ABSTRACT

Cloud Computing is a coming technology in IT-Sector at a large scale due to the virtualization. As cloud computing is going to be a big market in IT fields, all kind of non-cloud environments tend to shift in clouds. The java applications that are deployed on non-cloud environments yet now, also tend to deploy on java platforms for cloud. In this process, there are several issues that affect the java applications on cloud platforms over non-clouds. This research points to one of these major issues- performance issue. A java application deployed on a java-cloud has a lesser performance than non-cloud platforms due to the framework and application specifications. To resolve this kind of issue, application and frameworks need to be optimized.

PROBLEM STATEMENT

Java Web Deployment in Cloud Computing is facing several challenges like performance issue, cost issue, security issue and reliability issue. When a java application is deployed on a java-cloud, the performance of the application is expected to be the equal or better to the non-cloud environments but in the matter of performance, java applications are facing this major challenge. Until unless this major issue is resolved, the cloud computing is not best suited platform for java-applications.

Each java-application is based on a particular framework. Thus to overcome these kinds of issues, the framework and application should be optimized or customized as per the requirements. These issues make the application lesser reliable in cloud computing. If these issues doesn't solve within a particular time, the java applications on clouds would never be a better than non-cloud.

CHAPTER – 1 CLOUDS AND CLOUD COMPUTING

1.1 INTRODUCTION

1.1.1 CLOUD COMPUTING

Cloud Computing can be considered as a Service over a network and a step ahead to the virtualization as service in the internet which is dynamic. It represents a separate way to explain and management computational resources. It includes delivery of the application as a service throughout the internet and the software that provide services in the data-centre and hardware and the paradigm shift. The data centre-software and hardware is known as a cloud.

Cloud Computing could be explained as an Example. Assume one wants to perform a computational task. For the task, he/she turns on the computer, the OS is loaded from hard drive, but if we use cloud computing, the OS is loaded from the network. This service provided over the network is known as Cloud Computing. It is an On-Demand Self-Service based technology. According to this, if one wants any resources to be used, requests for that over the network. Cloud Computing includes services like operating systems, software, data etc. The term cloud refers to the pool of virtualized computer resources including hardware and software resources.

Cloud Environment and Non-Cloud Environment could be differentiated as a telephone system. Non-Cloud Environment could be considered as Wired Telephones in Telephone systems. An IP address is assigned like 76.152.781.981. This IP address is like telephone number, as one can determine the location of the user, by watching the first few digits of Telephone number in wired telephony system. Apart from it, in Cellular phones, one cannot determine the location by its number as similar as Cloud Environment.

Most of the companies have started their trend with Cloud services Like Google has started Google App Engine (since July 2008), Microsoft started Windows Azure (since October 2008), Amazon started AWS, EC2 (Early 2006). These are having a revolutionary step in IT industry.

Web Deployment in Clouds are as similar as deploying a web-application on the server with many benefits. Web-Deployment on cloud facilitates one by shift paradigm.

1.1.2 CLOUD

A cloud is a pool of virtualized computer resources.
A cloud can:

- Group a variety of different loads, including wedge-style back-end works and collaborating ,User-facing applications
- Allow loads to be positioned and scaled-out swiftly through the quick provisioning of Virtual machines or somatic machines.
- Support terminated, self-recovering, extremely accessible programming prototypes those allow loads to improve from many obvious hardware/software disasters.
- Observer resource use in real time to enable rebalancing of provisions when desired.

A Cloud is a simulated galaxy available to deploy the applications, whereas Cloud Computing is a universal word for anything that involves distributing hosted services over the Internet. At its humblest, it is providing the assets and proficiencies of information technology enthusiastically as a service. Cloud Computing is a style of computing in which enthusiastically accessible and often virtualized assets are delivered as a service over the Internet. It generally incorporates Infrastructure as a Service (IaaS), Platform as a service (PaaS), and Software as a Service (SaaS).

1.1.3 POSITIVE ASPECTS

A. Attributes

The attributes of cloud computing are:

*1) Service Based***:** User worries are distant from supplier trepidations through service edges that are fine defined. The edges hide the execution specifics and allow a completely automated response by the provider of the service to the consumer of the service.

*2) Elastic***:** The service can scale capacity up or down as the consumer demands at the speed of full automation (which may be seconds for some services and hours for others). Elasticity is a trait of shared pools of resources.

*3) Shared***:** Services share a pool of resources to build economies of scale. IT resources are used with maximum efficiency. The underlying infrastructure, software or platforms are shared among the consumers of the service (usually unknown to the consumers). This enables unused resources to serve multiple needs for multiple consumers, all working at the same time.

*4) Metered by Use***:** Services are tracked with usage metrics to enable multiple payment models. The service provider has a usage accounting model for measuring the use of the services, which could then be used to create different pricing plans and models. These may include pay-as-you-go plans, subscriptions, fixed plans and even free plans. The implied payment plans will be based on usage, not on the cost of the equipment.

5)Uses Internet Technologies: The service is delivered using Internet identifiers, formats and protocols, such as URLs, HTTP, IP and representational state transfer Web-oriented architecture.

B. Benefits

The most frequently cited benefits of cloud computing are:

- It is agile, with ease and good speed of deployment
- Its cost is use-based, and will likely be reduced
- In-house IT costs are reduced
- Capital investment is reduced
- The latest technology is always delivered
- The use of standard technology is encouraged and facilitated.

As an application moves to the cloud, the access to it becomes more simple and ubiquitous. Low cost ultra-light devices and inexpensive hand held devices build on latest operating systems such as android provide access to the internet, the number and types of tasks taking advantage of the new technology will increase by several orders of magnitude, going far beyond the comparatively modest list of things that we use computers and the Internet for today.

1.2 CLOUD TYPES

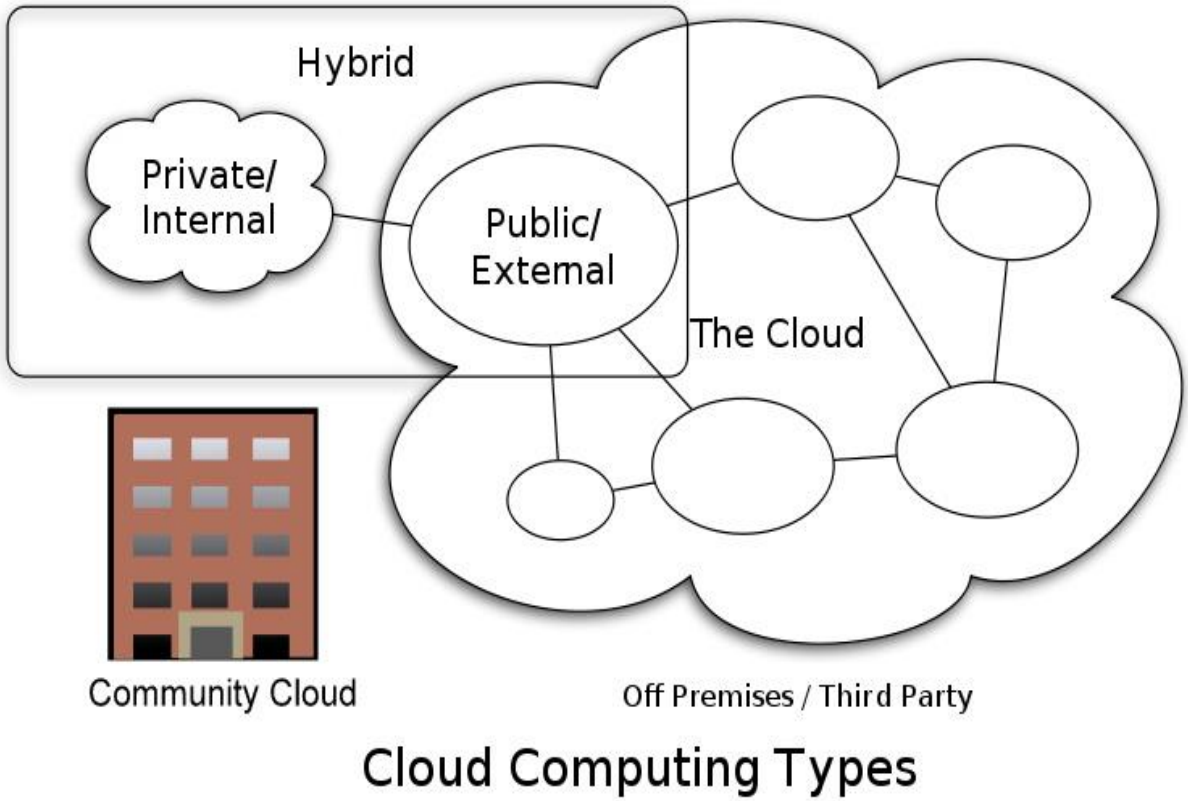
There are four kinds of clouds in cloud computing which are described as following:

1.2.1 PUBLIC CLOUD

A public cloud, or outer cloud, is the most common form of cloud computing, in which services are made available to the general public in a pay-as-you-go manner. Customer's individual users or enterprises access these services over the internet from a third-party provider who may share computing resources with many customers. The public cloud model is widely accepted and adopted by many enterprises because ,the leading public cloud vendors as Amazon, Microsoft and Google, have equipped their infrastructure with a vast amount of data centres, enabling users to freely scale and shrink their rented resources with low cost and little management burden. Security and data governance are the main concern with this approach.

1.2.2 PRIVATE CLOUD

A Private Cloud, or internal cloud, is used when the cloud infrastructure, proprietary network or data centre, is operated solely for a business or organization, and serves customers within the business firewall. Most of the private clouds are large company or government departments who prefer to keep their data in a more controlled and secure environment.



(Figure 1: Cloud Computing Types)

1.2.3 HYBRID CLOUD

A composition of the two types (private and public) is called a Hybrid Cloud, where a private cloud is able to maintain high services availability by scaling up their system with externally provisioned resources from a public cloud when there are rapid workload fluctuations or hardware failures. In the Hybrid cloud, an enterprise can keep their critical data and applications within their firewall, while hosting the less critical ones on a public cloud.

1.2.4 COMMUNITY CLOUD

The idea of a Community Cloud is derived from the Grid Computing and Volunteer Computing paradigms. In a community cloud, several enterprises with similar requirement can share their infrastructures, thus increasing their scale while sharing the cost. Another form of community cloud may be established by creating a virtual data centre from virtual machines instances deployed on underutilized users machines.

1.3 CLOUD SERVICES

A cloud is a collection of systems and their classes that provides remote availability on the user end. All the resources, applications are part of a cloud. The services provided by cloud computing has a following classification:

1.3.1 INFRASTRUCTURE AS A SERVICE

The IaaS is further classified into:

i) Computation as a Service (CaaS): in this, the servers which are based on VM (Virtual Machine) are hired and charged as time based on the capacity– mainly CPU and RAM size, features of the virtual machine, OS and deployed software.

ii) Data as a Service (DaaS), in this, an unlimited data storage is provided to all the users to store their data and the cost of the data is charged as per GB and depends on the provider.

Google has provided a standard all-inclusive answer to Cloud Computing, known as the Google App Engine. App Engine delivers many useful features for consumers, including an established and inexpensive billing system able to charge for computing at a very fine-grained level (memory usage, CPU usage, data transfer, etc.), deployment between multiple locations, elastic IP addresses, connection to a customer's existing infrastructure through a Virtual Private Network, watching services by Google App Engine. App Engine has installed such well granularity and exactness that it has become a standard model in cloud computing.

Go Grid also provides Hybrid Hosting, which is a distinguishing feature. Many applications simply don't run well in a pure multi-tenant server environment. Databases perform better on a dedicated server where they don't have to compete for input/output resources, and the situation is similar with web server applications. Go Grid provides these special applications with dedicated servers that also have high security assurance.

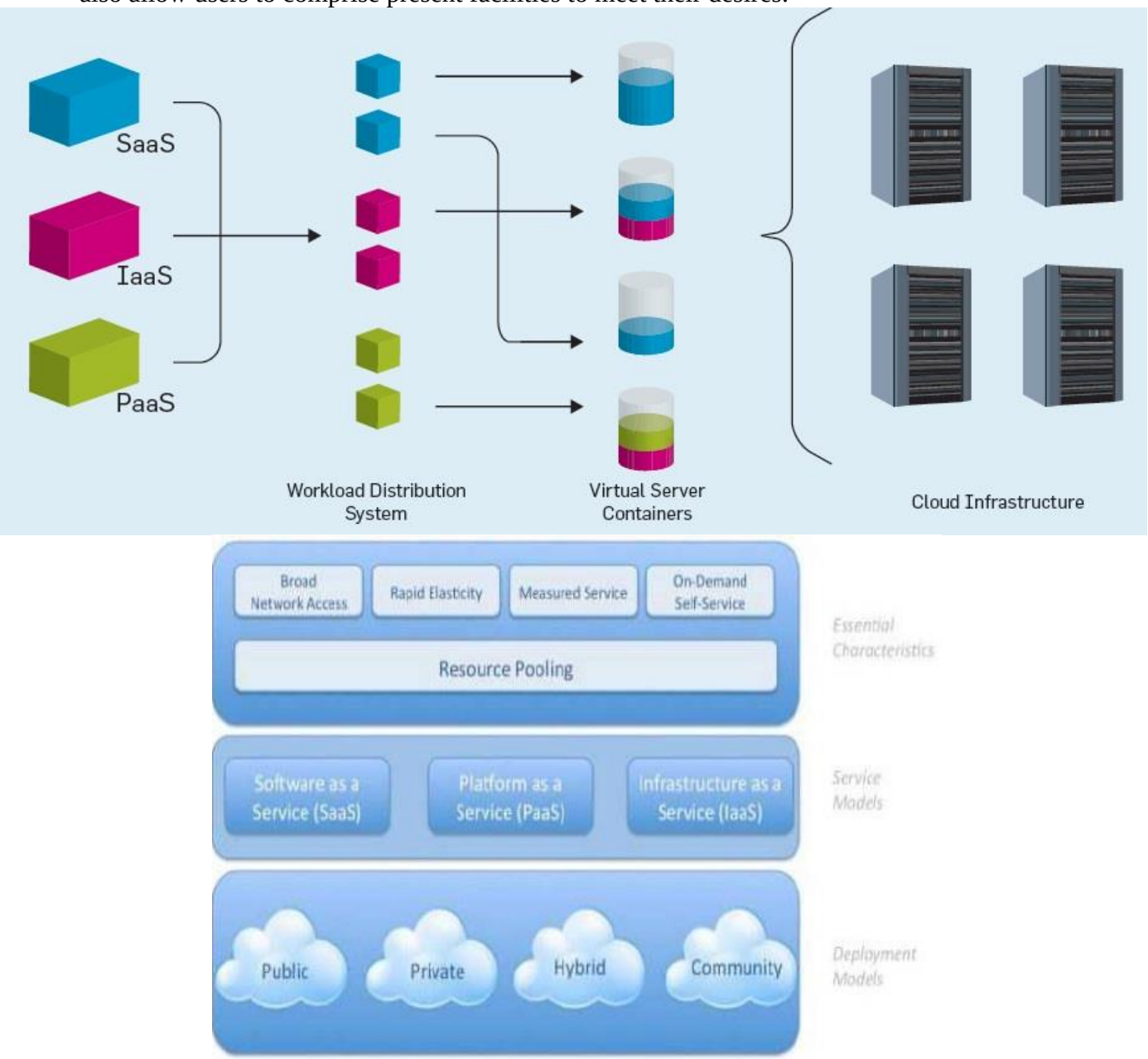
1.3.2 PLATFORM AS A SERVICE

Platform as a Service (PaaS) cloud systems offer an performance atmosphere that application facilities can run on. The atmosphere is not just a pre-installed operating system but is also combined with a programming-language-level platform, which users can be used to improve and build uses for the platform.

Microsoft's cloud policy is to build a cloud stage that users can move their applications to in a continuous way, and ensure its managed assets are accessible to both cloud services and on-premises applications. To reach this, Microsoft presented the Windows Azure Platform (WAP), which is tranquil of a cloud operating system named Windows Azure, and a set of supportive services. Windows Azure is the main part of the WAP. It services virtual machines as its runtime environments.

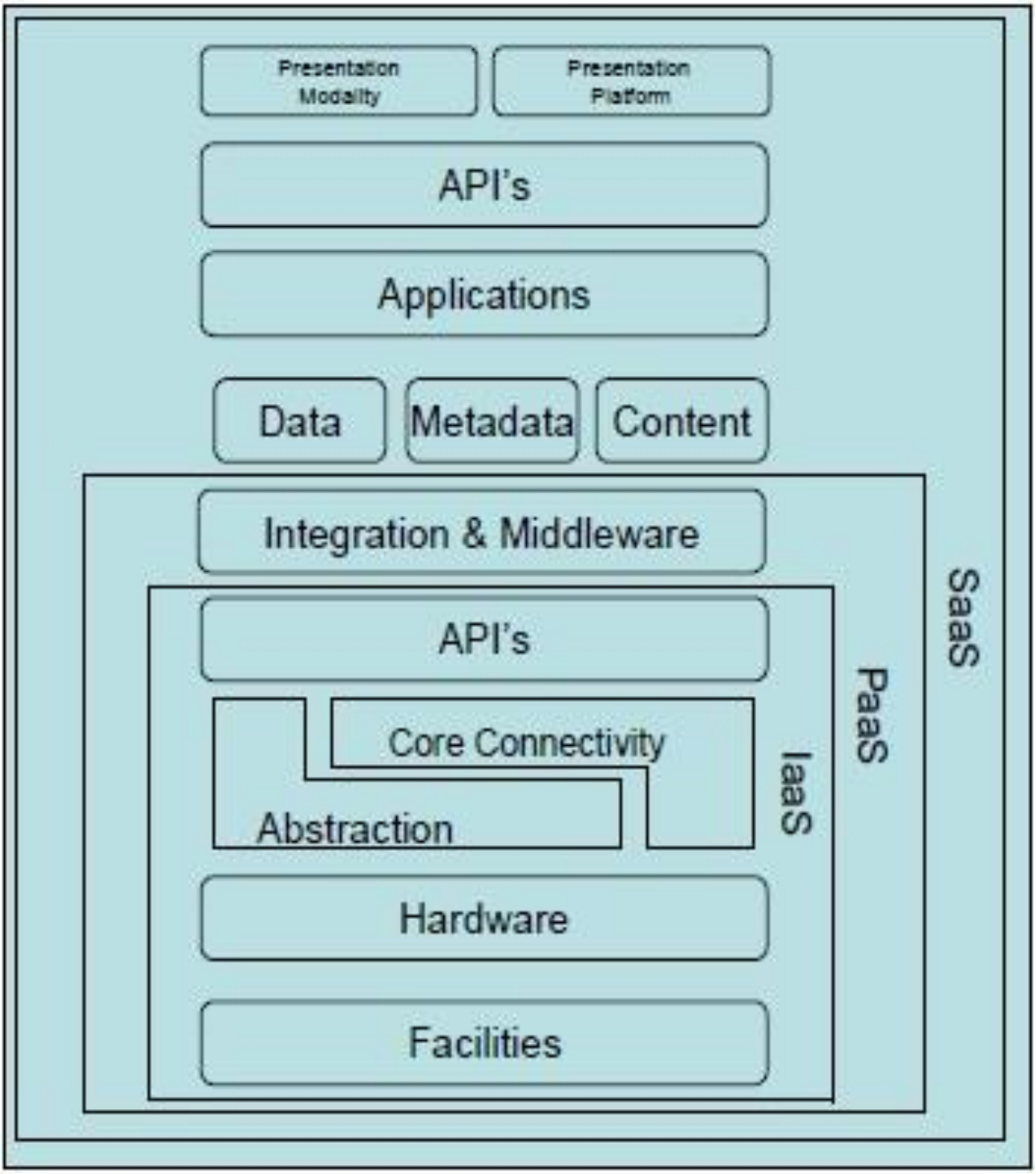
1.3.3 SOFTWARE AS A SERVICE

Software-as-a-Service (SaaS) is based on certifying software use on plea, which is already installed and running on a cloud platform. These on-demand applications may have been developed and installed on the PaaS or IaaS layer of a cloud platform. SaaS replaces outdated software convention with a Contribute/Rent model, reducing the user's physical equipment deployment and management costs. The SaaS clouds may also allow users to comprise present facilities to meet their desires.

