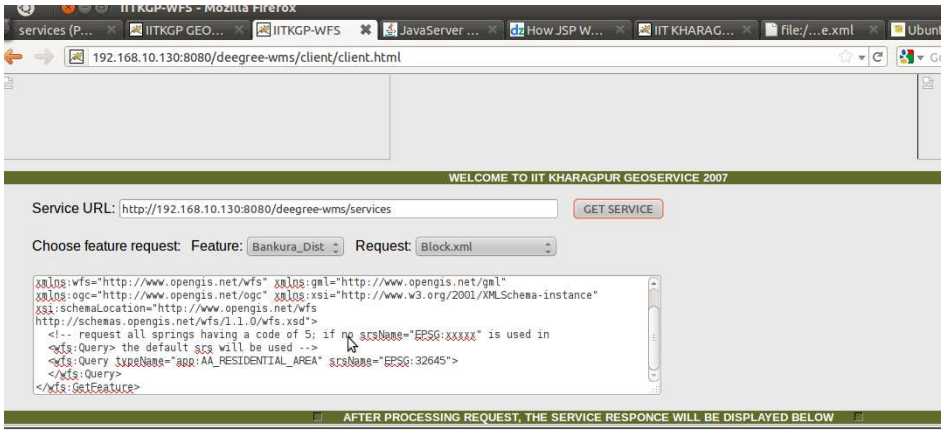


Figure 5.6: A WMS Client running in the cloud [SaaS]

Chapter 6

Performance Evaluation

This chapter is mainly presents the behavior of the system according to the set up. Eucalyptus provides different types of preset infrastructure types for the users of the system to run their images on. In this chapter we will see how each of the system behaves when they are under pressure and accordingly a prior recommendation is will be given to the users before choosing the infrastructure type with respect to their usage requirements.

6.1 Description of the Test

An Infrastructure as a Service cloud provides mainly virtual machines that are configured in terms of number of CPU core i.e. processing power and amount of Memory i.e. RAM. Now in out system setup there are 5 preset images. So we considered running an web application on those different type of virtual machines and putting load on the same and at the same time calculating the corresponding CPU and Memory utilization for the VMs. Here are the description of the systems and the tools used to calculate the values. In the next section we put those values in to graph for having a visual understanding.

6.1.1 Types of Images in Cloud Setup

Before start the discussion first have a look at the available infrastructure in Table 6.1.1 in our set up (which is customizable). For each of the instance we have loaded same pre-customized Operating system and create the same workload for all the instances. Details of the Workload and the measurement of the corresponding CPU and Memory utilization can be found in the Appendix A.

6.1.2 Tools Used

In this subsection lets get little familiarized to the tools used to get this experiment done. We needed to produce a Workload, mainly huge amount of http requests and at the same time the corresponding CPU and Memory utilization was measured.To generate the workload

Table 6.1: Type of Available Infrastructures

Type	Number of CPU	RAM	Hard Disk
m1.small	1	192	2
c1.medium	1	256	5
m1.large	2	512	10
m1.xlarge	2	1024	20
c1.xlarge	4	2040	20

from the client side a server benchmarking tool from apache-utils was used namely 'ab'. Tools for measuring these are 'ss' (an advanced version of network monitoring tool 'netstat') which was used for calculating the amount of load on server at any point of time, 'iostat' which reports Central Processing Unit (CPU) statistics and finally 'free' which displays amount of free and used memory in the system used for finding the memory utilization. Details of the used commands can be found in Appendix A. And finally after all the data was gathered 'Matlab' was used for generating the graphs that are in the following sections.

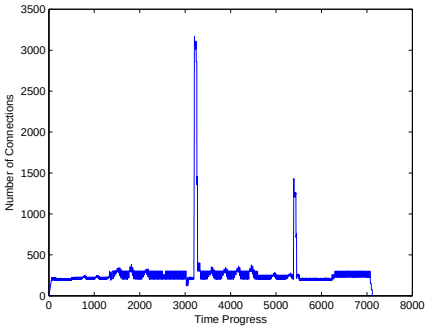
6.2 Behavior of the Systems

This section shows the results we got after running the Workload over each type of available infrastructure(in our implementation) on the Eucalyptus. The workload is around 4.9 lacks of http request generated at different concurrency level by the benchmarking tool provided by apache-utils called 'ab' (discussed earlier in 6.1.2), of which each request in turn calls a python code written in the server to calculate prime number upto 10000 which is basically giving load to the server processing. Following is the results of the same Workload on each type of infrastructure, each of which running same image (Customized image of Ubuntu Server v11.04 release - Natty, discussed in 4.4).

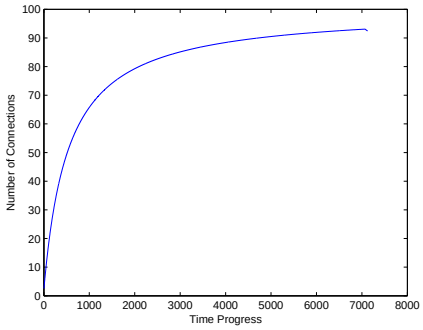
6.2.1 Scenario 1: [m1.small]

This particular infrastructure known to Eucalyptus as 'm1.small' type of infrastructure is actually the lowest configuration machine available in our implementation. This particular virtual machine consists of 1 CPU core, 192 MB of Random Access Memory and only 1 GB of hard-disk. Following is the result of the Workload running on this VM. Figure (a) shows the loads pattern, Figure (b) shows the corresponding CPU utilization and Figure (c) shows the corresponding Memory Utilization. And at last Figure 6.2 shows all these graphs together in a normalized scale of 0 to 1.

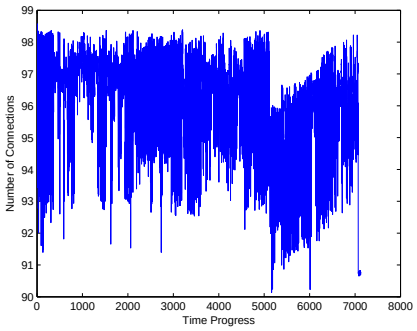
From Figure (a) we can see the characteristics of the load that has been given to the m1.small type of instance, as per the x-axis in this graph shows the progress in time(in 1 second per unit), it clearly shows that the time consumed to complete the workload has taken around 7000 seconds. In this particular instance the amount of RAM is 192 MB, and