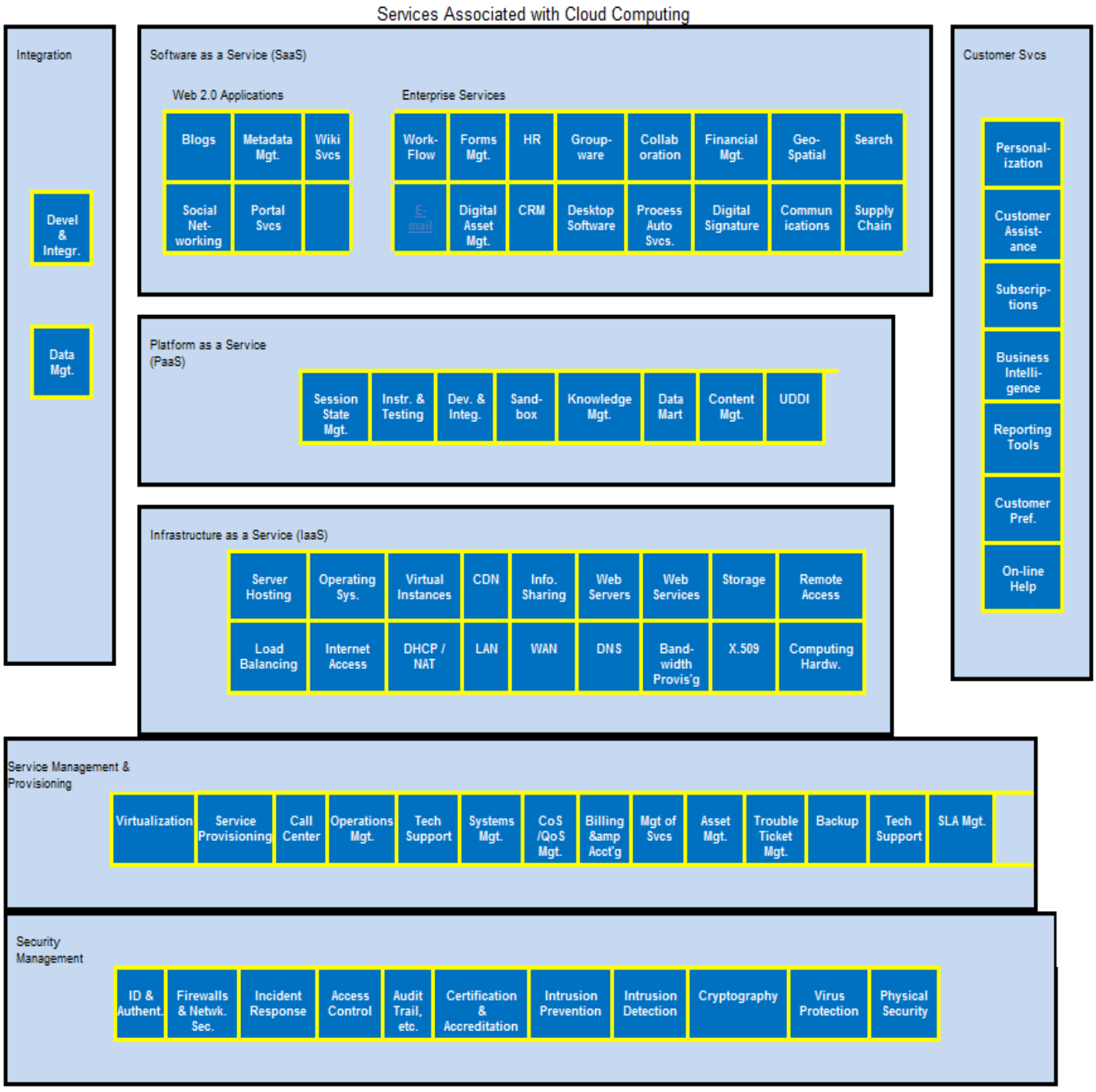


(Figure 2: Cloud Services)

1.4 CLOUD ARCHITECTURE



(Figure 3 (a): Cloud Architecture)



(Figure 3 (b): Services Associated Cloud Architecture)

CHAPTER 2 – JAVA PLATFORMS IN CLOUD COMPUTING

2.1 JAVA-CLOUD PROVIDERS

There are several cloud platforms that provide java environment in cloud-market. Some of these are discussed here:

2.1.1 Google

2.1.1.1 Google App Engine

Google App Engine introduced by Google in July 2008. Google App Engine being a huge cloud platform for java and python provides several services on its cloud (Google Cloud-The Public Cloud). Google provides SaaS (Software as a Service) and PaaS (Platform as a Service). Google App Engine is free up to certain services. It charges for storage (database), higher-bandwidth etc. Google App Engine is reliable, easy and efficient for web-applications. Google App Engine provides Java Runtime Environment and Python Runtime Environment. It has 9-10% market share.

Google App Engine Applications are virtualized by multiple servers and Data-centers. GAE uses a security technique called defense in depth for application security.

2.1.2 AMAZON WEB SERVICES

Amazon Web Services (AWS) provides the top-most cloud computing platforms like EC2 (Elastic Computing Cloud). It provides public clouds like GAE. Amazon started its cloud computing by introducing a cloud based message queuing service which was known as Amazon Simple Queue Service (SQS). By adding some services, they introduced Simple Storage Service (S3) and then Elastic Compute Cloud (EC2) was introduced. AWS provides RDS (Relational Data Services) for database services. It has the maximum market share with 50-60% shares.

2.1.2.1 EC2 (Elastic Compute Cloud)

Xen Virtualization is used in EC2. EC2 rents the virtual servers that run on a remote location. These servers are known as Amazon Machine Images (AMI).EC2 provides a good security environment by using multilevel security strategies. It provides a perfect environment for virtual computing. The instances of the virtual servers function as a virtual private sector.

2.1.2.2 S3 (Simple Storage Service)

Simple Storage Service is an infrastructure service that is used to store data in a better manner. S3 stores data with their data too (metadata). Buckets are used to manage the objects in S3.Bucket Size may vary from 2KB to 5 GB for metadata. S3 provides the virtual file system for constant storage for its applications.

2.1.2.3 SQS (Simple Queue Service)

Amazon Simple Queue Service is used as message queue in clouds. REST and SOAP calls could be used to access SQS as similar to S3. It provides On-Demand Self-Service facility. Message Oriented Middleware is used in SQS that makes sure that the message is delivered only once.

2.1.3 MICROSOFT

2.1.3.1 Windows Azure

Microsoft provides Operating System as a Service. Windows Azure is an Operating System which is a Cloud Operating System. As per a survey, it is commercially being used in 40 countries. It supports almost all kinds of languages like C++, C#, .NET, SOAP, REST, XML, JAVA, PHP, and RUBY for application-developments. The data stored in Windows Azure are in blobs, tables and queues.

2.1.4 RACKSPACE

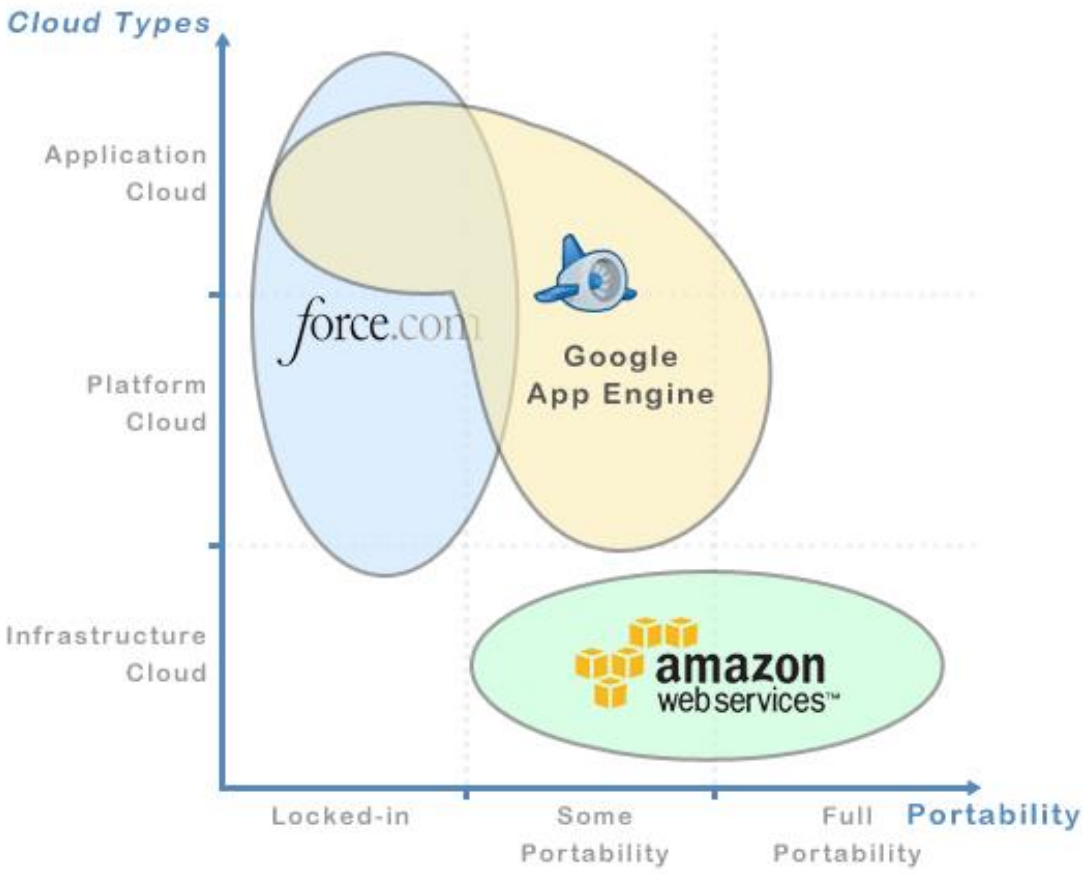
RackSpace Cloud was introduced in March, 2006. RackSpace provides public and private clouds. It only provides Infrastructure as a Service. It provides the top-most enterprise level hosting services. It offers cloud sites, cloud files, cloud servers and cloud load balancers. Cloud Sites are good hosting platform for traditional web-hosting services. Cloud Files are used for data storage and developed by OpenStack. Cloud Servers are used to deploy the applications by the clients and use Xen Virtualization as similar to Amazon.

2.1.5 SALESFORCE

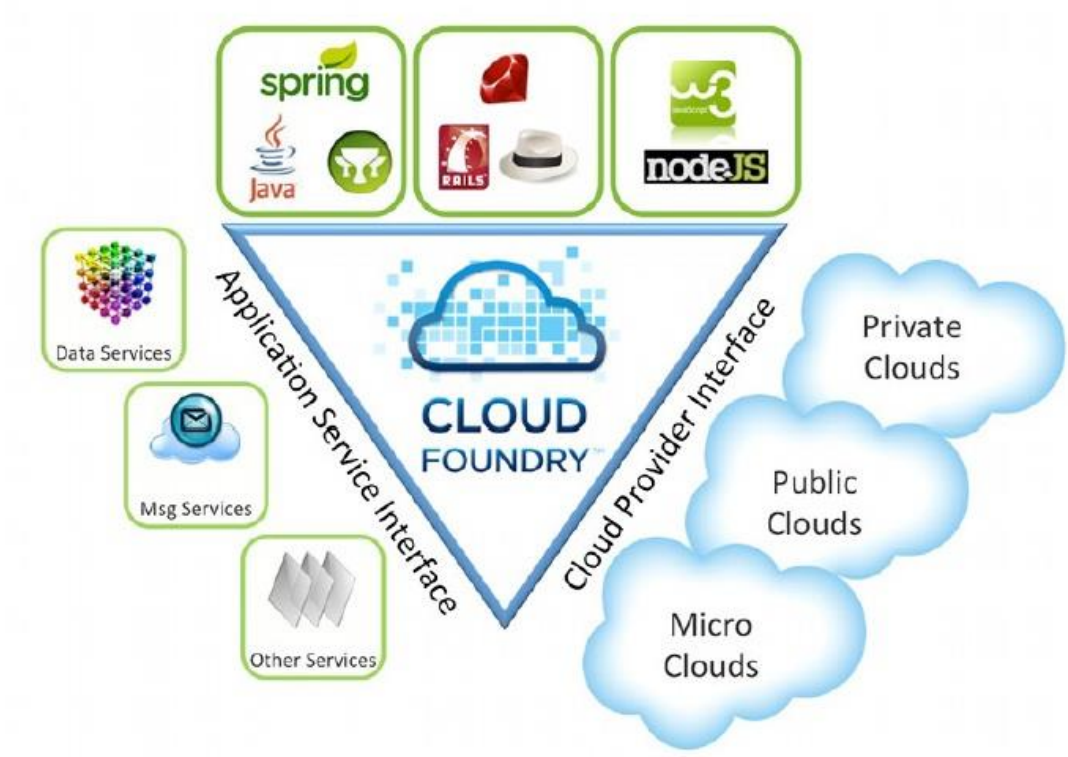
SalesForce is top-leader for the business purpose applications. It provides Private Cloud only. It was introduced in 1999. It provides PaaS and SaaS but IaaS. Some of its services are: Sales Cloud, Chatter, Service Cloud, Force.com, Data Cloud, Jigsaw, Heroku, RemedyForce and AppExchange.

2.2 COMPARISON AMONG THE JAVA CLOUD PLATFORMS

	AMAZON AWS	GOOGLE	MICROSOFT	RACKSPACE	SALESFORCE
PRODUCTS / SERVICES	EC2, SQS, S3	GAE (Google App Engine)	Windows Azure, SQL Azure	Cloud Hosting, Cloud Services, Cloud Sites, Cloud Files, Cloud Load Balancers	Sales Cloud, Force.com, Heroku
SERVICE STARTED	EARLY 2006	JULY 2008	OCTOBER 2008	MARCH 2006	1999
CLOUD TYPE	PUBLIC CLOUD	PUBLIC CLOUD	PRIVATE CLOUD AND PUBLIC CLOUD	PRIVATE CLOUD AND PUBLIC CLOUD	PRIVATE CLOUD
CLOUD SERVICES	IaaS, PaaS	SaaS, PaaS	PaaS, IaaS	IaaS	SaaS, PaaS
MARKET SHARE	50-60%	9-10%	8-9%	10%	8%



(Figure 4: Cloud Portability in Several Clouds)



(Figure 5: Cloud Foundry)

CHAPTER 3 – ISSUES IN JAVA WEB-DEPLOYMENT IN CLOUD COMPUTING

3.1 INTRODUCTION

As there are several platforms for java clouds. Google App Engine is used for the analysis of the issues in java web-deployment in cloud computing.

Google App Engine, as discussed earlier, developed by Google in Mid 2008, has been a reliable, easy and cost-efficient cloud platform for java and python developers/companies. It provides java runtime environment and python runtime environment. It also provides Dynamic Web Serving. Dynamic Web Serving means the dynamic management of the applications over the cloud. It also provides the facility to optimize the software and hardware resources. Dynamic Web Services have been a powerful feature of google app engine due to its flexibility functionality. Every application is individually managed as per the dynamic serving. This also affects the performance and cost of the application.

Google App Engine functionality also includes the Triggers Scheduled Tasks. Triggered Scheduled Tasks always have been a helping hand for automation. It reduces manual processing which reduces the application-management-time. Triggering the tasks helps applications for multiple helpful processes and makes the application reliable.

Google App Engine queues the tasks for batching jobs. Assume there are n numbers of jobs like j_1, j_2, j_3…, j_n. It queues the jobs in a queue using a queue-management-system. The jobs are differentiated through their priorities. The batch which is created to handle the jobs in queue, has several features like in the case of same priority jobs, First Come First Serve (FCFS) is used. Batching jobs sometimes may reduce the application performance due to their scheduling facilities like if there are any job which require some particular resource and isn't available at that time, it would have to wait along the remaining jobs (having lesser priority that this job).

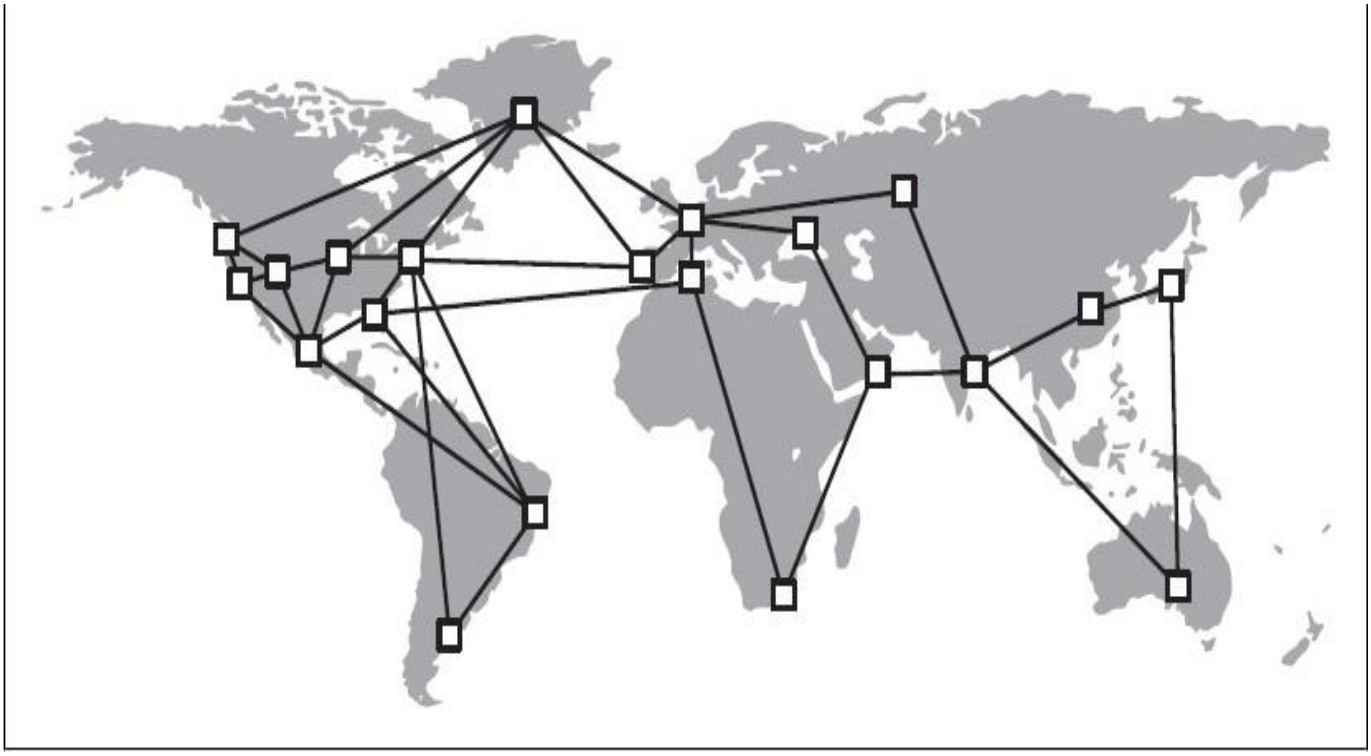
Google App Engine requires Google Account. If one having google account, doesn't need to create separate for the Google App Engine. Google App Engine also provides the email delivery about the deployed application and its new features.

It provides persistent storage. Persistent Storage means the predefine storage for any particular. In queries, sorting and transactions, the storage which is required should be specific. Persistent Storage is also beneficial for the applications in the matter of reliability and the performance.

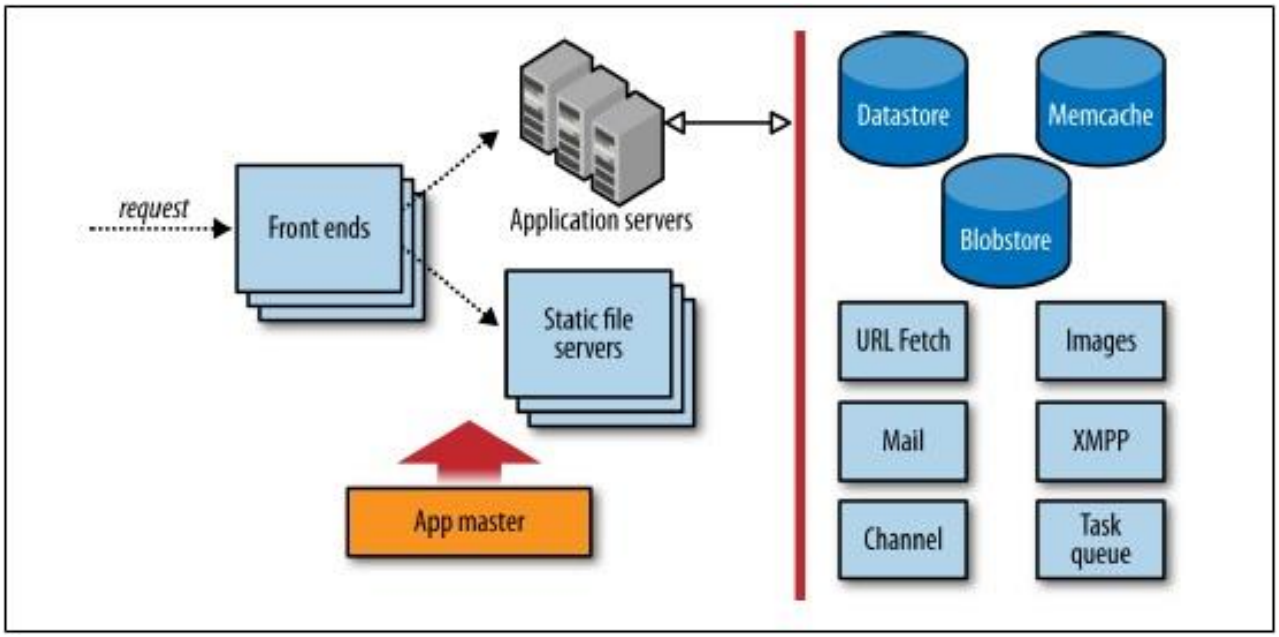
Google App Engine provides facilities for both java and python. Python and Java applications may co-exist in the Run Time Environment. Java works on Servlet Technology and Python works on WSGI technology.

It provides Automatic Scaling which is used to scale the application in the matter of size. Scaling is used for calculating the bandwidth required for the application. GAE also provides load balancing in which the load to a particular application is balanced if the co-

applications are having lesser load than the application, the load could be balanced. Load balancing is calculated by number of users, number of requests etc.



(Figure 6: Data-Centre of Google)



(**Figure 7: Architecture of Request Handling in Google App Engine)**

Scalability provided by GAE is Datastore. Relational Databases are not used in GAE due to the lack of scalability. Google provides Big-Table-Infrastructure, which is used as a datastore in GAE applications. It provides unlimited size of datasets. First public API of Big-Table is Datastore. If one wants to use datastore, he needs to use Java Data Objects (JDO) or Java Persistence API (JPA) class. One does not need to normalize its data to use Datastore.

3.2 DEPLOYMENT OF A JAVA-WEB-APPLICATION ON GOOGLE APP ENGINE

The architecture of App Engine—and therefore an App Engine application—can be summarized as shown in Figure 3-1. (There are some lines missing from this simplified diagram. For instance, frontends have direct access to the Blobstore. We'll take a closer look at these in later chapters.)

The first stop for an incoming request is the App Engine frontend. A load balancer, a dedicated system for distributing requests optimally across multiple machines, routes the request to one of many frontend servers. The frontend determines the app for which the request is intended from the request's domain name, either the Google Apps domain and subdomain or the *appspot.com* subdomain. It then consults the app's configuration to determine the next step.

The app's configuration describes how the frontends should treat requests based on their URL paths. A URL path may map to a static file that should be served to the client directly, such as an image or a file of JavaScript code. Or, a URL path may map to a request handler, application code that is invoked to determine the response for the request. You upload this configuration data along with the rest of your application.

If the URL path for a request does not match anything in the app's configuration, the frontends return an HTTP 404 Not Found error response to the client. By default, the frontends return a generic error response. If you want clients to receive a custom response when accessing your app (such as a friendly HTML message along with the error code), you can configure the frontend to serve a static HTML file.

A web application is an application that responds to requests over the Web. Typically, these requests come from a user's web browser, when the user types the URL of your app or visits a link or bookmark, or when your app's JavaScript client code makes requests of its own. Requests could also come from other clients on the network, such as mobile or desktop applications, or systems accessing your app as a service.

To build an App Engine application, you write code for one or more *request handlers*, and describe to App Engine which requests go to which handlers, using configuration. The life of a request handler begins when a single request arrives, and ends when the handler has done the necessary work and calculated the response.

If the URL path of the request matches the path of one of the app's static files, the frontend routes the request to the static file servers. These servers are dedicated to the task of serving static files, with network topology and caching behavior optimized for fast delivery of resources that do not change often. You tell App Engine about your app's static files in the app's configuration. When you upload the app, these files are pushed to the static file servers.

If the URL path of the request matches a pattern mapped to one of the application's request handlers, the frontend sends the request to the app servers. The app server pool starts up an instance of the application on a server, or reuses an existing instance if there is one already running. The server invokes the app by calling the request handler that corresponds with the URL path of the request, according to the app configuration.

A request handler runs in an *application instance*, a copy of your application in the memory of an application server. The instance is in a portion of the server isolated from whatever else is on the machine, set up to perform equivalently to a dedicated machine with certain hardware characteristics. The code itself executes in a *runtime environment* prepared with everything the request handler needs to inspect the request data, call services, and evaluate the app's code. There's enough to say about instances and the runtime environment that we'll give the subject its own chapter

You can configure the frontend to authenticate the user with Google Accounts. The frontend can restrict access to URL paths with several levels of authorization: all users, users who have signed in, and users who are application administrators. With a Google Apps domain, you can also set your application to allow only users on the domain to access URLs, such as for an employee-only website or school campus. The frontend checks whether the user is signed in, and redirects the user to the Google Accounts sign-in screen if needed.

The frontend takes the opportunity to tailor the response to the client. Most notably, the frontend compresses the response data, using the gzip format, if the client gives some indication that it supports compressed responses. This applies to both app responses and static file responses, and is done automatically. The frontend uses several techniques to determine when it is appropriate to compress responses, based on web standards and known browser behaviors. If you are using a custom client that does not support compressed content, simply omit the "Accept-Encoding" request header to disable the automatic gzip behavior.

JRE FOR DEPLOYING APPLICATION:

The Java runtime environment behaves like a J2EE servlet container. When the app instance receives a request, it determines the servlet class to call by comparing the URL path to the servlet mappings in the deployment descriptor. The server uses the standard servlet interface to invoke the servlet, passing it a populated request object and an empty response object. The application's servlet code populates the response object and exits, and App Engine returns the response to the client.

The Java runtime environment uses the Java 6 virtual machine (JVM). The JVM runs Java bytecode, which is what you get from passing Java code to a Java compiler. It's also what you get from compilers for other languages that produce Java bytecode, such as Scala, and from interpreters for other languages implemented in Java bytecode, such as JRuby (Ruby), Rhino (JavaScript), Groovy, and even Jython (a Python interpreter implemented in Java). You can use any language that compiles to or has an interpreter for the JVM to write applications for App Engine, as long as the result implements a servlet interface.

Having a complete JVM also means you can use many third-party libraries with your application. Some restrictions apply—we'll look at a few in a moment—but in most cases, using a library is a simple matter of including the JAR or class files in the application's WAR.

An app can ask for information about the current environment by using the `SystemProperty` API, in the `com.google.appengine.api.utils` package. App Engine sets static fields of this class to the application ID (`applicationId`), the application version (`applicationVersion`), the version of the runtime environment (`version`), and whether the app is running in the development environment or on App Engine (`environment`):

```
import com.google.appengine.api.utils.SystemProperty;

// ...
        String applicationId = SystemProperty.applicationId.get();

        if (SystemProperty.environment.value() ==
            SystemProperty.Environment.Value.Development) {
            // ... only executed in the development server ...
        }
```

In the Java runtime environment, sandbox restrictions are enforced within the JVM. These restrictions are implemented using a combination of JVM permissions, a Java Runtime Environment (JRE) class whitelist, and alternate implementations for specific functions. This fine-grained approach allows more third-party libraries to work and makes other code easier to port than relying on JVM permissions alone.

The Java runtime environment includes a subset of the JRE classes. You can find a complete list of supported JRE classes in the official documentation. The development server enforces this list, so if your code (or some library code) crosses a line, the development server throws an exception.

Reflection is supported for all the app's own classes. Custom class loaders are supported, but with all classes granted the same permissions. Native JNI code is not supported.

3.2.1 HOW TO DEPLOY AN APPLICATION ON JAVA CLOUD

Following steps are used to deploy a java web application in Google App Engine:

1.) Insert the Google app engine configuration file named appengine-web.xml which includes the application name, application version and application information.

2.) Download the appengine-java-sdk tool which is freely available over the internet or Netbeans and Eclipse provides plug-ins for java deployment on google app engine.

3.) The appengine-java-sdk tools provide a appcfg.cmd file through which we can deploy our web application.

4.) First we need to create an application on appengine.google.com after authentication of google-account. The application name should be as same to our java web-application.

5.) Using command prompt, following command would deploy the java web-application on the Google App Engine:

    appcfg.cmd update PATH_TO _JAVA_WEB_APPLICATION

6.) By submitting the following command it asks for google account authentication. After the success of authentication, it uploads the java web application on google app engine in the authenticated account.

7.) Now the application is successfully deployed on google app engine. To access the web-page, the URL provided by google app engine would be:
    application_name.appspot.com

8.) To access the administration of all deployed web applications, appengine.google.com need to be visited.

Database Vocabulary Map

Datastore Vocabulary Map	
Relational DataBase	Google DataStore Equivalent
Database	Datastore
Table	Kind
Row	Entitiy
Row ID	Key
Column	Property

The appengine configuration file which is appengine-web.xml includes following code:

<?xml version="1.0" encoding="utf-8"?>

<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">

<application> application_name </application>

<version> version no. </version>

<!--

Allows App Engine to send multiple requests to one instance in parallel:

-->

<threadsafe> true </threadsafe>

<precompilation-enabled> true </precompilation-enabled>

<!-- Configure serving/caching of GWT files -->

<static-files>

<include path="**" />

<!-- The following line requires App Engine 1.3.2 SDK -->

<include path="**.nocache.*" expiration="0s" />

<include path="**.cache.*" expiration="365d" />

<exclude path="**.gwt.rpc" />

</static-files>

<!-- Configure java.util.logging -->

<system-properties>

<property name="java.util.logging.config.file" value="WEB INF/logging.properties"/>

<property name="app-id" value="app-name"/>

</system-properties>

<!-- HTTP Sessions are disabled by default. To enable HTTP sessions specify:

<sessions-enabled>true</sessions-enabled>

It's possible to reduce request latency by configuring your application to asynchronously write HTTP session data to the datastore:

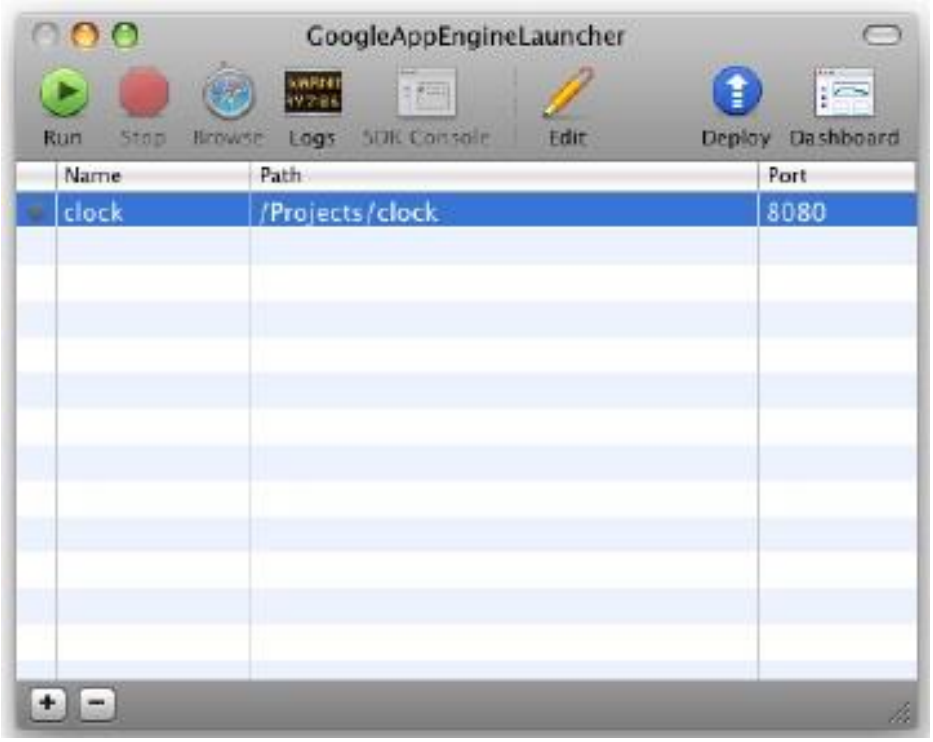
<async-session-persistence enabled="true" /> with this feature enabled, there is a very small chance your app will see

Stale session data. For details, see

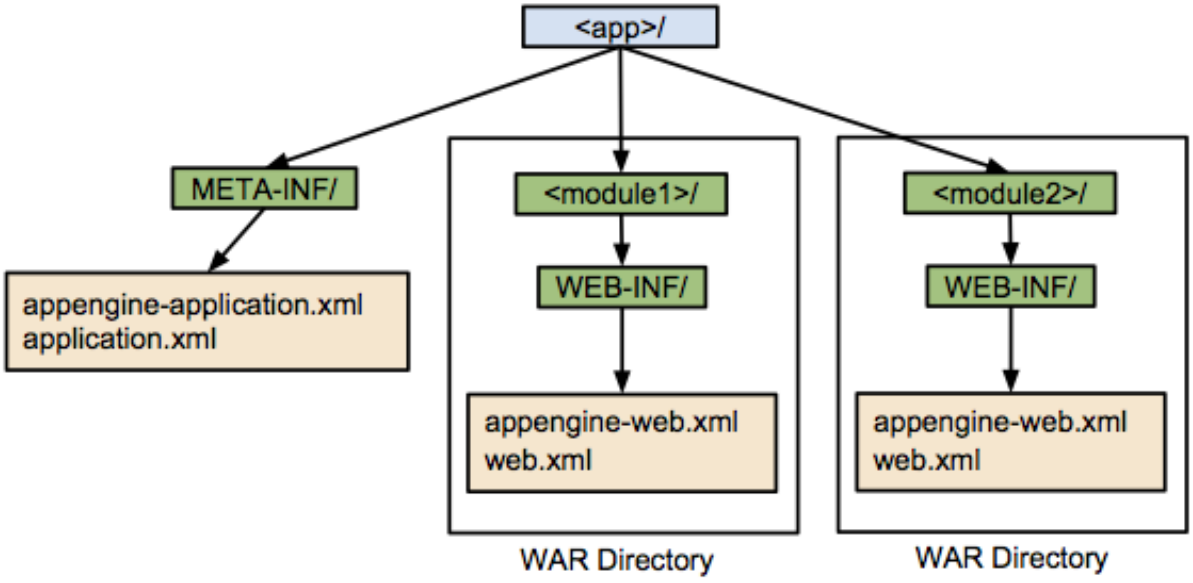
http://code.google.com/appengine/docs/java/config/appconfig.html#Enabling_Sessions

-->

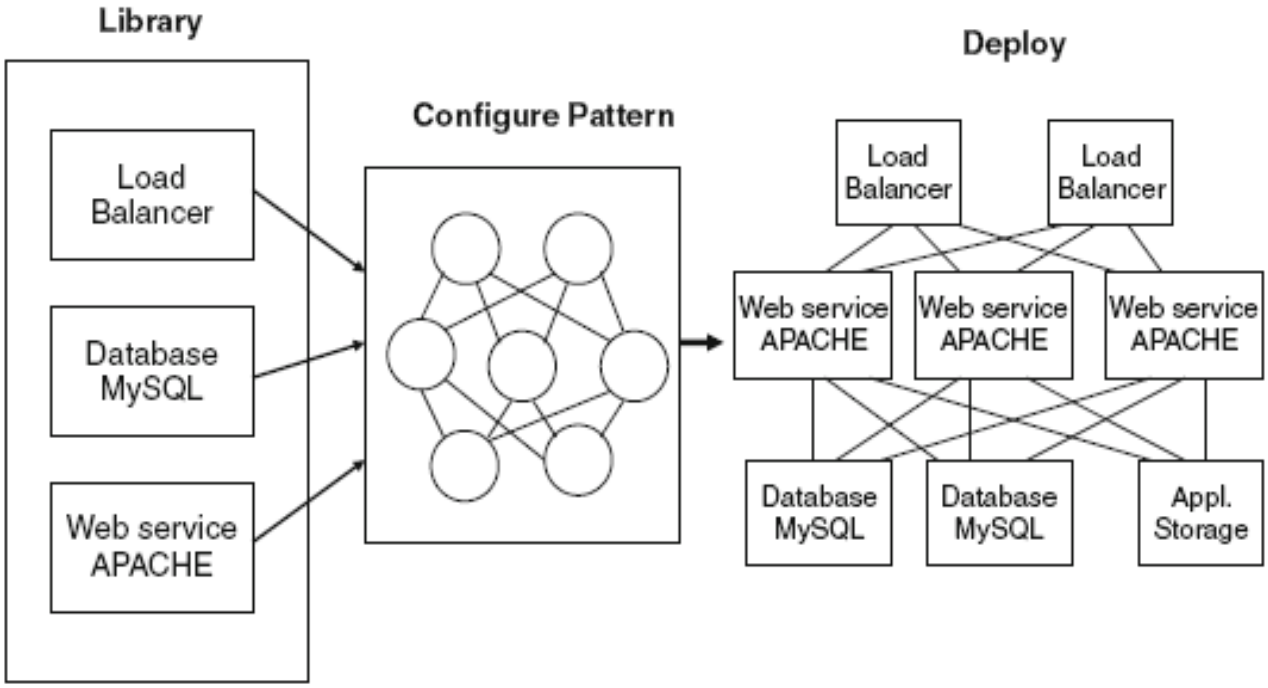
</appengine-web-app>



(Figure 8: Google App Engine Launcher for Mac OS)



(Figure 9: File Management in GAE Application)



(Figure 10: Deployment Strategy in GAE)

3.2.2 DEPLOYING A SAMPLE JAVA WEB-APPLICATION ON GOOGLE APP ENGINE

In this section we will develop a simple java web application and deploy it on the Google App Engine. First we will develop a simple java web application named SampleJavaApp.