(a): Workload: Number of Connections



(b): CPU Utilization



(c): Memory Utilization

Figure 6.1: Test Results with 1 CPU, 192 MB RAM

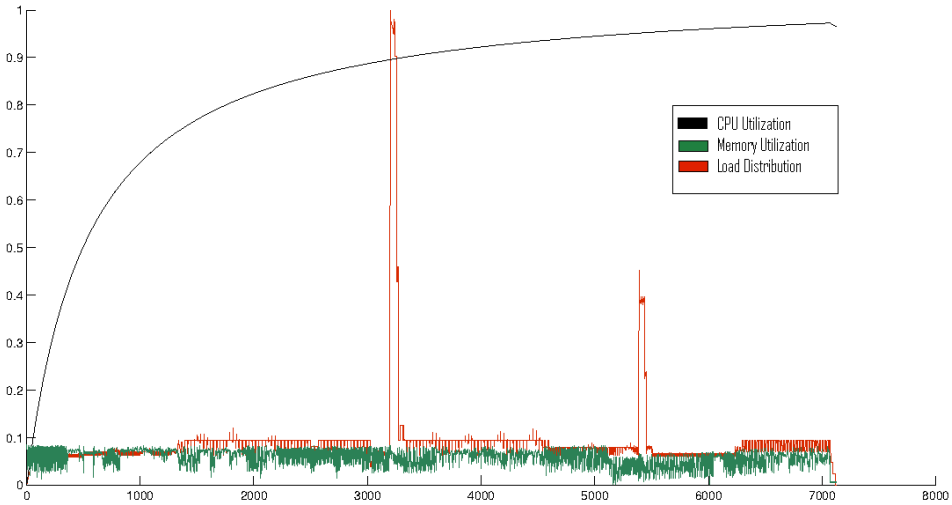


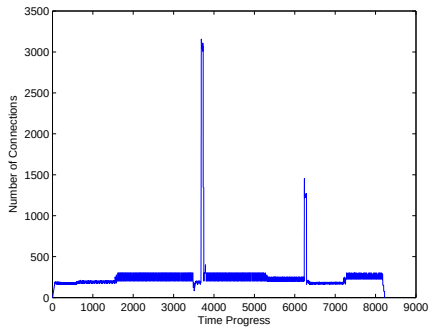
Figure 6.2: All the values are normalized in the range 0 to 1 [m1.small]

in Figure (b) shows the maximum CPU utilization is around 90%. So it can be inferred in

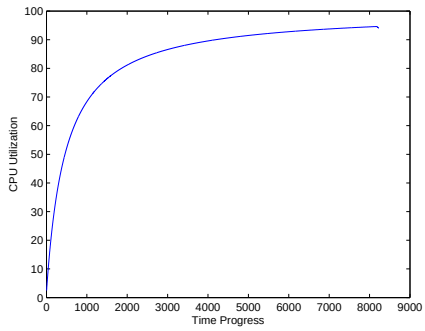
this case that with reference to the memory utilization which is around 100% in Figure (c) is that the load was so high for the amount of free RAM and the available processing power that it took really long time to serve the workload. So to conclude this particular type of infrastructure is suitable for small sized Operating System and basically personal usage of the same. Figure 6.2 gives a overall picture of all these graphs in a normalized value of 0 to 1, which shows the CPU utilization is gradually increasing with respect to the continuous load to the server.

6.2.2 Scenario 2: [c1.medium]

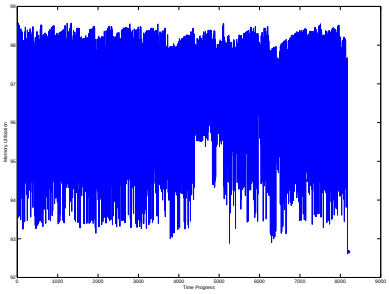
This particular infrastructure known to Eucalyptus as 'c1.medium' type of infrastructure is slightly powerful than m1.small VM. This particular virtual machine consists of 1 CPU core, 256 MB of Random Access Memory and only 5 GB of hard-disk. Following is the result of the Workload running on this VM. Figure (a) shows the loads pattern, Figure (b) shows the corresponding CPU utilization and Figure (c) shows the corresponding Memory Utilization. And at last Figure 6.4 shows all these graphs together in a normalized scale of 0 to 1.



(a): Workload: Number of Connections



(b): CPU Utilization



(c): Memory Utilization

Figure 6.3: Test Results with 1 CPU, 256 MB RAM

From Figure (a) we can see the characteristics of the load that has been given to the c1.medium type of instance, as per the x-axis in this graph shows the progress in time(in 1

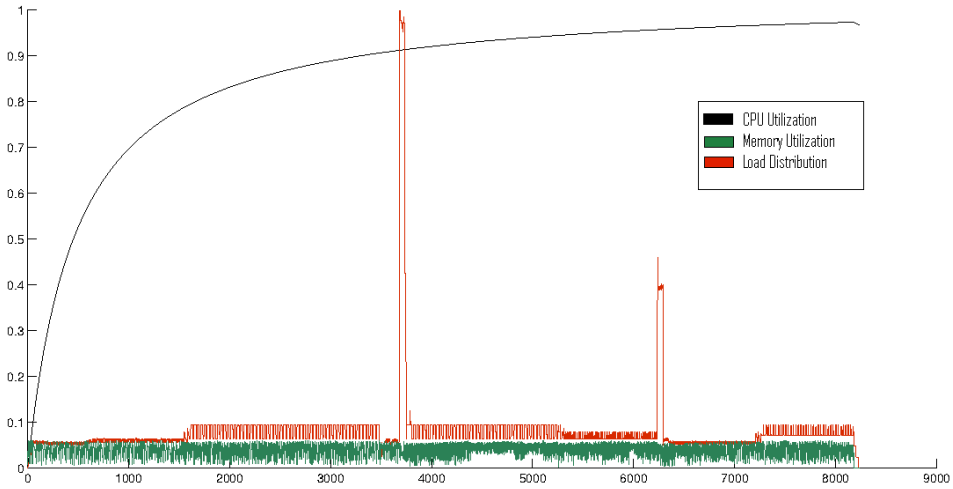


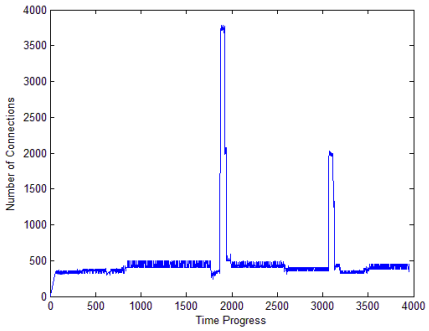
Figure 6.4: All the values are normalized in the range 0 to 1 [c1.medium]

second per unit) which is again the same as previous instance type. In this particular instance the amount of RAM is 256 MB, and in Figure (b) shows the maximum CPU utilization is again around 90% and the memory utilization is also similar in this case. So it can be inferred in this case that with reference to the memory utilization as in Figure (c) is that as the increment of the size of the RAM is not significant the CPU utilization and the time taken for the instance to serve the requests is almost the same as before. So to conclude this particular type of infrastructure is also suitable for small sized Operating System and again recommended for personal use of the user. Figure 6.4 gives a overall picture of all these graphs in a normalized value of 0 to 1, which shows the CPU utilization is gradually increasing with respect to the continuous load to the server.

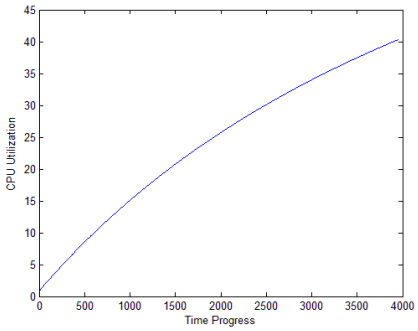
6.2.3 Scenario 3: [m1.large]

This particular infrastructure known to Eucalyptus as 'm1.large' type of infrastructure is the first multicore virtual machine in our case. This particular virtual machine consists of 2 CPU core, 512 MB of Random Access Memory and only 10 GB of hard-disk. Following is the result of the Workload running on this VM. Figure (a) shows the loads pattern, Figure (b) shows the corresponding CPU utilization and Figure (c) shows the corresponding Memory Utilization. And at last Figure 6.6 shows all these graphs together in a normalized scale of 0 to 1.