SampleJavaApp (The root folder of Java Application)

- WEB-INF
  - classes
    - myapps
      - Servlet1.class
      - Servlet2.class
  - appengine-web.xml
  - web.xml
- index.html

1. ***index.html :***

```
<html> <head>
<title>Welcome!</title> </head>
<body bgcolor="PINK">
<table width="100%" height="100%">
<tr height="15%" align="center">
<td>
<h1>SAMPLE JAVA APP</h2><HR>
</td>
</tr>
<tr height="60%" align="center">
<td>
<form name="form1" action="Servlet2" method="POST">
<table cellspacing="8" cellpadding="8" border="2">
<tr>
<b>LOGIN FORM:</b>
<td>
<b>USER-NAME: </b>
</td>
<td>
<input type="text" name="uname"/>
</td>
</tr>
<tr>
<td>
<b>PASSWORD: </b></td>
<td>
<input type="password" name="pwd"/>
</td>
</tr>
```

```
</table>
<input type="submit" name="submit" value="SUBMIT">
<input type="reset" name="reset" value="RESET">
</form> <BR>
<b>REGISTRATION FORM:</b>
<form name="form2" action="Servlet1" method="POST">
<table cellspacing="8" cellpadding="8" border="2">
<tr>
<td>
<b>NAME: </b>
</td>
<td>
<input type="text" name="name"/>
</td>
</tr>
<tr><td>
<b>E-MAIL ID: </b></td>
<td><input type="text" name="email"/>
</td>
</table>
<input type="submit" name="submit" value="SUBMIT">
<input type="reset" name="reset" value="RESET">
</form>
</td>
</tr>
<tr height="20%">
</tr>
<tr height="5%" align="center">
<td>
<hr> ANKIT KUMAR SAHU 2013
```

```
</td>
</tr>
</table>
</body>
</html>
```

2. ***web.xml :***

```
<web-app>
    <servlet>
        <servlet-name>Servlet1</servlet-name>
        <servlet-class>myapps.Servlet1</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Servlet1</servlet-name>
        <url-pattern>/Servlet1</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>Servlet2</servlet-name>
        <servlet-class>myapps.Servlet2</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Servlet2</servlet-name>
        <url-pattern>/Servlet2</url-pattern>
    </servlet-mapping>
    <servlet>
        <servlet-name>ErrorHandlerServlet</servlet-name>
        <servlet-class>myapps.ErrorHandlerServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ErrorHandlerServlet</servlet-name>
        <url-pattern>/error</url-pattern>
```

```
        </servlet-mapping>
        <welcome-file-list>
            <welcome-file>index.html</welcome-file>
        </welcome-file-list>
</web-app>
```

3. ***appengine-web.xml :***

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
<application> samplejavaapp </application>
<version>1</version>
<threadsafe>true</threadsafe>
<system-properties>
        <property name = "java.util.logging.config.file" value = "WEB-INF/logging.properties"/>
        <property name="app-id" value="samplejavaapp"/>
 </system-properties>
</appengine-web-app>
```

4. ***Servlet1.java :***

```
package myapps;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;
public class Servlet1 extends HttpServlet
{
public void doGet(HttpServletRequest servletrequest, HttpServletResponse servletresponse)
{
PrintWriter swriter=null;
try{
swriter =servletresponse.getWriter();
```

```
String name=servletrequest.getParameter("name");
String email=servletrequest.getParameter("email");
name=name.trim();
email=email.trim();
if(name.equals("") || email.equals(""))
        servletresponse.sendRedirect("index.html");
else
        {
        swriter.println("<HTML><BODY BGCOLOR=\"PINK\">");
        swriter.println("<TABLE WIDTH=\"100%\" HEIGHT=\"100%\">");
        swriter.println("<TR HEIGHT=\"20%\" ALIGN=\"CENTER\">");
        swriter.println("<TD><h1> SAMLE JAVA APP </h1><HR></TD></TR>");
        swriter.println("<TR HEIGHT=\"70%\" ALIGN=\"CENTER\"><TD>");
        swriter.println("<h1>HELLO "+name+"!"+"</h1>");
        swriter.println("<h2>YOUR E-MAIL:"+email+" IS REGISTERED</h2>");
        swriter.println("<p>your login details will be sent u via mail...as soon as
        possible</p>");
        swriter.println("<A HREF=\"index.html\">HOME</a>");
        swriter.println("</TD></TR>");
        swriter.println("<TR HEIGHT=\"10%\" ALIGN=\"CENTER\">");
        swriter.println("<TD><hr>ANKIT KUMAR SAHU 2013</TD></TR>");
        swriter.println("</TABLE></BODY></HTML>");
        }
}
catch(Exception myappexcptn)
{
        try{
        servletresponse.sendRedirect("error");
        }
        catch(Exception myappexcptn2)
        {
```

```
            System.out.println(myappexcptn2.getMessage());
            }
    }
    finally
    {
            try{
            if(swriter!=null)
            swriter.close();
            }
            catch(Exception myappexcptn3)
            {
            System.out.println(myappexcptn3.getMessage());
            }
    }
    }
    public void doPost(HttpServletRequest servletrequest, HttpServletResponse
    servletresponse)
    {
            doGet(servletrequest, servletresponse);
    }
    }
```

5. ***Servlet2.java :***

```
    package myapps;
    import javax.servlet.http.HttpServlet;
    import javax.servlet.http.HttpServletRequest;
    import javax.servlet.http.HttpServletResponse;
    import java.io.PrintWriter;
    public class Servlet2 extends HttpServlet
    {
    public void doGet(HttpServletRequest servletrequest, HttpServletResponse
    servletresponse)
```

```
{
PrintWriter swriter=null;
try{
swriter =servletresponse.getWriter();
String uname=servletrequest.getParameter("uname");
String pwd=servletrequest.getParameter("pwd");
uname=uname.trim();
swriter.println("<HTML><BODY BGCOLOR=\"PINK\">");
swriter.println("<TABLE WIDTH=\"100%\" HEIGHT=\"100%\">");
swriter.println("<TR HEIGHT=\"20%\" ALIGN=\"CENTER\">");
swriter.println("<TD><h1> SAMLE JAVA APP </h1><HR></TD></TR>");
swriter.println("<TR HEIGHT=\"70%\" ALIGN=\"CENTER\"><TD>");
if(uname.equals("admin") && pwd.equals("admin"))
{
        swriter.println("<h1>HELLO !</h1>");
        swriter.println("<h2>YOUR LOGIN IS SUCCESSFUL</h2>");
}
else if(uname.equals("") || pwd.equals(""))
{
        servletresponse.sendRedirect("index.html");
}
else
{
        swriter.println("<h1>TRY AGAIN</h1>");
        swriter.println("<h2>YOUR LOGIN IS FAILURE</h2>");
}
swriter.println("<A HREF=\"index.html\">HOME</a>");
swriter.println("</TD></TR>");
swriter.println("<TR HEIGHT=\"10%\" ALIGN=\"CENTER\">");
swriter.println("<TD><hr>ANKIT KUMAR SAHU 2013</TD></TR>");
```

```
swriter.println("</TABLE></BODY></HTML>");
}
catch(Exception myappexcptn)
{
        try{
        servletresponse.sendRedirect("error");
        }
        catch(Exception myappexcptn2)
        {
        System.out.println(myappexcptn2.getMessage());
        }
}
finally
{
        try{
        if(swriter!=null)
                swriter.close();
        }
        catch(Exception myappexcptn3)
        {
        System.out.println(myappexcptn3.getMessage());
        }
}
}
public void doPost(HttpServletRequest servletrequest, HttpServletResponse servletresponse)
{
        doGet(servletrequest, servletresponse);
}
}
```

6. ErrorHandlerServlet.java :

```
package myapps;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.PrintWriter;

public class ErrorHandlerServlet extends HttpServlet

{

public void doGet(HttpServletRequest servletrequest, HttpServletResponse servletresponse)

{

try{

        PrintWriter swriter=servletresponse.getWriter();;

        swriter.println("<HTML><BODY BGCOLOR=\"RED\">");

        swriter.println("<p><h2><BR><center> DUE TO SOME TECHNICAL PROBLEMS, WE ARE UNABLE TO SERVE YOU NOW<BR>PLEASE TRY AFTER SOME CLOCK CYCLES..");

        swriter.println("<br><a href=\"index.html\"> RELOAD </a>");

        swriter.println("</center></h2></p></BODY></HTML>");

        if(swriter!=null)

                swriter.close();

}catch(Exception myappexcptn)

{

        try{

        servletresponse.sendRedirect("index.html");

        }

        catch(Exception myappexcptn2)

        {

        }

}

}
```

```
public void doPost(HttpServletRequest servletrequest, HttpServletResponse servletresponse)
{
        doGet(servletrequest, servletresponse);
}
}
```

JAVA INSTALLATION:

First, to compile a java web application, Java SDK is needed, following steps explain:

- Download JDK6 (Java Development Kit 6.0) from this link:

  http://www.oracle.com/technetwork/java/javase/downloads/index.html

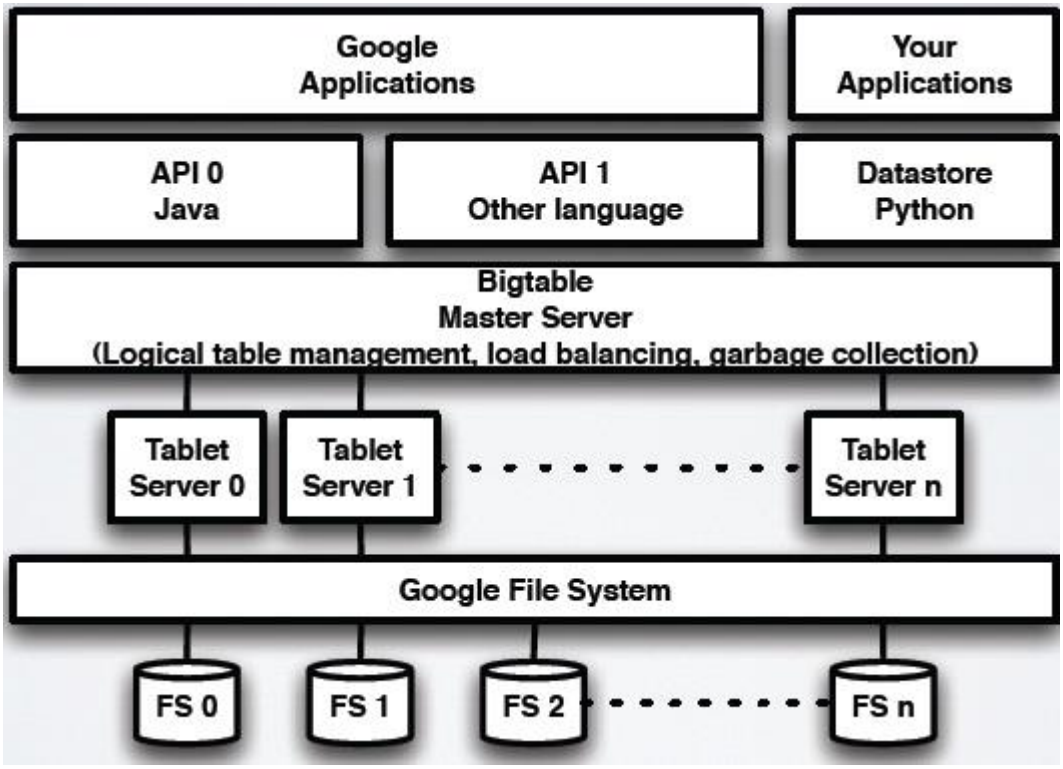
- Install the JDK for compilation of the java applications.

  You can test whether the Java development kit is installed on your system and check which version it is by running the following command at a command prompt (in Windows, Command Prompt; in Mac OS X, Terminal):