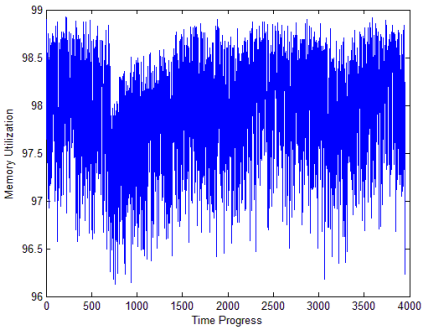
From Figure (a) we can see the characteristics of the load that has been given to the m1.large type of instance, as per the x-axis in this graph shows the progress in time(1 second unit) which in this case shows a significant amount of decrease in time to be concrete in numbers it is around 4000 seconds. In this particular instance the amount of RAM is 512 MB, double as compared to the c1.medium type, and in Figure (b) shows the maximum



(a): Workload: Number of Connections



(b): CPU Utilization



(c): Memory Utilization

Figure 6.5: Test Results with 2 CPU, 512 MB RAM

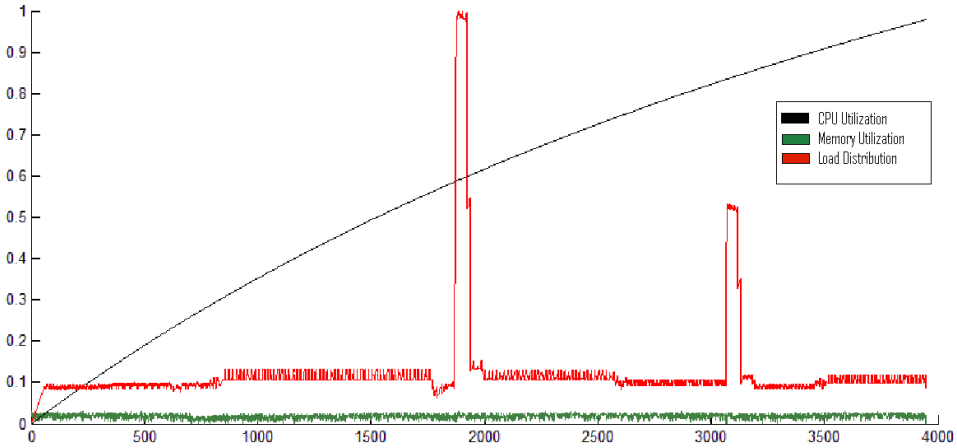


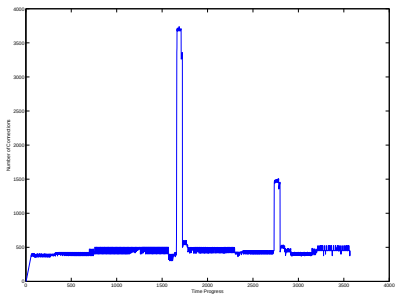
Figure 6.6: All the values are normalized in the range 0 to 1 [m1.large]

CPU utilization is around 40% as there is 2 CPU cores in this case but at the same time the memory utilization as in Figure (c) shows around 100% of memory utilization which as compared to previous types is better and thus it can be inferred in this case that as the

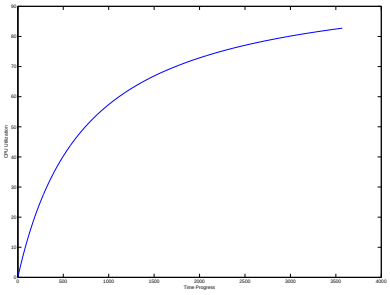
size of the RAM increases as well as the number of CPU the same amount of processing takes lesser time and also it gives significant amount of processing power to spare. So to conclude this particular type of infrastructure is quite suitable for small and medium sized Operating System and again recommended for personal use of the user as well as small or medium sized web application to deploy. Figure 6.4 gives a overall picture of all these graphs in a normalized value of 0 to 1, which shows the CPU utilization is gradually increasing with respect to the continuous load to the server.

6.2.4 Scenario 4: [m1.xlarge]

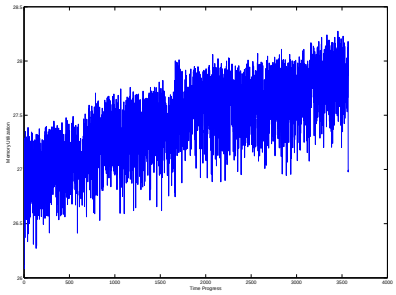
This particular infrastructure known to Eucalyptus as 'm1.xlarge' type of infrastructure is more voluminous in terms of hard-disk as well as Random Access Memory. This particular virtual machine consists of 2 CPU core, 1024 MB of Random Access Memory and only 20 GB of hard-disk. Following is the result of the Workload running on this VM. Figure (a) shows the loads pattern, Figure (b) shows the corresponding CPU utilization and Figure (c) shows the corresponding Memory Utilization. And at last Figure 6.8 shows all these graphs together in a normalized scale of 0 to 1.



(a): Workload: Number of Connections



(b): CPU Utilization



(c): Memory Utilization

Figure 6.7: Test Results with 2 CPU, 1024 MB RAM

From Figure (a) we can see the characteristics of the load that has been given to the m1.xlarge type of instance, as per the x-axis in this graph shows the progress in time(1

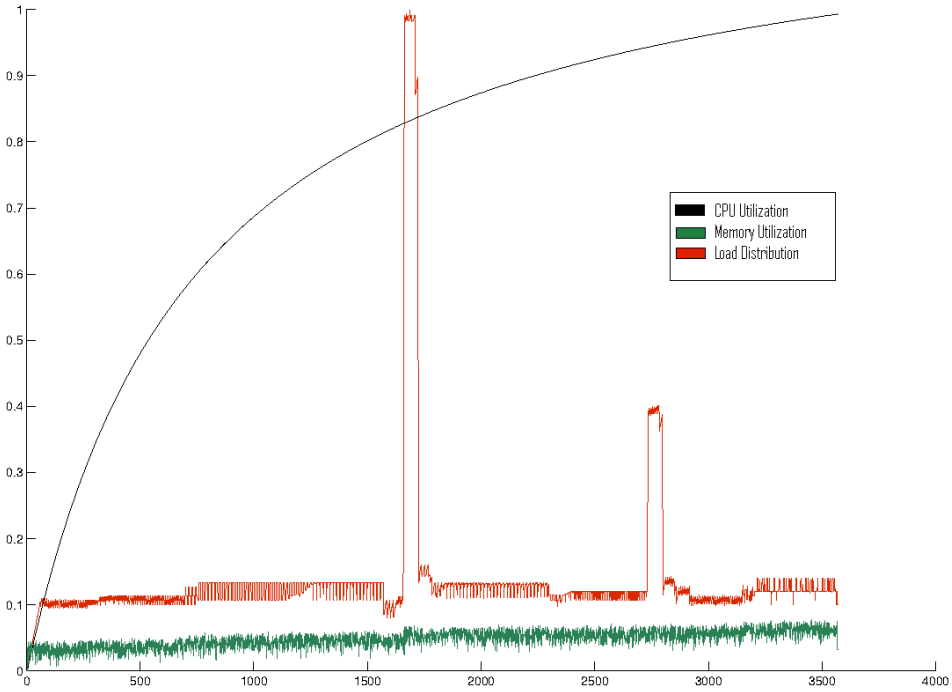


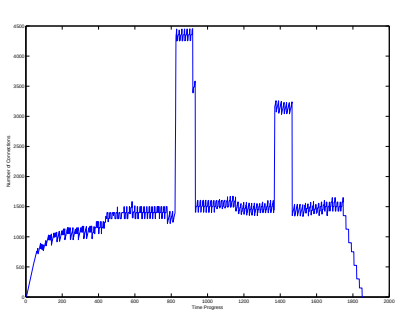
Figure 6.8: All the values are normalized in the range 0 to 1 [m1.xlarge]

second unit), it clearly shows that the time consumed to complete the workload has taken around 3500 seconds. In this particular instance the amount of RAM is 1024 MB, double as compared to the m1.large type, and in Figure (b) shows the maximum CPU utilization is around 80% as there is 2 CPU cores in this case but at the same time the memory utilization as in Figure (c) shows around 30% which as compared to previous types is much low and thus it can be inferred in this case that as the size of the RAM increases but the number of CPU remains same the processing became faster as the amount of free RAM is quite large and the amount of processing power utilization is quite high. So to conclude this particular type of infrastructure is quite suitable for medium and large sized Operating System and again recommended for small or medium sized web application with quite high load possibility to deploy. Figure 6.4 gives a overall picture of all these graphs in a normalized value of 0 to 1, which shows the CPU and memory utilization is gradually increasing with respect to the continuous load to the server.

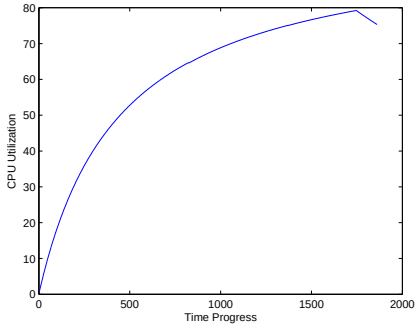
6.2.5 Scenario 5: [c1.xlarge]

This particular infrastructure is the last and most powerful VM in our implementation and it is known to Eucalyptus as 'c1.xlarge' type of VM. This VM surpasses earlier types by number of CPU cores and the amount of Random Access Memory. This particular virtual

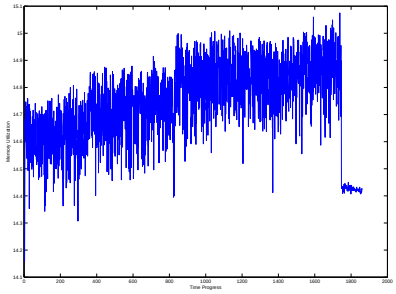
machine consists of 4 CPU core, 2040 MB of Random Access Memory and only 20 GB of hard-disk. Following is the result of the Workload running on this VM. Figure (a) shows the loads pattern, Figure (b) shows the corresponding CPU utilization and Figure (c) shows the corresponding Memory Utilization. And at last Figure 6.2 shows all these graphs together in a normalized scale of 0 to 1.



(a): Workload: Number of Connections



(b): CPU Utilization



(c): Memory Utilization

Figure 6.9: Test Results with 4 CPU, 2040 MB RAM

From Figure (a) we can see the characteristics of the load that has been given to the c1.xlarge type of instance, as per the x-axis in this graph shows the progress in time(1 second unit), it clearly shows that the time consumed to serve the workload has taken around 1700 seconds. In this particular instance the amount of RAM is 2040 MB and the number of CPU cores is 4, double as compared to the m1.xlarge type, and in Figure (b) shows the maximum CPU utilization is around 80%at the same time the memory utilization as in Figure (c) shows around 15% which as compared to previous types is much less and thus it can be inferred in this case that as the size of the RAM increases in huge amount as well as the number of CPU the processing became much faster as the amount of free RAM is quite large and the amount of processing power utilization is quite high. So to conclude this particular type of infrastructure is quite suitable for large sized Operating System and recommended for medium and large sized web application with really high load possibility to deploy. Figure

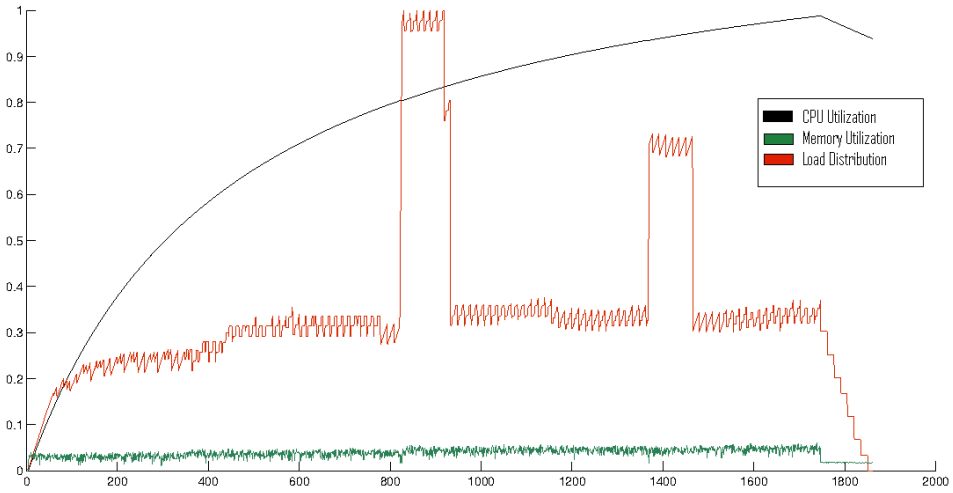


Figure 6.10: All the values are normalized in the range 0 to 1 [c1.xlarge]

6.10 gives a overall picture of all these graphs in a normalized value of 0 to 1, which shows the CPU utilization is gradually increasing whereas the memory utilization remains constant with respect to the continuous load to the server.

6.3 Comparison

In this section each of the Figure 6.11 & Figure 6.12 respectively shows a consolidated graph of the previously seen CPU and Memory performances for all types of Instances altogether, where x-axis is the time one instance took to complete the same amount of load and y-axis shows the corresponding CPU and Memory utilization in two different figures.

6.4 Discussion

In this chapter we have presented the different types of infrastructure available in our setup and for each of them we run one instance in the cloud with same configuration, then for a similar load we studied the behaviour of the systems and with that knowledge we can now give recommendation for the type of usage each of the type is suitable for. To summarize the recommendations are as follows, the small and medium type of the instances are suitable for personal usage of the users but not for usage as a highly loaded server. Next is the large type of image which will be allowable for personal use of the user as well as small or medium sized web application to deploy with moderate amount of load. Last two types are the extra large type of infrastructures, type 1 which are recommended for small or medium sized web application with quite high load possibility to deploy and the type 2 is recommended for medium and large sized web application with really high load possibility to deploy.