  ```
  javac -version
  ```

App Engine for Java includes support for two major standard interfaces for databases: Java Data Objects (JDO) and the Java Persistence API (JPA). Like the other standards-based interfaces in the App Engine Java API, using one of these interfaces makes it easier to move your application from and to another platform. JDO and JPA support different kinds of databases, including object databases and relational databases. They provide an object-oriented interface to your data, even if the underlying database is not an object store.

The JDO and JPA implementations are built on top of a low-level API for the App Engine datastore. The low-level API exposes all of the datastore's features, and corresponds directly to datastore concepts. For instance, you must use the low-level API to manipulate entities with properties of unknown names or value types. You can also use the low-level API directly in your applications, or use it to implement your own data management layer.



(Figure 11: Google Data-Store)

Data type	Java type
Unicode text string (up to 500 bytes, indexed)	`java.lang.String`
Long Unicode text string (not indexed)	`datastore.Text`
Short byte string (up to 500 bytes, indexed)	`datastore.ShortBlob`
Long byte string (not indexed)	`datastore.Blob`
Boolean	`boolean`
Integer (64-bit)	`byte`, `short`, `int`, or `long` (converted to `long`)
Float (double precision)	`float` or `double` (converted to `double`)
Date-time	`java.util.Date`
Null value	`null`
Entity key	`datastore.Key`
A Google account	`...api.users.User`
A category (GD)	`datastore.Category`
A URL (GD)	`datastore.Link`
An email address (GD)	`datastore.Email`
A geographical point (GD)	`datastore.GeoPt`
An instant messaging handle (GD)	`datastore.IMHandle`
A phone number (GD)	`datastore.PhoneNumber`

Steps to Deploy Java Web Application on GAE:

- To deploy java web application SampleJavaApp, An Application is needed to create on appengine.google.com and name could be same or different as our root folder. Whether it is same or different, it should be mentioned in appengine-web.xml.
- After creating the application on appengine.google.com, we need to provide a unique application identifier. This identifier would be used in accessing deployed web-application. URL to the deployed application would be:

    application_identifier.appspot.com

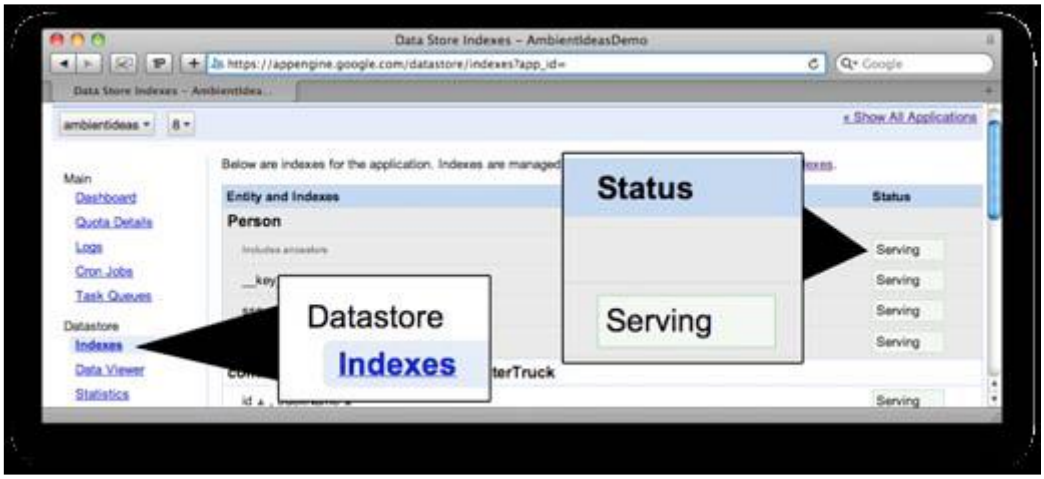
Suppose the identifier, we took for the application is: samplejavaapp and it is not taken by any other yet now, then our application's URL would be: samplejavapp.appspot.com.

- Using the appengine-java-sdk tools, the application can be uploaded on the created application on appengine.google.com. To upload application, the command is used as follows:

    appcfg.cmd update Path_To_The_SampleJavaApp

    Entering this command, the authentication would be asked for google account. After Successful authentication, the application will be deployed automatically on the google app engine.

- After the deployment on google app engine, our application is accessible on above given URL. The Administration for handling the application or to have a sight on the application is served on appengine.google.com.

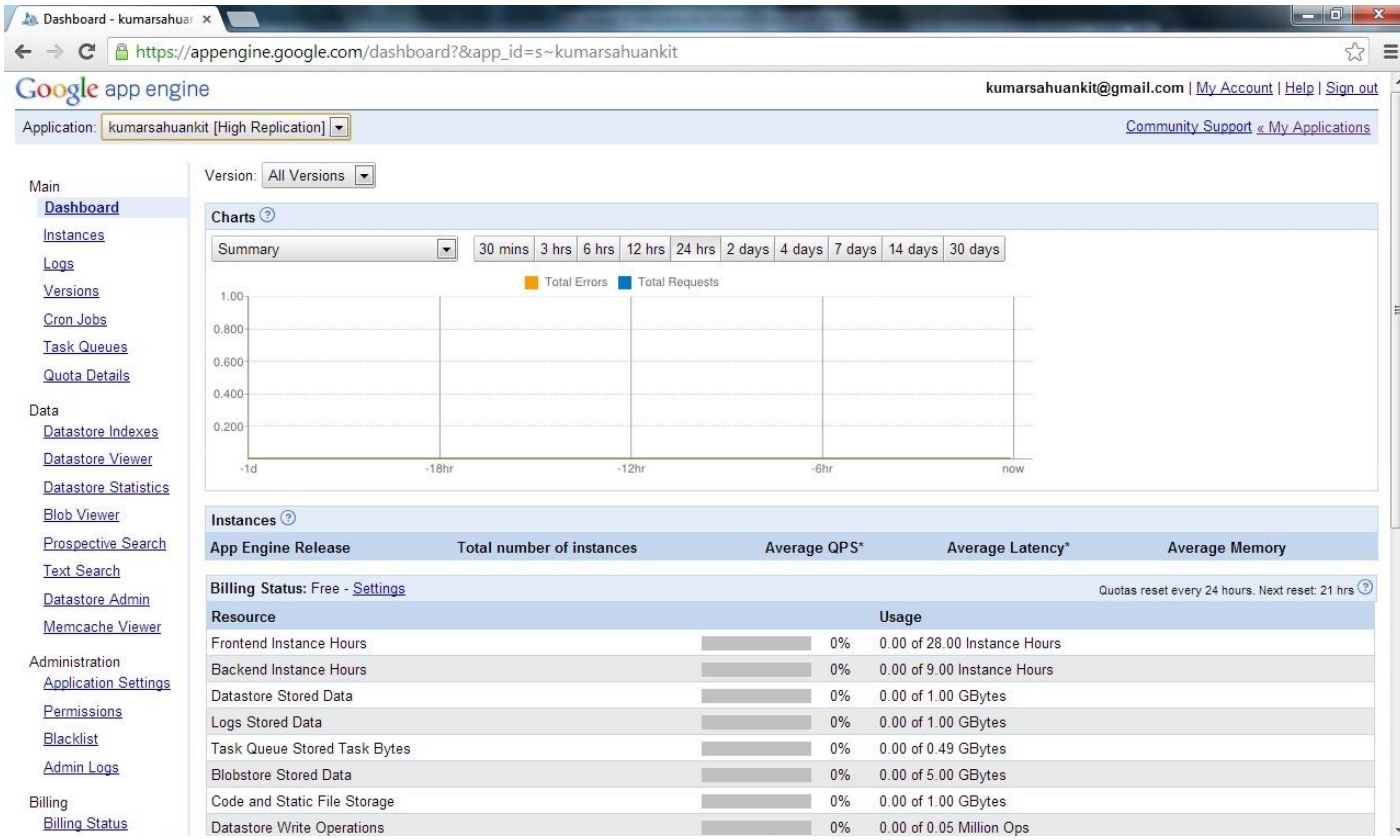


(Figure 12 (a): Data Store Indexes in GAE)

(Figure 12 (b): Application Overview in GAE)

Above image depicts the web-applications deployed on Google App Engine which is Home-Page. The Button "Create Application" is used to create application and then the application is uploaded. The Home Page includes Application name, Application Title, Storage Scheme and Status.

(Figure 12 (c): Data-Store Queries in GAE)



(Figure 12 (d): Dashboard of an Application in GAE)

These steps are used to deploy a java web-application on Google App Engine. Deployment on GAE has been easier and reliable. Java Web Applications need JRE to execute the Java-Application and GAE provides the required environment. Environment for Java Deployment is user-friendly. One can deploy 10 Applications on GAE in free of cost. To deploy more applications, the billing status should be activated. Google App Engine has been a good platform for new cloud users as well as experienced users.

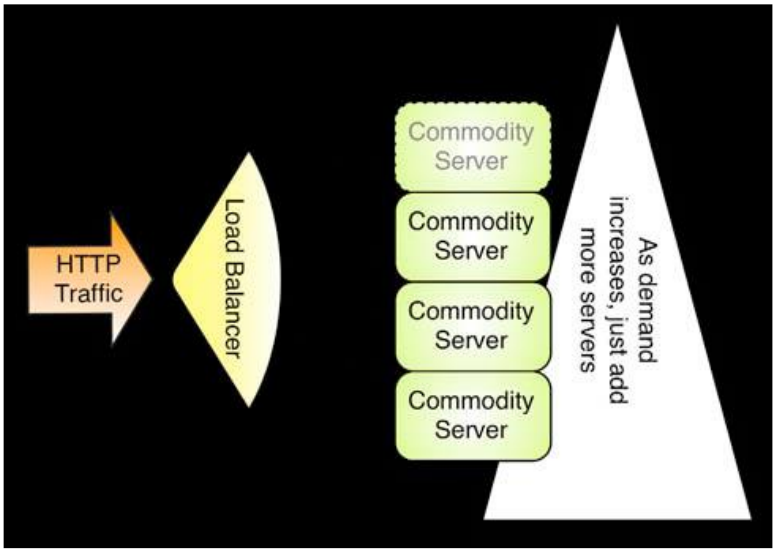
GAE has been a revolutionary cloud platform for Cloud users. GAE is a sub-part of Google Cloud. GAE is java and python application cloud within Google Cloud. Google Cloud provides a wide collection in the matter of cloud data.

In GAE, the application administration provides several admin-functionalities like Logs, Versions, Instances, Task Queues and Quota Details etc. In the Charts Panel, the application-load is depicted as graph. For using the database, one needs to pay a certain amount of money. A simple web-application is free to deploy on GAE without database. To use database, the Billing Status should be activated only through payment.

One can even queue his tasks, by using Task Queues for automation. It provides a better approach for automation the application management. Task Queuing has been beneficial in many purposes. Google App Engine queues tasks on the basis of their priorities.

3.2.3 REQUEST HANDLING IN JAVA WEB-APPLICATION ON CLOUDS

In a Java Web Application, the services are provided by Java Servlets. As we know that HTTP protocol is used for web applications, Servlets provide the HTTP services. Hyper Text Transfer Protocol serves in request and response. There exist multiple requests for each server and each response for each request. Request and response are the major terms in HTTP-Protocol. In java, there exists an interface named Servlet which is used to create a class as a Servlet. For HTTP-Protocol, the HttpServlet class is provided for the developers. HttpServlet class provides Get and Post methods to perform a task in a Servlet. Request is sent to a server from browser (client) and Response is sent to the browser (client) from server.