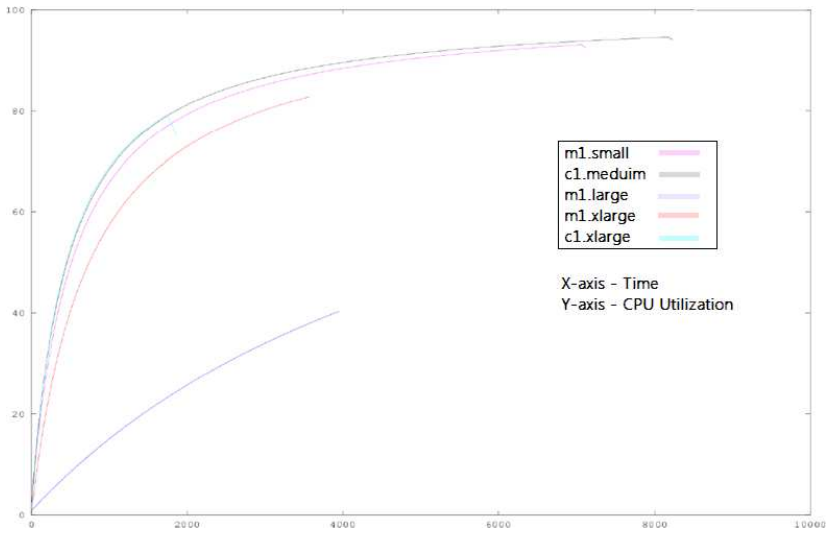


Figure 6.11: Comparing the CPU utilization for all types

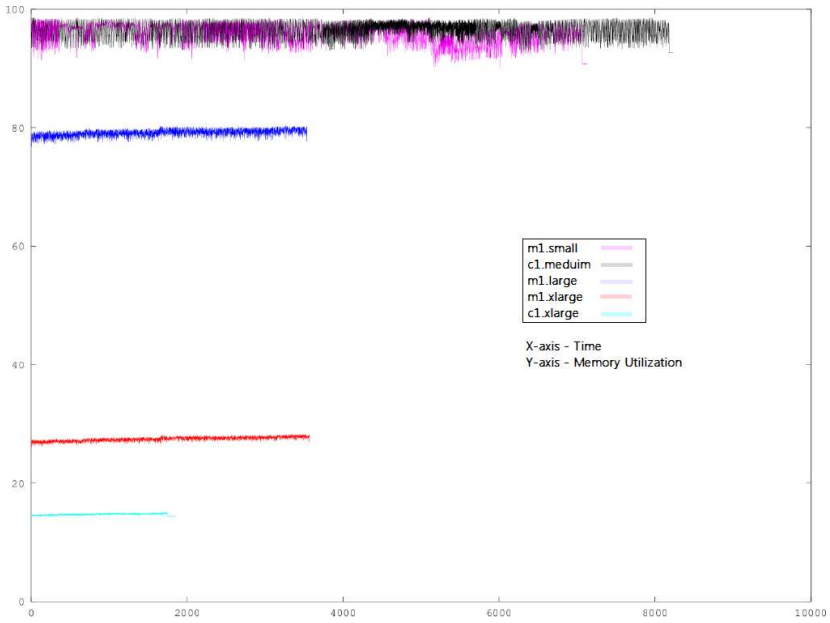


Figure 6.12: Comparing the Memory utilization for all types

Chapter 7

Conclusion & Future Work

This is the final Chapter of the thesis which concludes the work by the following two sections. In the first section let us summarize the work accomplished during this thesis work and in the following and last section some of possible future work direction on the same are given.

7.1 Contribution

In this thesis we the main keywords are Cloud Computing and Geographical Information System, the idea was to explore the possibilities how these two can be integrated and what will be the effect of that altogether. So a model in Section 3.2 was proposed showing how a business organization can shift itself to the cloud and a problem modeling following to that. Then in the project we explored many Open source cloud systems available as described in Chapter 2. Thereafter we selected Eucalyptus to proceed with and the setting up of the same was done, this creates the base of the proposed model. Now the second level needed to be set up, for which we choose Java Platform provided by apache and created web portal so that the customers at that level gets the ability to just open and upload their application right there to finalize the last level of the model. Thus the whole model was realized as proposed. Now it was time for geospatial services to come into the picture. For this we used a previously prepared set of Geospatial Services from the Enterprise-GIS Framework and tried to figure out different scenarios of deployment case, by end the that we came up with two very distinctive model of deployment described in Subsection 5.2.1 and Subsection 5.2.2. Now that these models are prepared, the realization of those are needed to be done which was shown in Section 5.4. And then an example scenario and its description for the geospatial cloud was given in the Section 5.3. Lastly we evaluated some performance measures in terms of system utilization which was described in Chapter 6. Thus the goal for this project was achieved but there are more possibilities and chances of improvement in the current setup which are discussed in the following section after a small conclusion in the following paragraph.

The idea of putting everything into the cloud, in this case the whole GIS gives the big

picture of the *GIS-cloud or GIS-in-the-cloud*. As described earlier in this thesis the Cloud provides the infrastructure to the owners. So that way the main GIS service providers will deploy their data service over the cloud in one way or another. At the same time using the Platform as a service paradigm, access will be given to developers to design and upload custom application for the organization itself which uses the data provided the the data service from single or multiple organizations from the cloud. Now the applications being the web service based Software as a Service will provide the final wrapping over the Cloud. At this point we can see that the whole GIS system has been deployed in the cloud in this manner, thus our goal succeeds as *GIS-in-the-cloud* or can be abbreviated as *GIS Cloud.*

7.2 Future Directions

This is a huge and very new are of development, so clearly it has really variant opportunities to work on. Here are some of the numerous possible ways of the same.

7.2.1 Developing Load-balancer

This is a requirement for the system now. As system automation is possible for cloud to dynamically scale up and scale down, as far as infrastructure is concerned in this particular case a good load balancer is required to distribute the forthcoming loads and same goes for the system scale down too.

7.2.2 Improvement of the Platform as a Service model

This is one level up to the abstraction level. This particular service can be improved by lot e.g. provision of multiple runtimes such as python, provision for different database connectivity as per requirement etc. thus making the platform more and more rich in terms of flexibility.

7.2.3 Security Aspects

For web based access to the system the security for accessing resources and the instances are critical. Good amount of work needed to be done in this aspect.

PaaS level Security should be another main concern in this case. As the same database management software and runtimes are shared among different users, so the inherent vulnerabilities are there and also multi-tenant access to these platform can create security breach from one user to another while sharing the data among themselves.

In the lowest level that is when the virtual machines are created, as the resource are being shared from a singular or shared device and also these resources are getting reused, so proper analysis need to be made to ensure that this type of sharing has no threat.

7.2.4 Standardization of Metrics

This is the most important among all the future works that needed to be done first. As standardizing the metrics can be very useful to measure the performance of the system as well as it can be used to calculate the usage of the system. Thus the load balancing aspect get more precise direction and most importantly the billing can be done in much more standard way that will be convenient to the users.