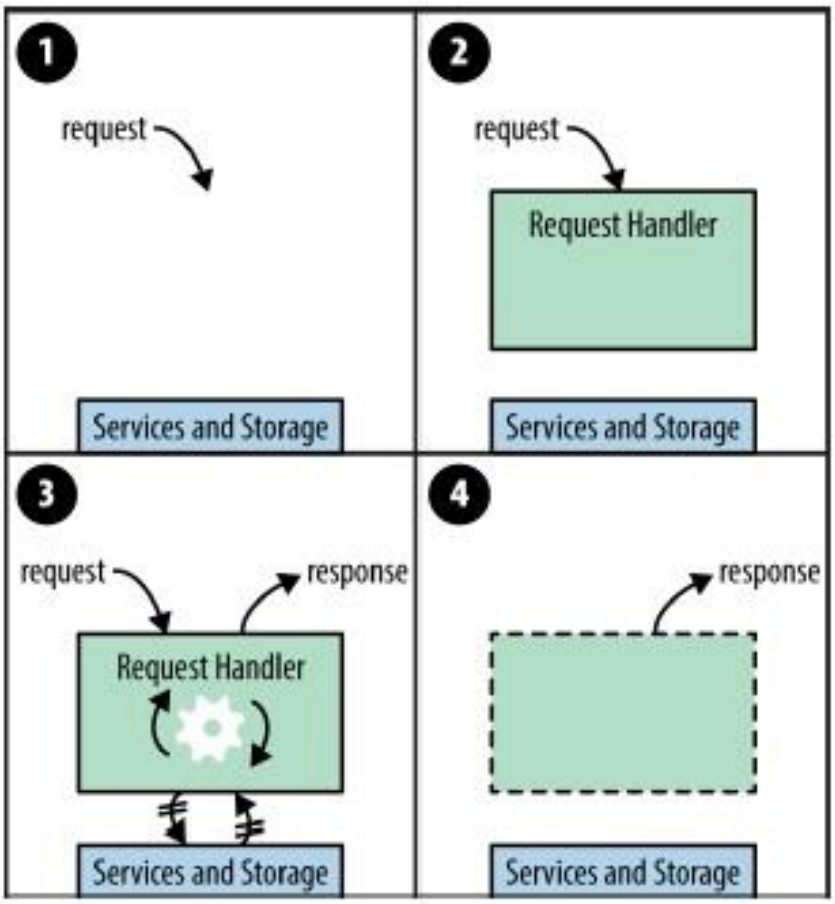


(HTTP Traffic Management)

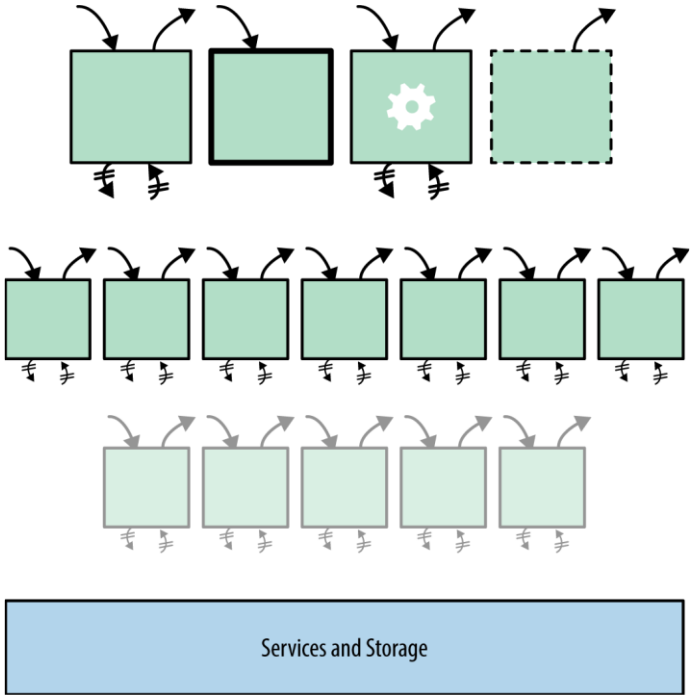
Feature	Limit
HTTP request size	10MB
HTTP response Size	10MB
Request or task duration	30 seconds
Maximum files in app	3000
Maximum size of all app files	150MB

Followings are the steps for Services in HTTP-Protocol:

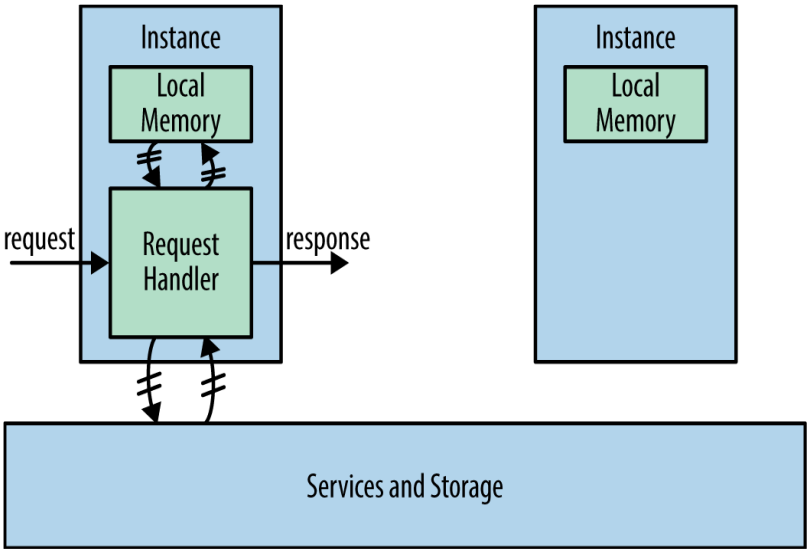
1. HTTP-Connection is established among client and server.
2. A request from client is sent to the Server (Services and Storage).
3. A request-handler is initiated and used to handle the request.
4. Request Handler fulfills the request and creates response using services.
5. The Request Handler is destroyed after sending the response to client.
6. Connection is closed.



(Figure 13 (a): Request-Response Cycle in HTTP-Protocol)

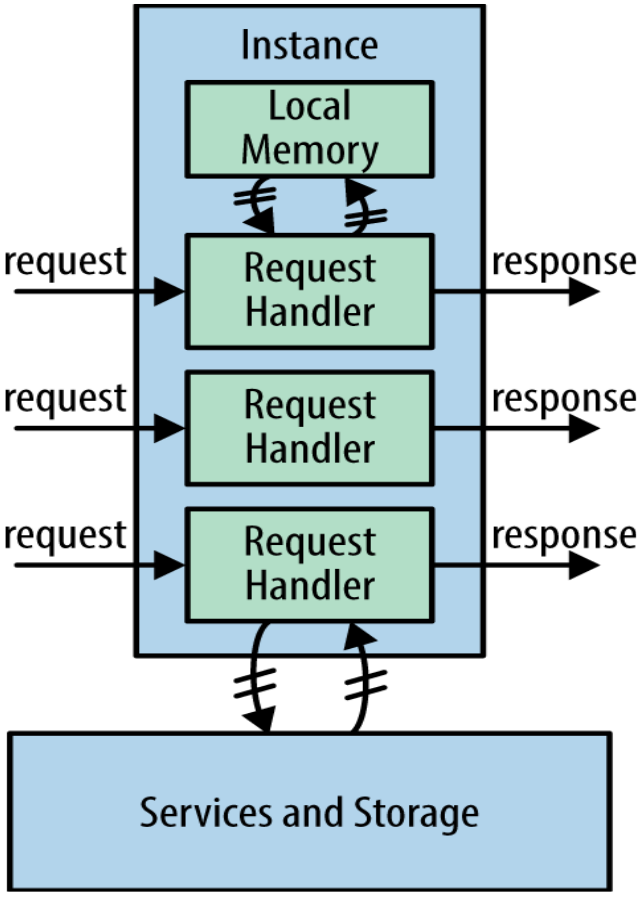


(Figure 13 (b): Multiple Requests to multiple request-handlers for single server)



(Figure 13 (c): Request handling at one instance and one ideal instance)

A Request is sent to a server which having two instances and the request is handled to an instance and the other one is idle



(Figure 13 (d): Multiple Request Handling)

More than one requests handled by single instance with using multiple threads

3.3 ISSUES IN JAVA APPLICATIONS ON CLOUD COMPUTING

As the java application is deployed on Clouds, there are several factors that affect the performance of the java-applications. Cloud Computing is a coming technology for the wide area of IT-industry, thus it should be an efficient and problem-free technology for a good survival. Now-a-days cloud computing is facing some challenges as compared to a non-cloud environment. These issues are major challenges for this coming technology. Each cloud provider needs to think over these issues for a better business in cloud computing. Issues which are troubling the cloud-environment consists cost, performance, reliability, security. Any technology which is expected to be adopted as wide area of the world should be problem-free thus cloud computing. All cloud platforms like EC2, GAE etc are facing these issues which are discussed further.

Issues that affect the cloud computing environments are:

i) Performance Issue,

ii) Cost Issue,

iii) Security Issue,

iv) Reliability Issue.

3.3.1 Performance Issue:

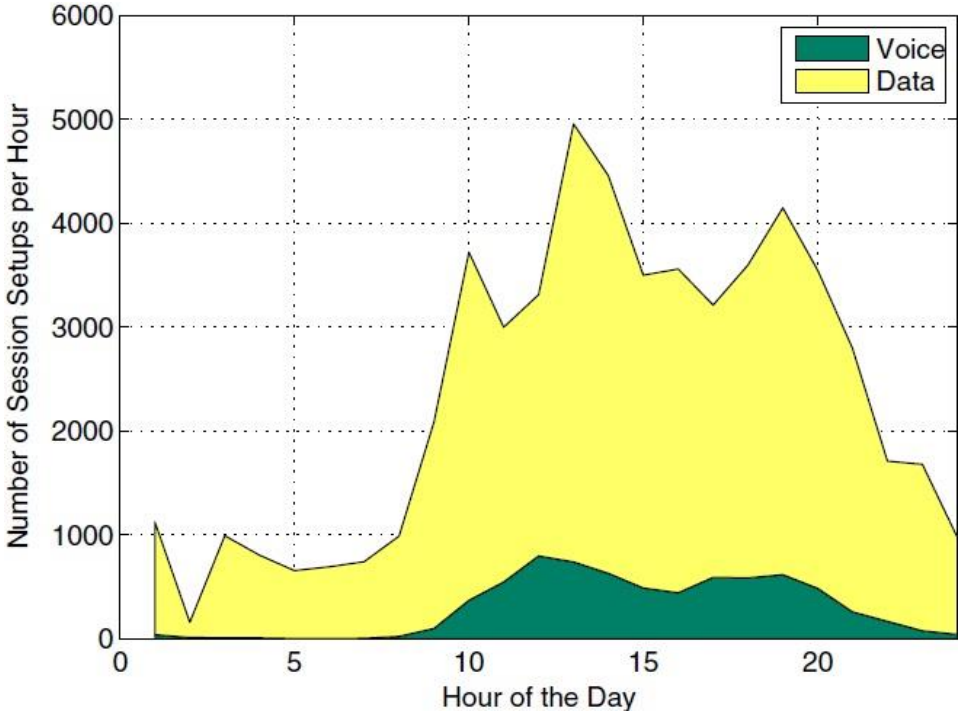
When a java application is deployed on a non-cloud environment and cloud environment, there is a performance difference in both situations. As we all know that java applications are developed using frameworks like spring, struts etc. These all frameworks are best suited for the non-cloud environment. These are not customized for cloud environment due to the current IT-industry requirements.

High Latency Delay is also a reason for the lesser performance of cloud environment as compared to non-cloud environment. In a cloud, all kind of data are stored as data-stores. The applications are also considered as a data for cloud. In a data store, all the applications that are java dependent use JRE to run the java applications. Cloud environment process the applications with Latency. Each application is provided a bandwidth. All requests are handled within the provided bandwidth.

The Performance issue has been an important issue in the matter of application-management. All kind of applications need to be entertained due to their performances in the opposite conditions too. This issue should be handled as soon as possible. The basic objective of this report includes the performance issue.

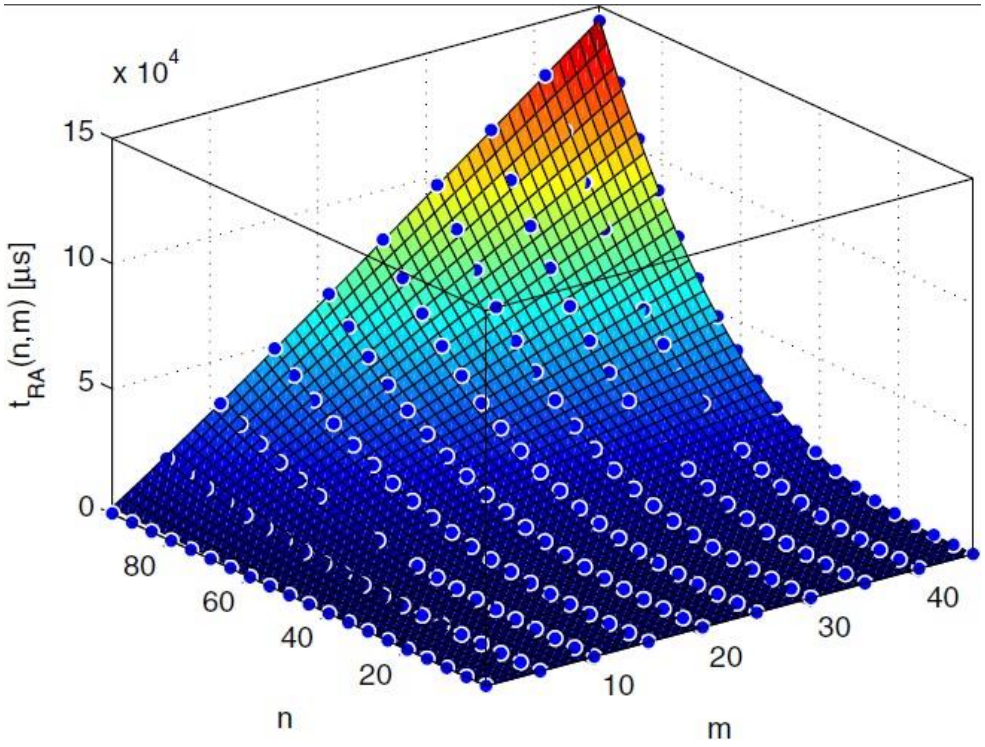
The Performance issue also includes SDR (Software Defined Radio) cloud latency delay which is explained as follows:

1.) The following image shows the relationship between voice and data over session per hour to illustrate the Hour of the day.



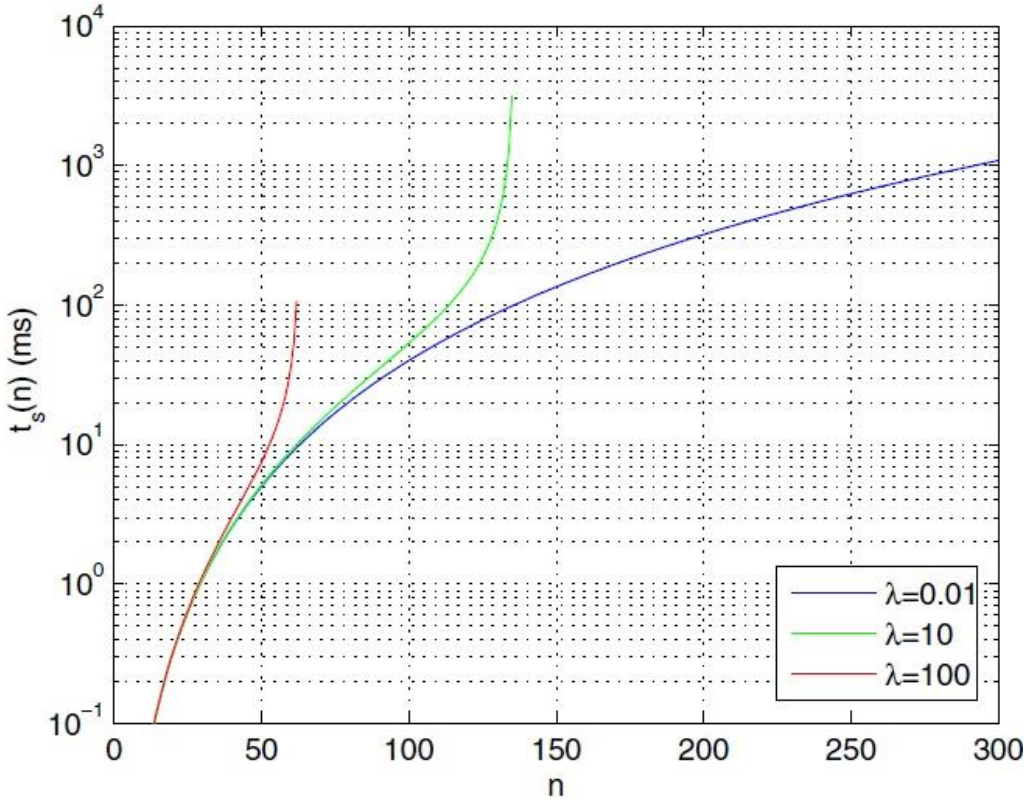
(Figure 14 (a): Relationship between voice and data)

2.) Following image depicts 3-D relationship in traffic management in Cloud.



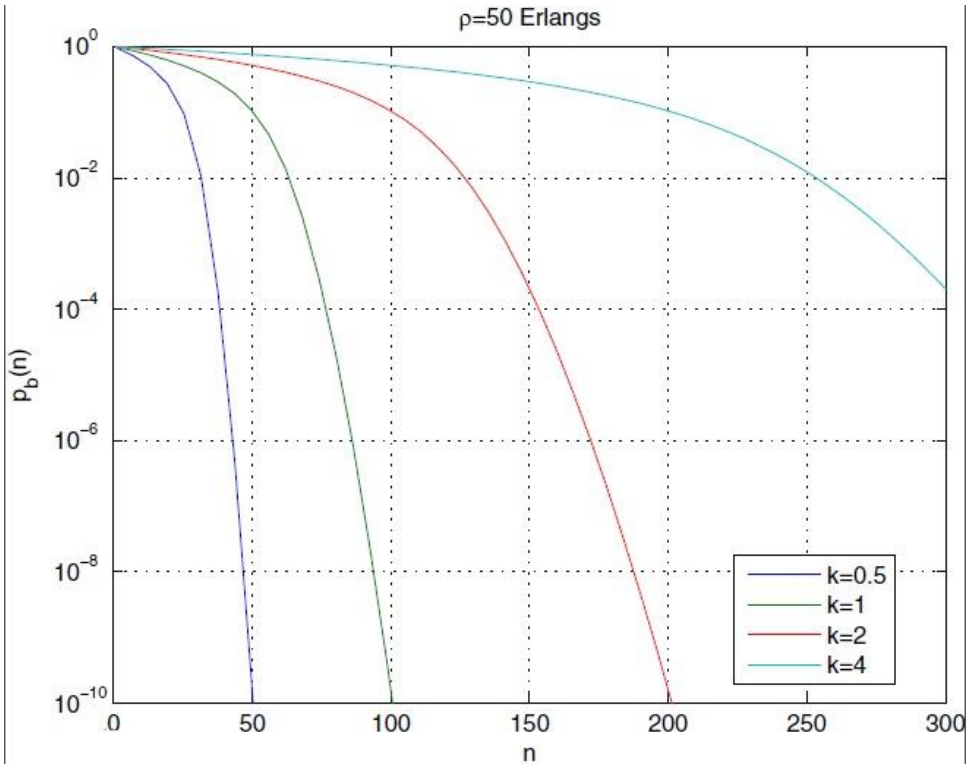
(Figure 14 (b): 3-D relationship in traffic management in cloud)

3.) Following graph shows wavelength as per the time(in milliseconds) to represent the latency delay



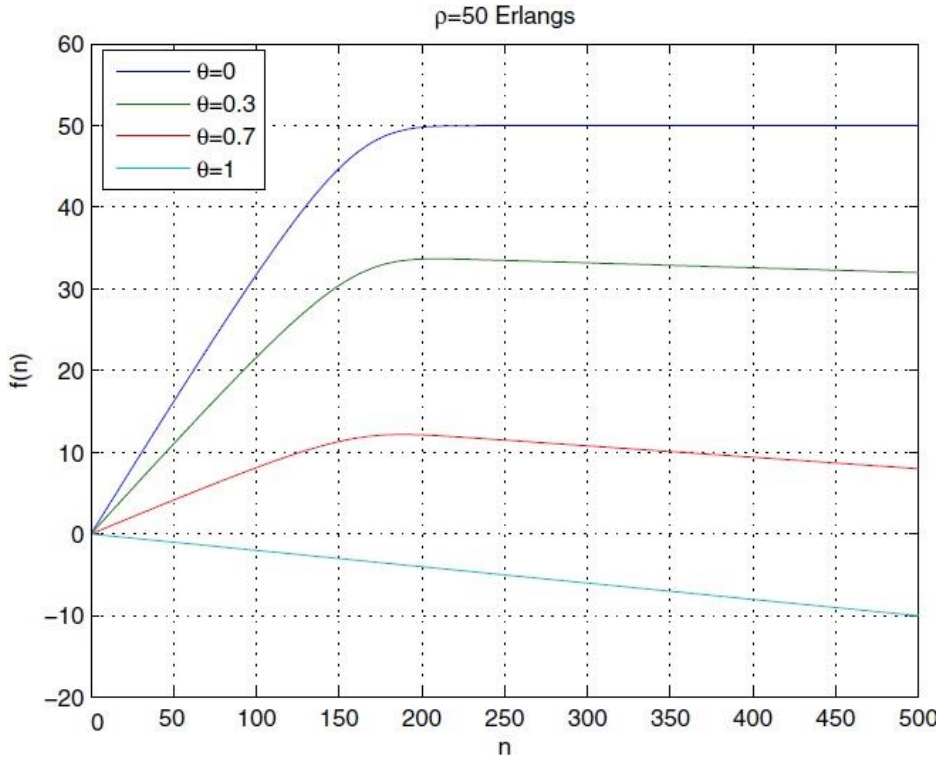
(Figure 14 (c): Graph between wavelength and time (High Latency Delay))

4.) Given graph shows relationship between k and p_o:



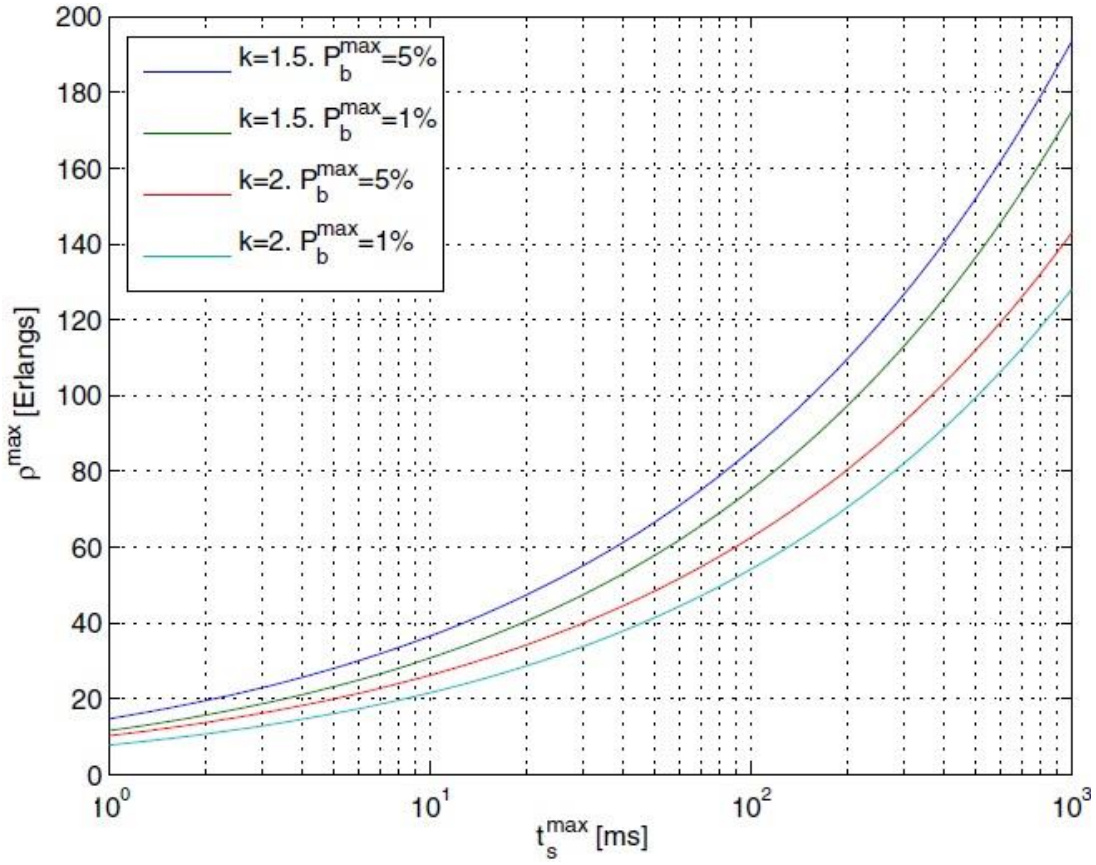
(Figure 14 (d): Relationship between k and p_o)

5.) Graph between latency delay and the specified time:



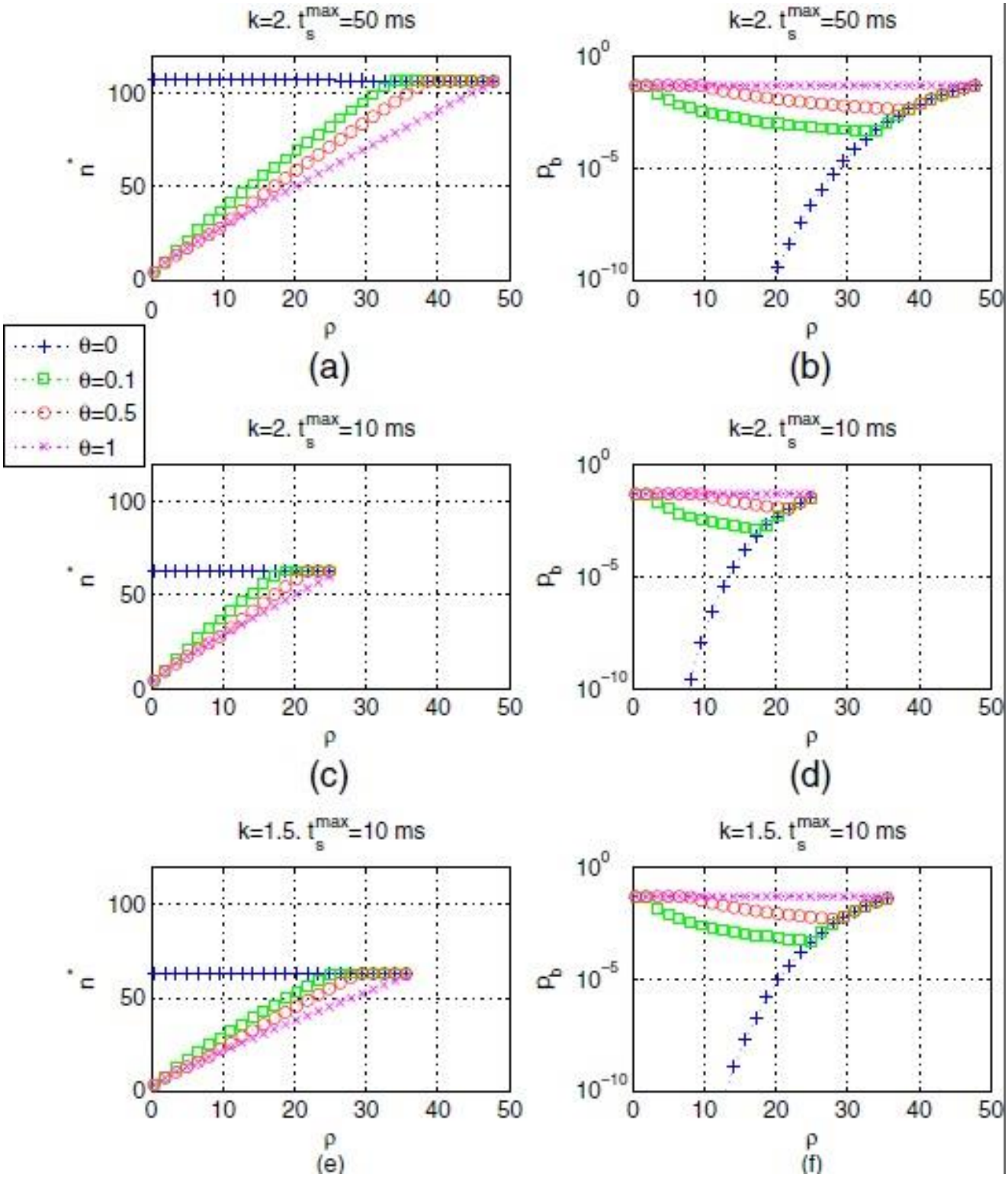
(Figure 14 (e): Graph between latency delay and the specified time)

6.) Graph between maximum latency and maximum time taken



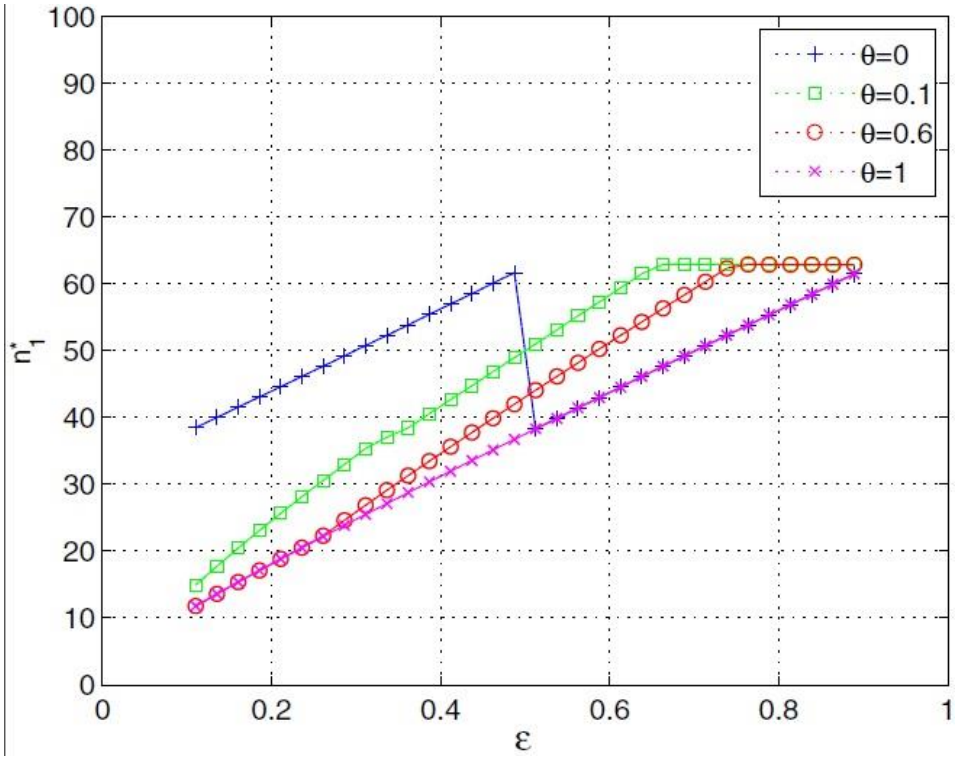
(Figure 14 (f): Graph between maximum latency and maximum time taken)

7.) Several graphs between latency and time with respect to time-slots:



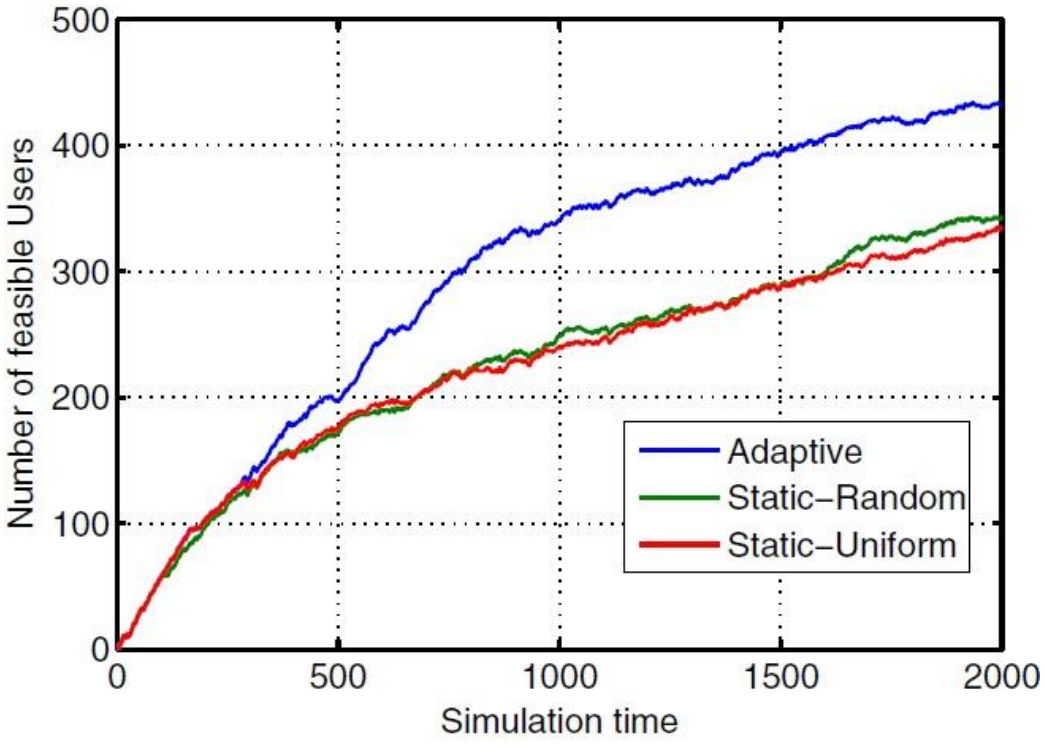
(Figure 14 (g): Graphs between latency and time with respect to time-slots)

8.) Graph between the traffic and latency delay:



(Figure 14 (h): Graph between the traffic and latency delay)

9.) Graph between simulation time and number of feasible users:



(Figure 14 (i): Graph between simulation time and number of feasible users)

Above Steps depicts the SDR Latency Delay which are responsible for Performance issue in cloud computing.

3.3.2 Cost Issue:

This issue includes the cost of the application management with the cloud management. These all kinds of cost are included in this issue. If cost is above expected, it also initiates the issue. This issue is important for a large scale in the IT-Sector. Cost-management is an important factor for this issue. This issue also affects the IT–Sector for their resource management techniques.

To solve these kinds of issue, one would have to optimize all the techniques for resource management and other management algorithms. Cost Issue makes the cloud computing a complex environment for coming time to replace all the technology in cloud environment as cloud computing.

3.3.3 Security Issue:

Security has been a crucial problem in all fields in IT-Sector. One cannot be sure in the security prospectus in any manner. This issue can be resolved at a particular level but not at the best level. One cannot guarantee the system to be the best secured system. In cloud computing, security level is very high. Cloud stores the data and handles all the resources. So the security in cloud computing requires security at the cloud level as well as the network through which the systems are connected in cloud computing.

Cloud Computing requires security at cloud level due to the data and resources which lie on cloud. Cloud computing in the secured manner can be the better IT-technology for the coming time in IT-Sector. For making a secured system, one wants the algorithm and the technology to be the best in security manner. Java is a example of the best secured technology in Web-application.

3.3.4 Reliability Issue:

Applications over the clouds are expected to be the reliable for both users and developers. The Cloud Computing is reliable for developers but it has not been a good reliable application for users as it should be due to their application specific properties.

Reliability is a good feature for applications for their long time execution and life cycle. Reliability issue is indirectly connected to the performance issue. Both are respectively connected to each-other. This issue can be resolved by eliminating the performance issue. Both issues are for users. Developers are provided reliable features in cloud computing but this reliability has no access to the users.

These issues in Cloud Computing has been a major problem in cloud computing. These four issues, Performance, Cost, Security and Reliability issues are a big challenge for cloud computing to be adapted in the IT-Sector for its positive aspects which can only be gained by eliminating these issues. To resolve these issues, there could be several methods like using the best optimized and secured algorithms and techniques.

3.4 PERFORMANCE ISSSUE IN JAVA APPLICATIONS IN CLOUD COMPUTING

The working area of this research report is the performance issue in java web-deployment which is explained as follows:

- Performance issue which is discussed earlier has been a major issue in user prospectus as well as the developer prospectus.
- A java web application deployed on cloud, expected to be the most appropriate application in secured manner but it should be reliable too.

To develop java web applications, frameworks are used. Spring and struts framework are most popular of them. Spring Framework has been customized for the cloud environment like GAE to provide a better performance in clouds. Spring Framework has released a specific version for developing the cloud applications on java. The java applications are framework specific. If a application is developed using a particular framework, it also depends on the performance, reliability and security on the respective framework.

The Struts framework (used at a large scale for java web applications) has not customized itself for clouds. It affects the performance of the application deployed on the cloud. The Struts framework is best suited framework for non-cloud environment which has been used yet now at a large scale. But now, to use the cloud environment, one can't use the same framework. Struts framework (based on the MVC Architecture-Model View Controller Architecture) cannot be used for java application on clouds with a good performance until unless it customizes itself like spring framework.

A Java Application which is deployed on a cloud expected to be an application with good performance irrespective of the framework used in it. As a user view, applications need to be reliable to use as well as lesser time taking in load. To fulfill these kinds of expectations, developers need to resolve these major issues like performance issues.

High Latency delay is another reason for performance issue. When a request is received at cloud for a particular service, the response sent to the client should be very lesser time taking respectively to the service provided. The higher difference between request received and response sent is proportional to High Latency Delay. For an ideal system, High Latency Delay should be zero. But in Real Time Systems, High latency Delay can never be zero due to the traffic load and number of requests arrival at a time.

Component Scanning is also responsible for lesser performance of java application on cloud. There are limited numbers of resource available at a cloud to serve clients. When a client requests for a particular resources and the resource is free, it is occupied by the client for a particular time-slot and the resource is marked as occupied in the system, it is called locking, in which the resource is locked for a particular time to a client. If another client request for the same resource in the locking period, it has to wait for a time-slot. To examine the resource availability, the components are scanned, thus it is also known as Component Scanning. It also reduces the performance of the application.

Thus there are several factors that affect the performance of a java application over the clouds. The working area of this research paper is the performance issue in cloud computing.