Appendix A

Codes and Scripts

Following is the Python Code used (Modified code in reference to Python-paramiko [27] package) to SSH in an IP and maintain the session:

```
"""Friendly Python SSH2 interface."""

import os
import tempfile
import paramiko

class Connection(object):
    """Connects and logs into the specified hostname.
    Arguments that are not given are guessed from the environment."""

    def __init__(self,
                 host,
                 username = None,
                 private_key = None,
                 password = None,
                 port = 22,
                 ):
        self._sftp_live = False
        self._sftp = None
        if not username:
            username = os.environ['LOGNAME']

        # Log to a temporary file.
        templog = tempfile.mkstemp('.txt', 'ssh-')[1]
        paramiko.util.log_to_file(templog)

        # Begin the SSH transport.
        self._transport = paramiko.Transport((host, port))
        self._tranport_live = True
        # Authenticate the transport.
        if password:
```

```
33              # Using Password.
                self._transport.connect(username = username, password = password)
35          else:
                # Use Private Key.
37              if not private_key:
                    # Try to use default key.
39                  if os.path.exists(os.path.expanduser('~/.ssh/id_rsa')):
                        private_key = '~/.ssh/id_rsa'
41                  elif os.path.exists(os.path.expanduser('~/.ssh/id_dsa')):
                        private_key = '~/.ssh/id_dsa'
43                  else:
                        raise TypeError, "You have not specified a password or key.
                            "
45
                private_key_file = os.path.expanduser(private_key)
47              rsa_key = paramiko.RSAKey.from_private_key_file(private_key_file)
                self._transport.connect(username = username, pkey = rsa_key)
49
    def _sftp_connect(self):
51      """Establish the SFTP connection."""
        if not self._sftp_live:
53          self._sftp = paramiko.SFTPClient.from_transport(self._transport)
            self._sftp_live = True
55
    def get(self, remotepath, localpath = None):
57      """Copies a file between the remote host and the local host."""
        if not localpath:
59          localpath = os.path.split(remotepath)[1]
        self._sftp_connect()
61      self._sftp.get(remotepath, localpath)

63  def put(self, localpath, remotepath = None):
        """Copies a file between the local host and the remote host."""
65      if not remotepath:
            remotepath = os.path.split(localpath)[1]
67      self._sftp_connect()
        self._sftp.put(localpath, remotepath)
69
    def execute(self, command):
71      """Execute the given commands on a remote machine."""
        channel = self._transport.open_session()
73      channel.exec_command(command)
        output = channel.makefile('rb', -1).readlines()
75      if output:
            return output
77      else:
            return channel.makefile_stderr('rb', -1).readlines()
79
    def close(self):
```

```
        """Closes the connection and cleans up."""
        # Close SFTP Connection.
        if self._sftp_live:
            self._sftp.close()
            self._sftp_live = False
        # Close the SSH Transport.
        if self._tranport_live:
            self._transport.close()
            self._tranport_live = False

    def __del__(self):
        """Attempt to clean up if not explicitly closed."""
        self.close()

def main():
    """Little test when called directly."""
    # Set these to your own details.
    myssh = Connection('10.14.99.116', username = 'root', password = 'root!@#$'
        )
#    myssh.put('ssh.py')
    a=myssh.execute('ls')
    for b in a:
  print b
    myssh.close()

# start the ball rolling.
if __name__ == "__main__":
    main()
```

Some of the important Shell Scripts written :

Show all the IP address pf Running Instances:

```
#SHOWS ALL THE IP ADDRESSES

#! /bin/bash
. /root/.euca/eucarc
euca-describe-instances |grep running|tr '\t' ' '|cut -d ' ' -f4
```

Shows the available keypairs :

```
#SHOWS ALL THE AVAVILABLE KEYPAIRS

#! /bin/bash
. /root/.euca/eucarc
euca-describe-keypairs |tr '\t' ' '|cut -d ' ' -f2
```

Gives the machine IDs corresponding to the name if the images:

```
#SHOWS ONLY MACHINE IMAGES IDS AND CORRESPONDING NAMES OF THE IMAGES

#!/bin/bash
. /root/.euca/eucarc
euca-describe-images|grep emi|tr -s ' \t' ':'|tr ':' ' '|cut -d ' ' -f 2,3
```

Following is a code written for multiple purpose :

```
# SHOWS THE FOLLOWING DEPENDING ON THE ARGUMENT
# instance pub-ip : CORRESPONDING INSTANCE-ID
# image pub-ip : CORRESPONDING MACHINE IMAGE-ID
# instance-id : CORRESPODING PUBLIC IP
# private pub-ip : CORRESPONDING PRIVATE IP
# status instance-id : STATUS OF INSTANCE-ID

#!/bin/bash
. /root/.euca/eucarc

if [ "$1" == "instance" ]
then
euca-describe-instances |grep INSTANCE |grep $2 |tr '\t' ' '|cut -d ' ' -f2

else if [ "$1" == "image" ]
  then
  euca-describe-instances |grep INSTANCE |grep $2 |tr '\t' ' '|cut -d ' ' -f3
  else if [[ "$1" == i-* ]]
    then
    euca-describe-instances |grep INSTANCE |grep $1 |tr '\t' ' '|cut -d ' ' -f4
    else if [ "$1" == "private" ]
      then
      euca-describe-instances |grep INSTANCE |grep $2 |tr '\t' ' '|cut -d ' ' -
          f5
      else if [ "$1" == "status" ]
        then
        euca-describe-instances |grep INSTANCE |grep $2 |tr '\t' ' '|cut -d ' '
            -f6
          else
          printf "\nUSAGE :: info [option] [option]\n\n\tOPTIONS::\n\n\
              tinstance pub-ip \n\timage pub-ip \n\tinstance-id\n\tprivate pub-
              ip\n\tstatus instance-id\n\thelp\n\n"
        fi
      fi
    fi
```

```
32    fi
fi
```

Shows the current instances status in all states, no filtering :

```
# SHOWS SPECIFIC INSTANCE INFORMATION
2
#!/bin/bash
4 . /root/.euca/eucarc
euca-describe-instances|grep emi|tr -s ' \t' ':'|tr ':' ' '|cut -d ' ' -f
    2,3,4,5,6,9,7
```

This code is used for terminating an instance :

```
#TO KILL AN INSTANCE
2
#!/bin/bash
4 # instnce-id  type-of-image key
. /root/.euca/eucarc
6 echo "euca-terminate-instances $1"
euca-terminate-instances $1
8
printf "instance $1 Terminated"
```

This code basically lists the components of the cloud and their IP :

```
#SHOWS ALL THE COMPONENT INFORMATION
2
#!/bin/bash
4 . /root/.euca/eucarc
euca_conf --list-walruses
6 euca_conf --list-clusters
euca_conf --list-scs
8 euca_conf --list-nodes
```

Runs an Instance and monitors the same until the status reached running :

```
1 #RUNNING AN INSTANCE

3 #!/bin/bash
# instnce-id  type-of-image key
5 . /root/.euca/eucarc
echo "euca-run-instances $1 -t $2 -k $3 "
7 euca-run-instances $1 -t $2 -k $3 |grep "INSTANCE" |tr '\t' ' ' | cut -d ' ' -
    f2 >_at
```

```
inst_id=`cat _at`
#rm _at
cat _at

count=1
while :
do
sleep 2
/home/gis-clc/pys/info.sh $inst_id > _at
cat _at
  if [ $count -gt 1 ]
  then
    break
  fi
count=`expr $count + 1`
done

count=1
pub_ip=`cat _at`
echo "" > _at
while :
do
sleep 4
/home/gis-clc/pys/info.sh status $inst_id > _at
cat _at
state=`cat _at`
#echo $pub_ip
#echo $state

  if [ "$state" == "running" ]
  then
    break
  fi
# if [ $count -gt 10 ]
# then
#   break
# fi
#count=`expr $count + 1`

done
printf "\n\ninstance $inst_id with IP  $pub_ip is now running\n"
printf "configure $pub_ip now\n"
#./configure-inst.sh $ip
```

Python Code Running As load in server in the back ground is following :

```
def primes(n):
        if n==2: return [2]
```

```
        elif n<2: return []
        s=range(3,n+1,2)
        mroot = n ** 0.5
        half=(n+1)/2-1
        i=0
        m=3
        while m <= mroot:
                if s[i]:
                        j=(m*m-3)/2
                        s[j]=0
                        while j<half:
                                s[j]=0
                                j+=m
                i=i+1
                m=2*i+3
        return [2]+[x for x in s if x]
print primes(10000)
```

Simple PHP code to cal the Python script :

```
<html>
    <head>
    </head>
    <body>
      <?php
  output=`/usr/bin/python /var/www/calculate.py`;
  echo $output;
      ?>
    </body>
</html>
```

Commands used for calculating cpu usage, memory usage and number of connection at any point of time are following:

```
Number of Connection :   ss -an |tr -s '\t '|cut -d ' ' -f 4|grep 80$|wc -l
CPU Utilization : iostat|grep '^ '|tr -s '\t '|cut -d ' ' -f7
Memory Utilization : free|grep Mem|tr -s  '\t '|cut -d ' ' -f2,3
```

The Workload is basically continuous request to the web server which are accomplished by a benchmarking tool called 'ab' provided by the apache-utils package, the sequence is as follows:

```
ab -k -n 10000 -c 10 http://<Ip_address>/index.php
ab -k -n 20000 -c 5 http://<Ip_address>/index.php
ab -k -n 50000 -c 25 http://<Ip_address>/index.php
```

```
ab -k -n 100000 -c 100  http://<Ip_address>/index.php
ab -k -n 10000 -c 20  http://<Ip_address>/index.php
ab -k -n 50000 -c 150  http://<Ip_address>/index.php
ab -k -n 80000 -c 100  http://<Ip_address>/index.php
ab -k -n 50000 -c 50  http://<Ip_address>/index.php
ab -k -n 20000 -c 200  http://<Ip_address>/index.php
ab -k -n 50000 -c 10  http://<Ip_address>/index.php
ab -k -n 50000 -c 75  http://<Ip_address>/index.php
```

Appendix B

Extra Screenshots

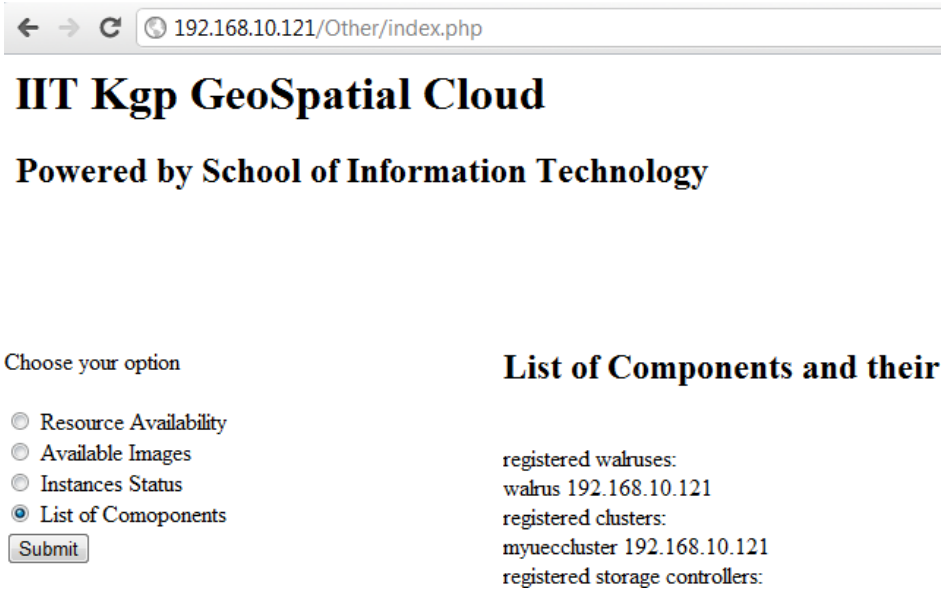


Figure B.1: Geospatial Cloud on the Web : Homepage

```
root@gis-clc:~# euca-describe-instances
RESERVATION     r-4DC8088D      admin   default
INSTANCE        i-4E2509A4      emi-873C1699    192.168.10.128  172.1.0.130     running hello   0               m1.small        2011-09-01T12:29:25.66
yueccluster     eki-F71E1868    eri-D81F17E1
RESERVATION     r-4D2E0921      admin   default
INSTANCE        i-451907E9      emi-D89F151C    192.168.10.130  172.1.0.132     running mykey   0               m1.small        2011-08-27T16:41:23.83
yueccluster     eki-23541658
RESERVATION     r-406D0712      admin   default
INSTANCE        i-4EE50828      emi-873C1699    192.168.10.134  172.1.0.136     running mykey   0               m1.xlarge       2011-09-01T17:30:44.82
yueccluster     eki-F71E1868    eri-D81F17E1
RESERVATION     r-3BFE078C      admin   default
INSTANCE        i-487B07B3      emi-04E9112F    192.168.10.133  172.1.0.135     running mykey   0               m1.xlarge       2011-09-01T17:29:48.55
cluster eki-63AF12DB    eri-9D9213B5
RESERVATION     r-3BF20650      admin   default
INSTANCE        i-4B4B0828      emi-11E415E5    192.168.10.131  172.1.0.133     running mykey   0               m1.small        2011-08-27T16:52:22.69
yueccluster     eki-5F19172C
RESERVATION     r-4B030901      admin   default
INSTANCE        i-4600084F      emi-C9D81014    192.168.10.129  172.1.0.131     running mykey   0               m1.xlarge       2011-09-01T17:17:52.61
yueccluster     eki-49701262    eri-8242134D
RESERVATION     r-40050775      admin   default
INSTANCE        i-387A07B6      emi-B0B91857    192.168.10.132  172.1.0.134     running mykey   0               c1.medium       2011-08-27T18:27:06.14
yueccluster     eki-0523199B
```

Figure B.2: List of Running Instances at Certain Point of Time

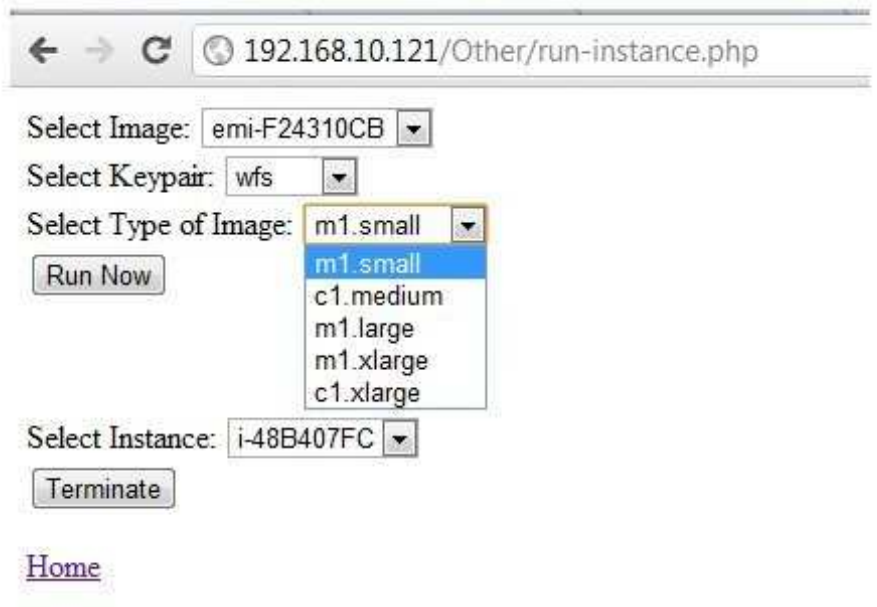


Figure B.3: Run or Terminate instances from the web

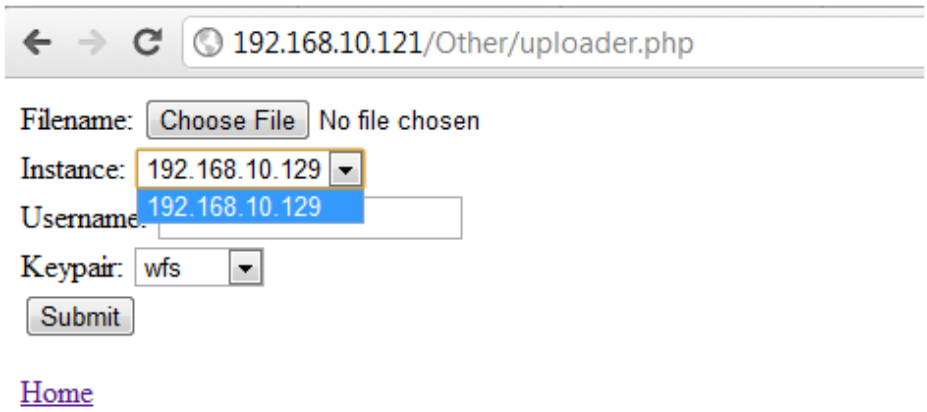


Figure B.4: Upload your application in Just a Click : PaaS Portal

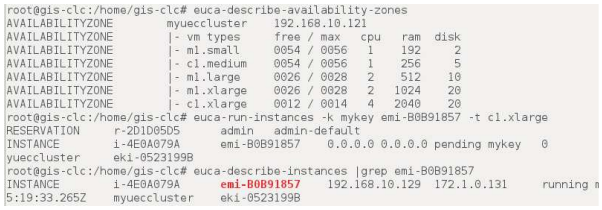


Figure B.5: Launch an Instance and it will run

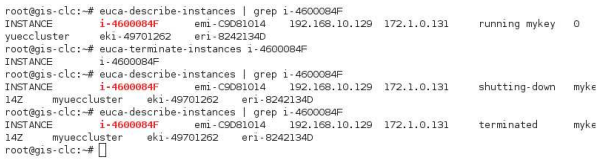


Figure B.6: From running – shutting-down – terminated

Glossary

3Tera	:	Tera, http://www.3tera.com/
AppScale	:	An Open source project that emulates GAE and provides PaaS
AWS	:	Amazon Web Service
Azure	:	Microsoft Azure, http://www.microsoft.com/windowsazure/
Axis2	:	Web Services / SOAP / WSDL engine
EBS	:	Elastic Block Storage provided by Amazon
EC2	:	Elastic Cloud Computing http://www.amazon.com/ec2/
EGIS	:	Enterprise GIS Framework for Mobile Computing
Euca2ools	:	Eucalyptus user tools
Eucalyptus	:	An Open source Cloud System which provides IaaS
GAE	:	Google AppEngine, http://code.google.com/appengine/
GIS	:	Geographical Information Systems
GML	:	Geography Markup Language
IaaS	:	Infrastructure as a Service
IBM Blue Cloud	:	http://www.ibm.com/ibm/cloud/
ISO	:	International Standardization Organization
KVM	:	Kernel Virtual Machine
NIST	:	National Institute of Standards and technology
OGC	:	Open Geospatial Consortium
OpenNebula	:	An Open source Cloud System which provides IaaS
OpenStack	:	An Open source Cloud System which provides IaaS
PasS	:	Platform as a Service
Rampart	:	Security module for Apache Axis2
REST	:	Representational State Transfer
S3	:	Simple Storage Service provided by Amazon
SaaS	:	Software as a Service
SFDC	:	Sales-Force-Dot-Com, http://www.salesforce.com/
SOAP	:	Simple Object Access Protocol
W3C	:	World Wide Web Consortium
WFS	:	Web Feature Service
WMS	:	Web Map Service
WPS	:	Web Processing Service
WSDL	:	Web Service Description Language

Bibliography

[1] M. P. and G. T., "Perspectives on cloud computing and standards," tech. rep.

[2] A. Weiss, "Computing in the clouds," *netWorker*, vol. 11, pp. 16–25, Dec. 2007.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009.

[4] "The new age of cloud computing and gis." Website. http://www.esri.com/news/arcwatch/0110/feature.html.

[5] "Twenty experts define cloud computing." Website. http://cloudcomputing.sys-con.com/read/612375_p.htm.

[6] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy, survey, and issues of cloud computing ecosystems," in *Cloud Computing* (N. Antonopoulos and L. Gillam, eds.), vol. 0 of *Computer Communications and Networks*, pp. 21–46, Springer London, 2010. 10.1007/978-1-84996-241-4_2.

[7] T. Bernhardsen, *Geographic Information Systems: An Introduction*. Wiley, New York, 2002.

[8] L. Feng, P. M. G. Apers, and W. Jonker, "Towards con-text-aware data management for ambient intelligence," in *Proc. of the 15th Intl. Conf. on Database and Expert Systems Applications*, (Zaragoza, Spain), pp. 422–431, 2004.

[9] A. Dasgupta and S. Ghosh, "An enterprise gis framework for mobile device based on orchestration of geospatial services," tech. rep., SIT, Indian Institute of technology, Kharagpur.

[10] E. I. S. A. Victoria Kouyoumjian, "Gis in the cloud," tech. rep., ESRI, 6 2011. http://www.esri.com/library/ebooks/gis-in-the-cloud.pdf.

[11] Q. Huang, C. Yang, D. Nebert, H. Wu, and K. Liu, "Cloud compting for geosciences: Deployment of geoss clearinghouse on amazons ec2,"

[12] J. Blower, "Gis in the cloud: implementing a web map service on google app engine,"

[13] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "Eucalyptus : A technical report on an elastic utility computing archietcture linking your programs to useful systems," Tech. Rep. UCSB Computer Science Technical Report Number 2008-10, Computer Science Department,University of California, Santa Barbara,Santa Barbara, California 93106, 2010.

[14] "Uec getting started with ubuntu enterprise cloud." Website, 05 2011. https://help.ubuntu.com/community/UEC/.

[15] "Eucalyptus administrative guide." Website, 10 2011. http://open.eucalyptus.com/wiki/EucalyptusAdministratorGuide_v2.0.

[16] I. U. Sumayah Alrwais, "Behind the scenes of iaas implementations."

[17] "Openstack administrator guide." Website. http://ken.pepple.info/openstack/2011/04/22/openstack-nova-architecture/.

[18] "Opennebula administrative guide." Website. http://www.opennebula.org/documentation:rel2.2:intro.

[19] "Amazon elastic compute cloud developer guide." Website, 03 2006. http://docs.amazonwebservices.com/AmazonEC2/dg/2007-01-19/.

[20] "Amazon web service ebs." Website, 03 2006. http://aws.amazon.com/ebs/.

[21] "Amazon simple storage service developer guide." Website, 03 2006. http://docs.amazonwebservices.com/AmazonCloudFront/latest/DeveloperGuide/AccessLogs.html.

[22] R. Lake, D. Burggraf, M. Trinic, and L. Rae, *Geography Markup Language*. Chichester,England, John Wiley and Sons, 2004.

[23] O. WFS, "Web Feature Service implementation specification 1.0.0," tech. rep., 2002. http://portal.opengeospatial.org/specs.

[24] O. WMS, "Open GIS Web Map Service implementation specification 1.3," tech. rep., 2004. http://portal.opengeospatial.org/files/.

[25] "Cloud computing." Website. http://en.wikipedia.org/wiki/Cloud_computing.

[26] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pp. 124 –131, may 2009.

[27] R. Pointer, "Package paramiko." Website. http://www.lag.net/paramiko/docs/.