To examine the performance issue, the web-application which has been deployed on Google App Engine earlier, would be useful. To examine this kind of issue, one needs to examine the difference between the application deployed on cloud environment and non-cloud environment.

First, the application deployed on cloud environment should also be deployed on non-cloud environment and then the web-page load checker tool can be used to find the difference of the both. There are several online tools which can be used to examine the webpage loading time. Some of these are explained further. As a sample application is deployed on java-cloud of GAE, the examination of the application would be done through these online tools.

Steps to examine the website performance:

- Provide the URL of both cloud environment to the application and non-cloud environment to the application to these tools.
- As results, these tools provide the time required to load the web-page at a single request.

Using the results, the performance can be analyzed of these two environments (cloud and non-cloud environments). While comparing the two environments in the matter of their request processing time, it also initiates the application assessment tools.

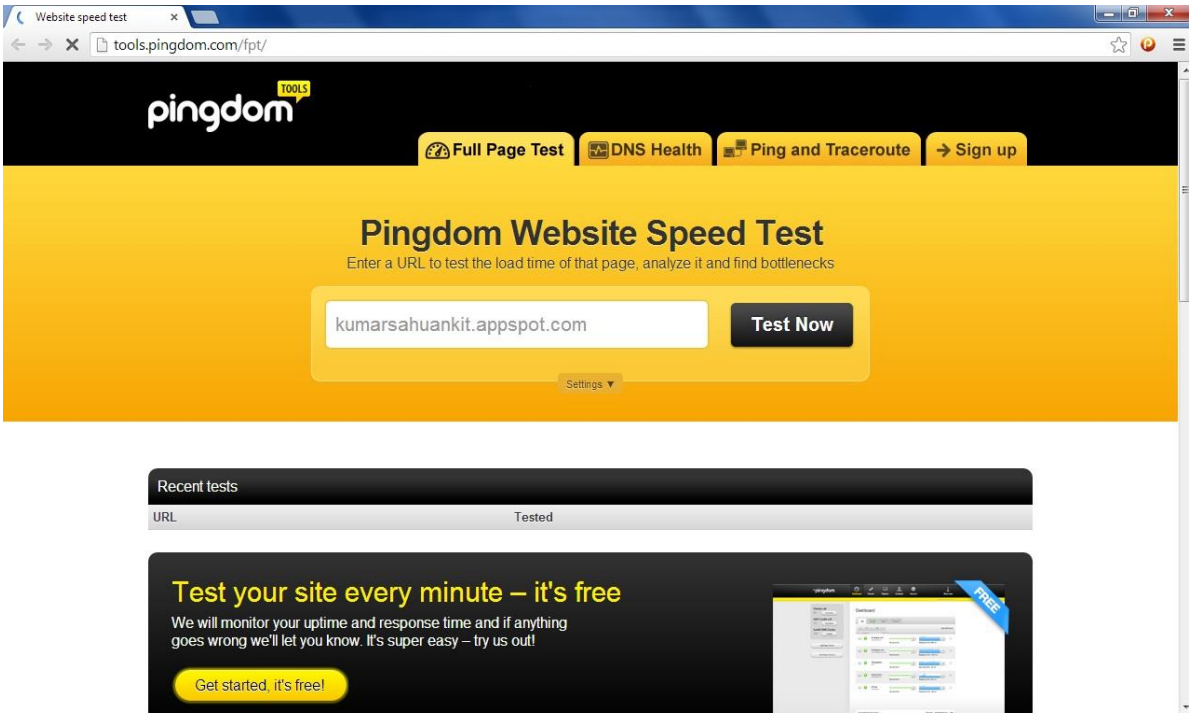
Some of the online tools for webpage load checker tools are named here:

Pingdom, webpagetest, gtmetrix, iwebtool etc.

1.) Pingdom: In Pingdom, one needs to input the website-address in specified textbox to website load check. The following image represents input panel of the pingdom. The sample application deployed on java-cloud GAE named kumarsahuankit.appspot.com is checked further on Pingdom.

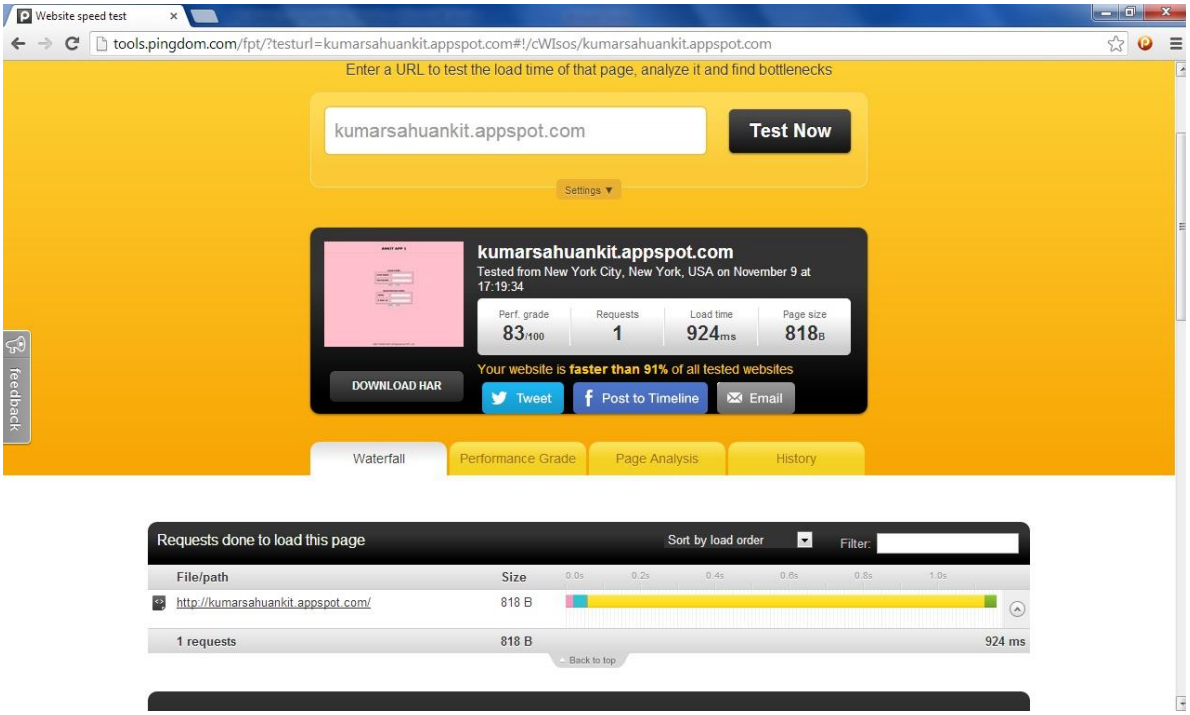
CLOUD ONLOAD TIME OF APPLICATION ACCESSING: NEARLY 1.5Seconds

NON-CLOUD ONLOAD TIME OF APPLICATION ACCESSING: NEARLY 200ms



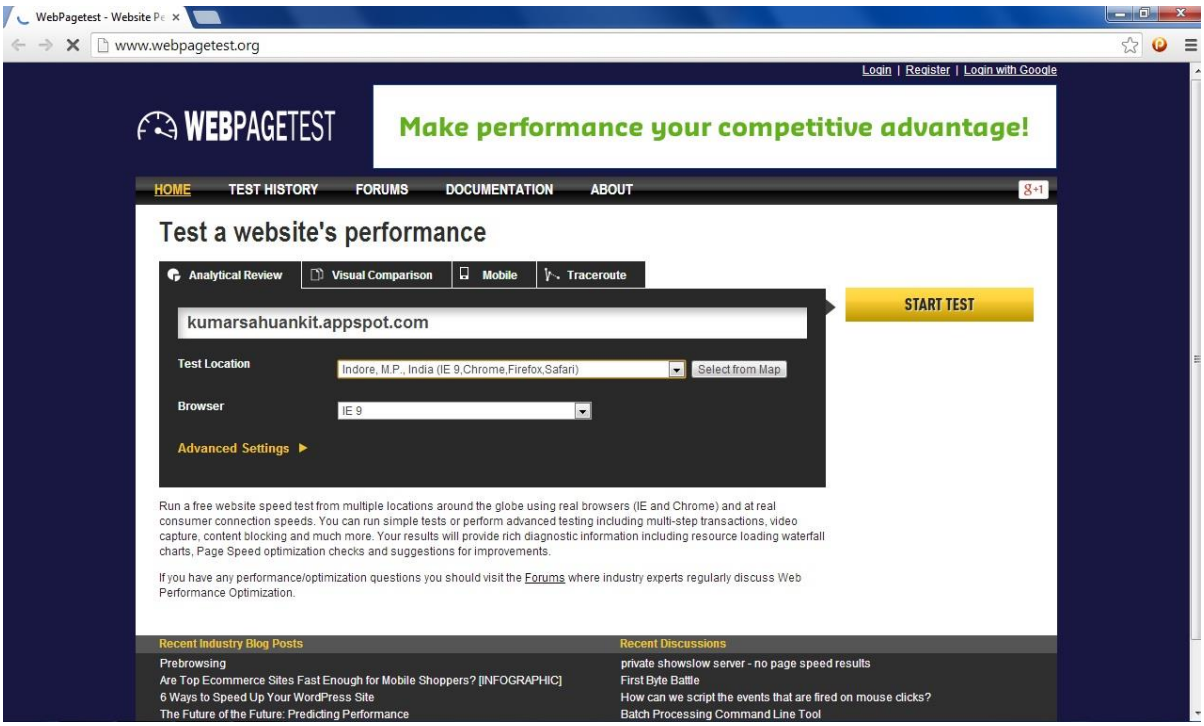
(Figure 15 (a): Online Website load Checker-Pingdom Input Panel)

Following image represents output of the web-checking on Pingdom. The Load Time according to Pingdom is 924ms. And the expected time taken on non-cloud environment is 210ms which is lesser than the cloud environment GAE.

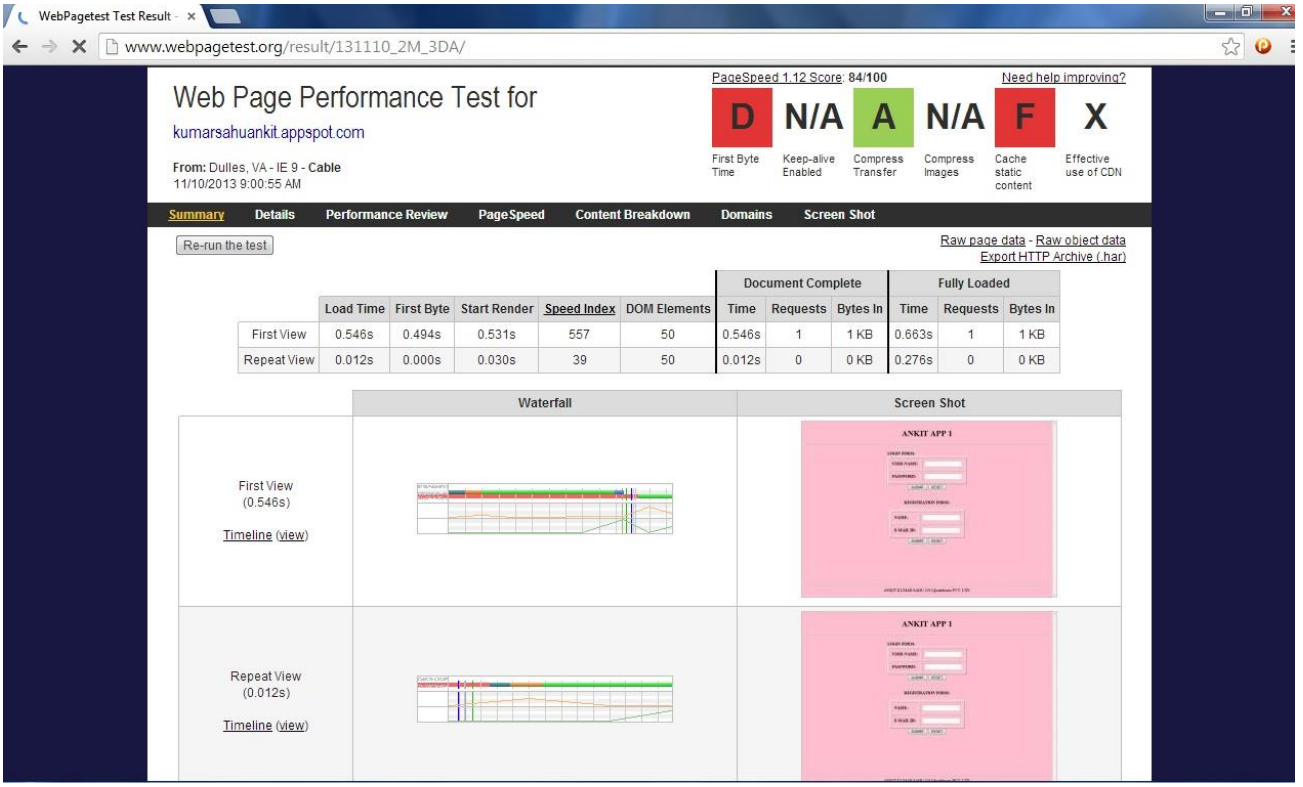


(Figure 15 (b): Online Website load Checker-Pingdom Output Panel)

2.) Webpage-test



(Figure 16 (a): Online Website load Checker-Webpagetest Input Panel)

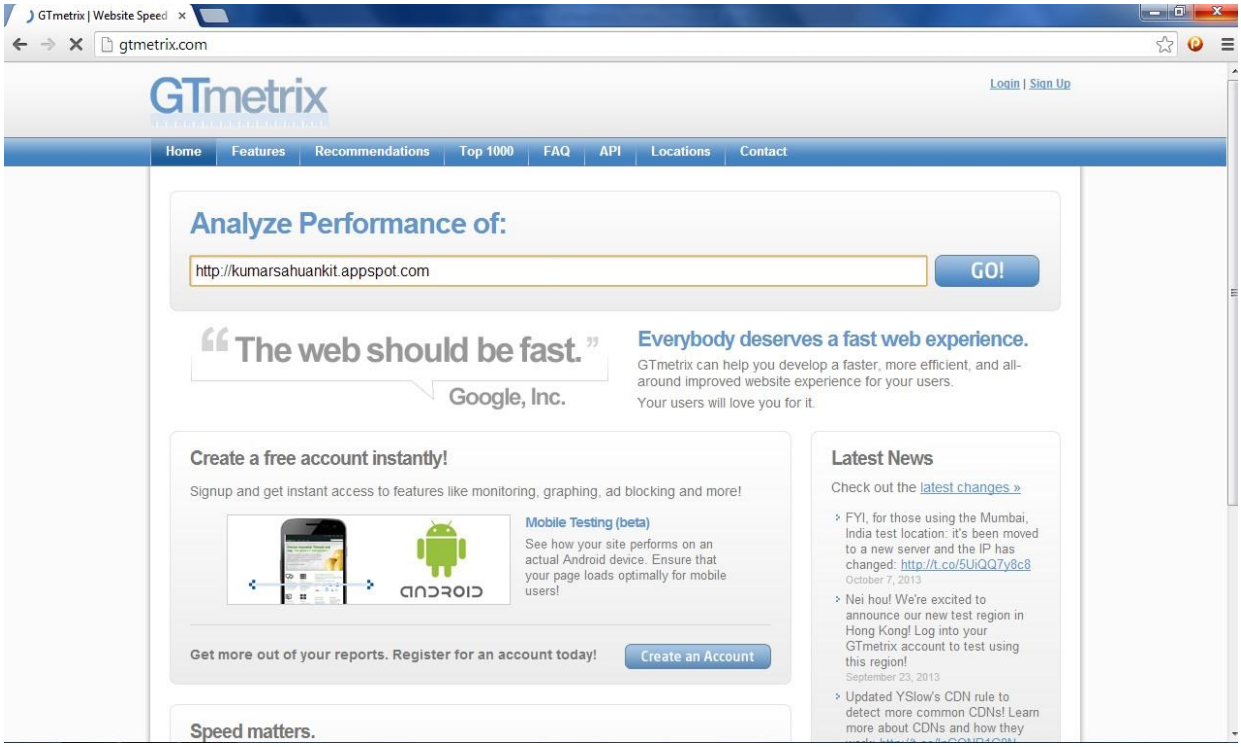


(Figure 16 (b): Online Website load Checker-Webpagetest Input Panel)

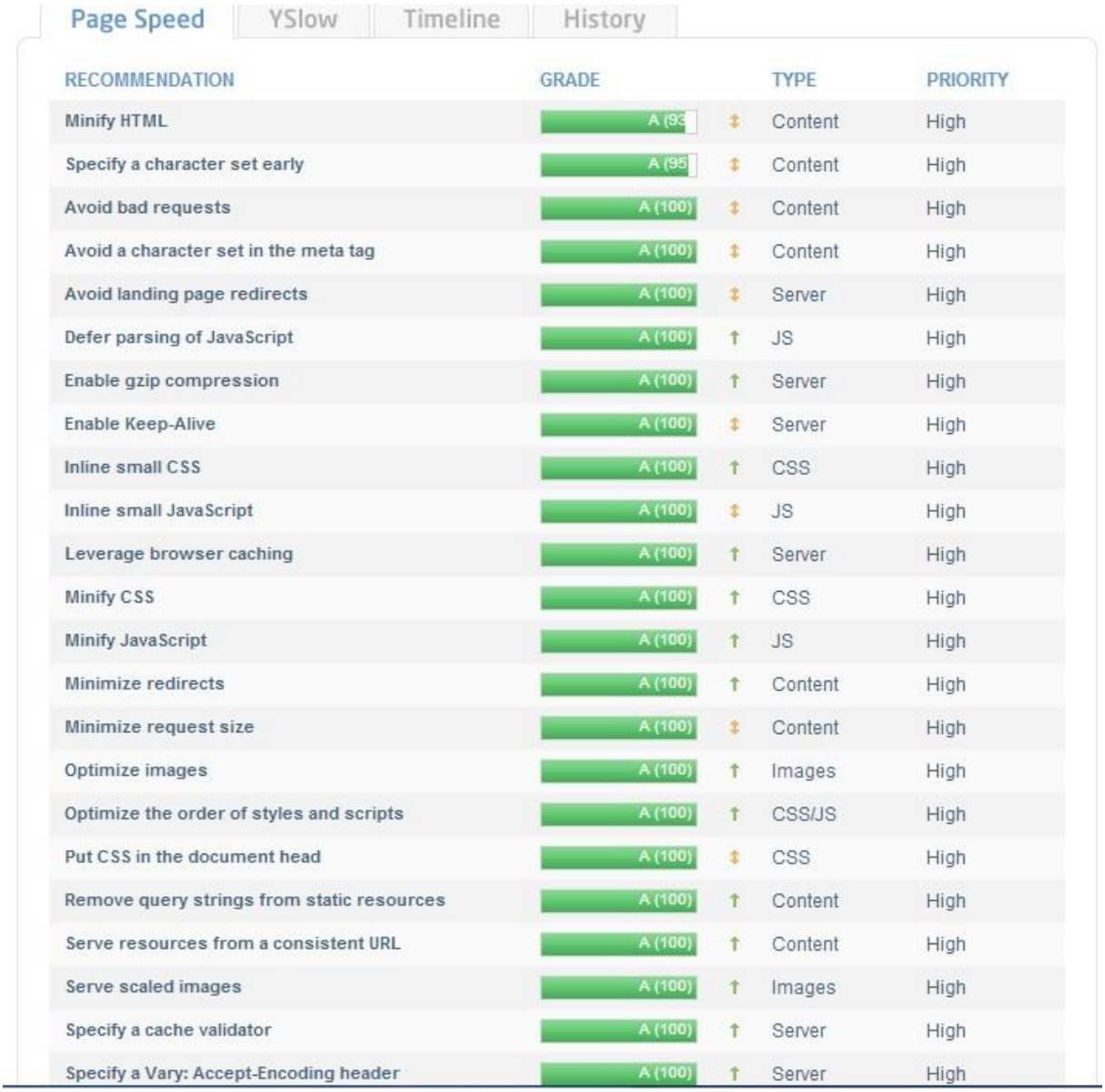
Output of web-checking: load time: 0.553s

3.) Gtmetrix

Input Panel:



(Figure 17 (a): Online Website load Checker-Gtmetrix Input Panel)



Page Speed | YSlow | Timeline | History

RECOMMENDATION	GRADE	TYPE	PRIORITY
Minify HTML	A (93)	Content	High
Specify a character set early	A (95)	Content	High
Avoid bad requests	A (100)	Content	High
Avoid a character set in the meta tag	A (100)	Content	High
Avoid landing page redirects	A (100)	Server	High
Defer parsing of JavaScript	A (100)	JS	High
Enable gzip compression	A (100)	Server	High
Enable Keep-Alive	A (100)	Server	High
Inline small CSS	A (100)	CSS	High
Inline small JavaScript	A (100)	JS	High
Leverage browser caching	A (100)	Server	High
Minify CSS	A (100)	CSS	High
Minify JavaScript	A (100)	JS	High
Minimize redirects	A (100)	Content	High
Minimize request size	A (100)	Content	High
Optimize images	A (100)	Images	High
Optimize the order of styles and scripts	A (100)	CSS/JS	High
Put CSS in the document head	A (100)	CSS	High
Remove query strings from static resources	A (100)	Content	High
Serve resources from a consistent URL	A (100)	Content	High
Serve scaled images	A (100)	Images	High
Specify a cache validator	A (100)	Server	High
Specify a Vary: Accept-Encoding header	A (100)	Server	High

(Figure 17 (b): Online Website load Checker-Gtmetrix Output Panel)

load time: 0.44s

4.) Iwebtool

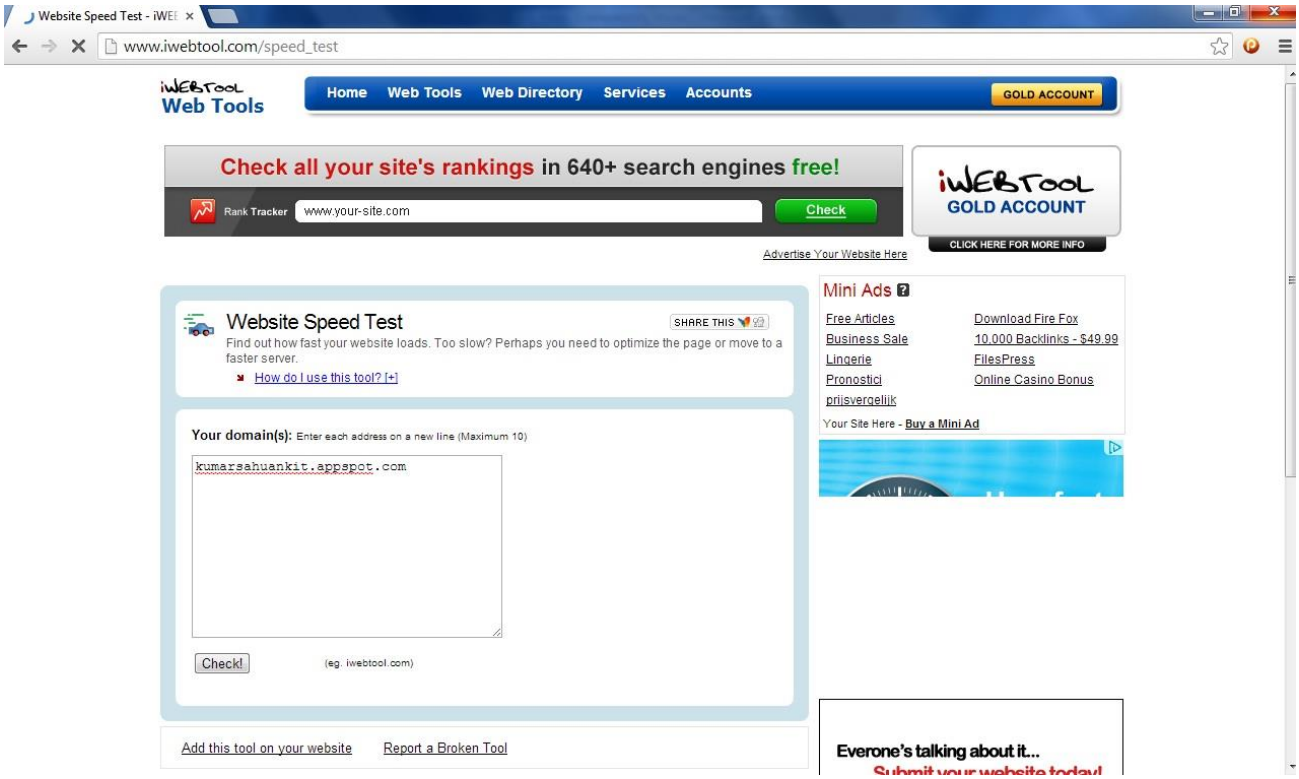


Figure 18 (a): Online Website load Checker-iwebtool Input Panel

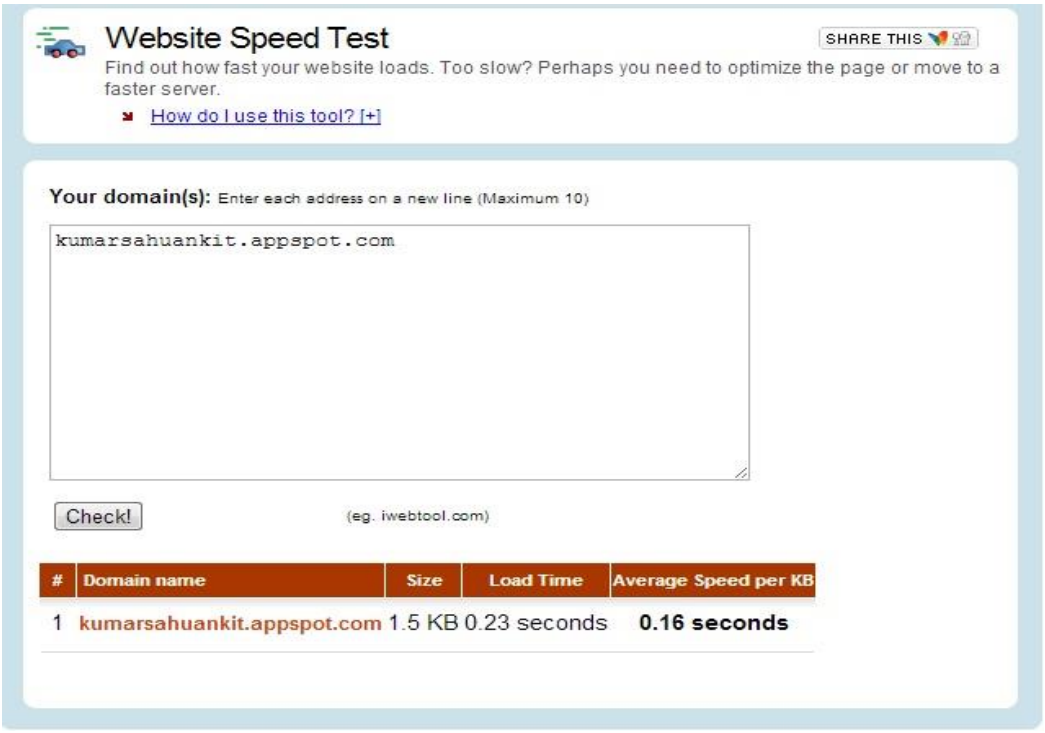


Figure 18 (b): Online Website load Checker-iwebtool Output Panel

Load time: 0.23 second

Using all these web-checkers, the performance issue is clear to cloud environments as compare to non-cloud.

There are several statics of the existing java clouds along their performances established in USA (table 1 and table 2)

Table I: Mean provisioning time for a simple deployment with no plugins.

	2 Nodes	4 Nodes	8 Nodes	16 Nodes
Amazon	55.8 s	55.6 s	69.9 s	112.7 s
Magellan	101.6 s	102.1 s	131.6 s	206.3 s
Sierra	371.0 s	455.7 s	500.9 s	FAIL

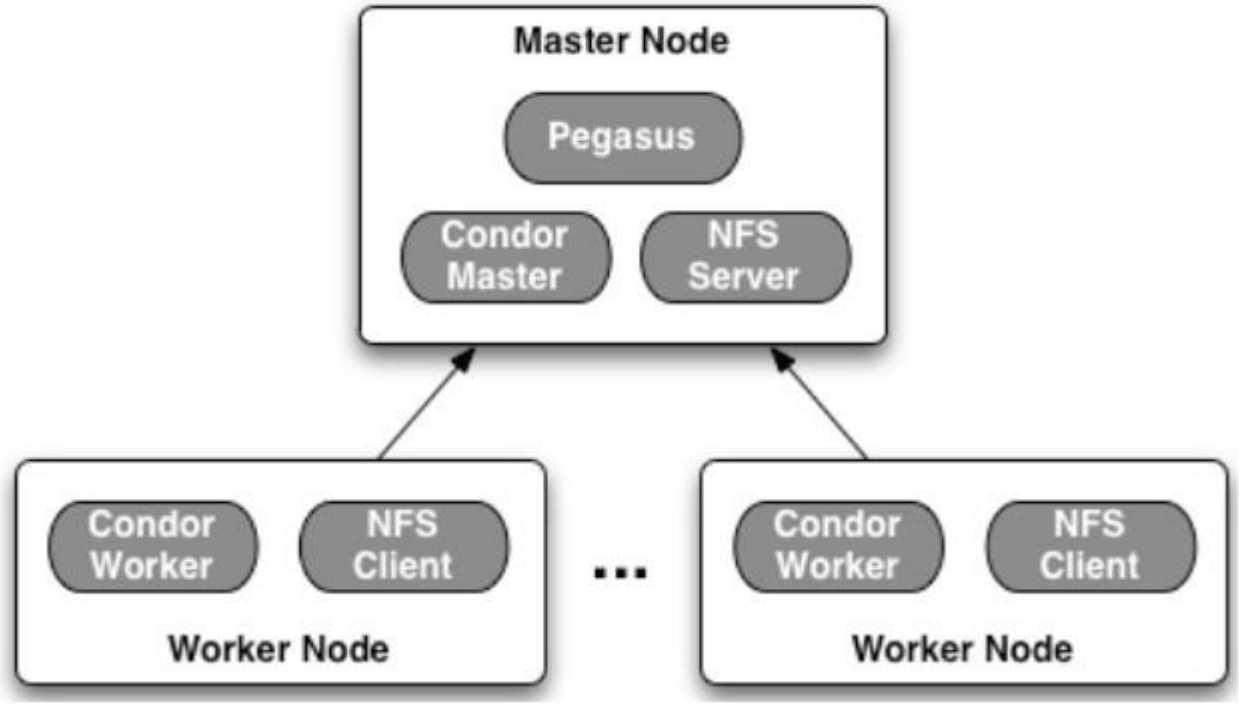
Table I represents the datasets of all the possible nodes along several cloud platforms depending on their type of cloud managements. Average time of the application is specified in this table. Sierra is failed in 16 Nodes due to the limited range of the present nodes.

Table II: Provisioning time for a deployment used for workflow applications.

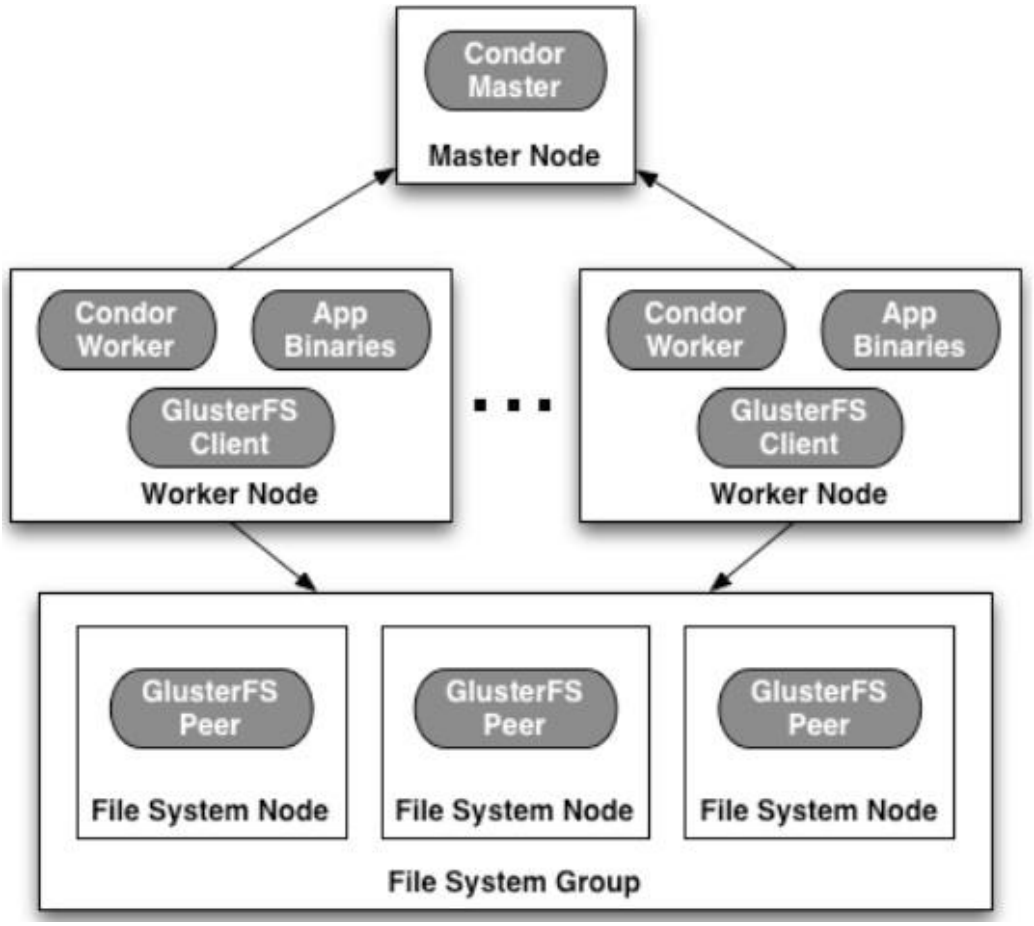
	2 Nodes	4 Nodes	8 Nodes	16 Nodes
Amazon	101.2 s	111.2 s	98.5 s	112.5 s
Magellan	173.9 s	175.1 s	185.3 s	349.8 s
Sierra	447.5 s	433.0 s	508.5 s	FAIL

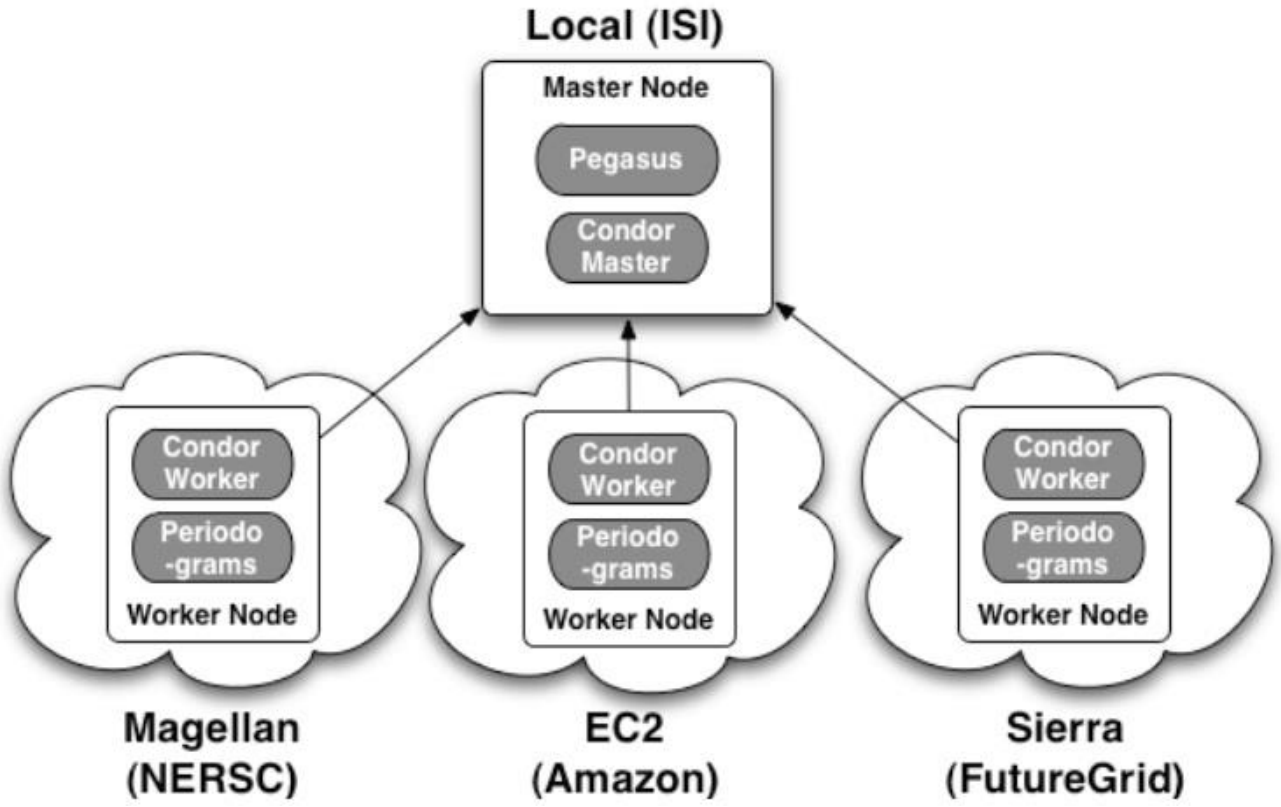
Table II represents the second request load cycle as same as table-I.

Architecture Comparison between several providers in performances



(This diagram depicts the connection between master and worker node. Pegasus, Condor and NFS are parts of the master node, as well as clients.)

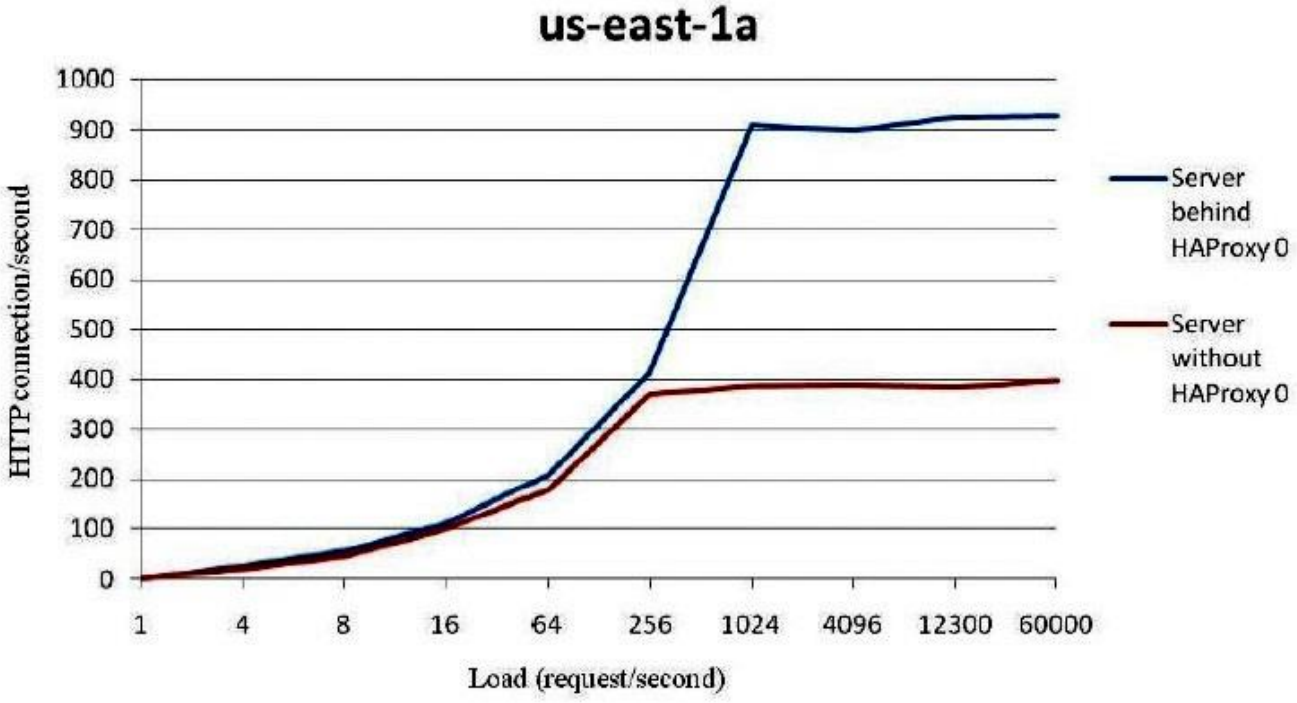




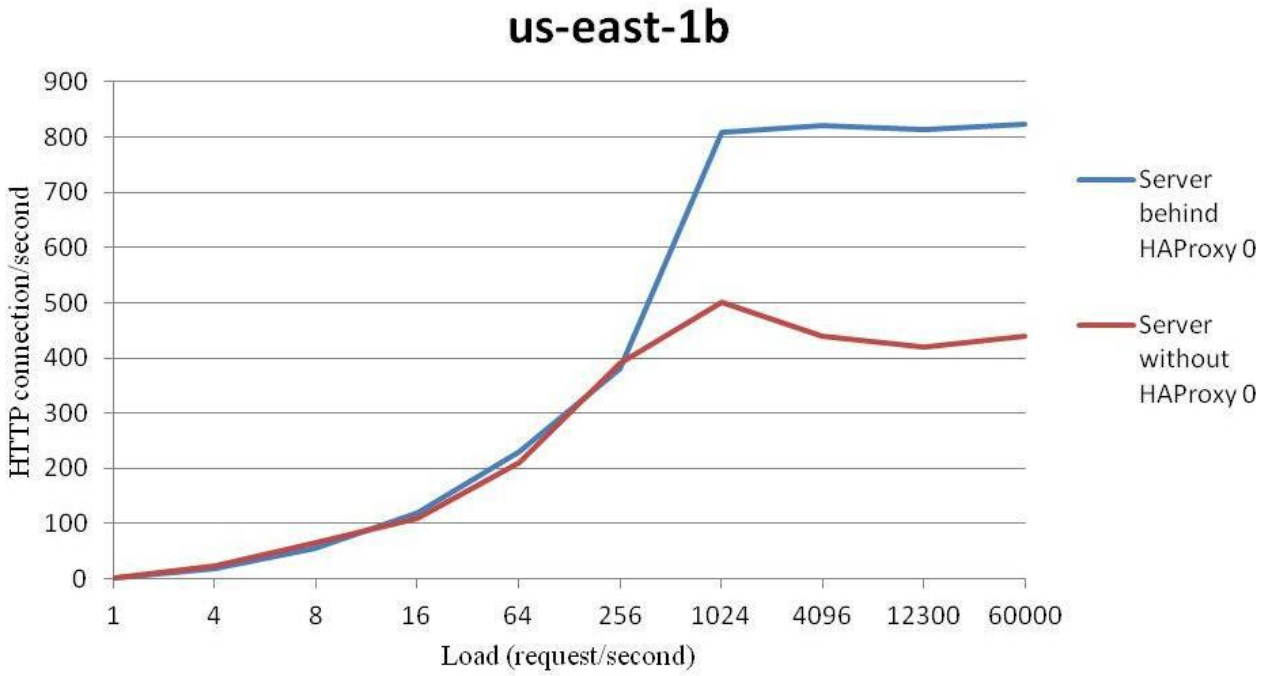
(Figure 19: Architecture Comparison between several providers in performances)

Traffic handling on a US-Server on cloud:

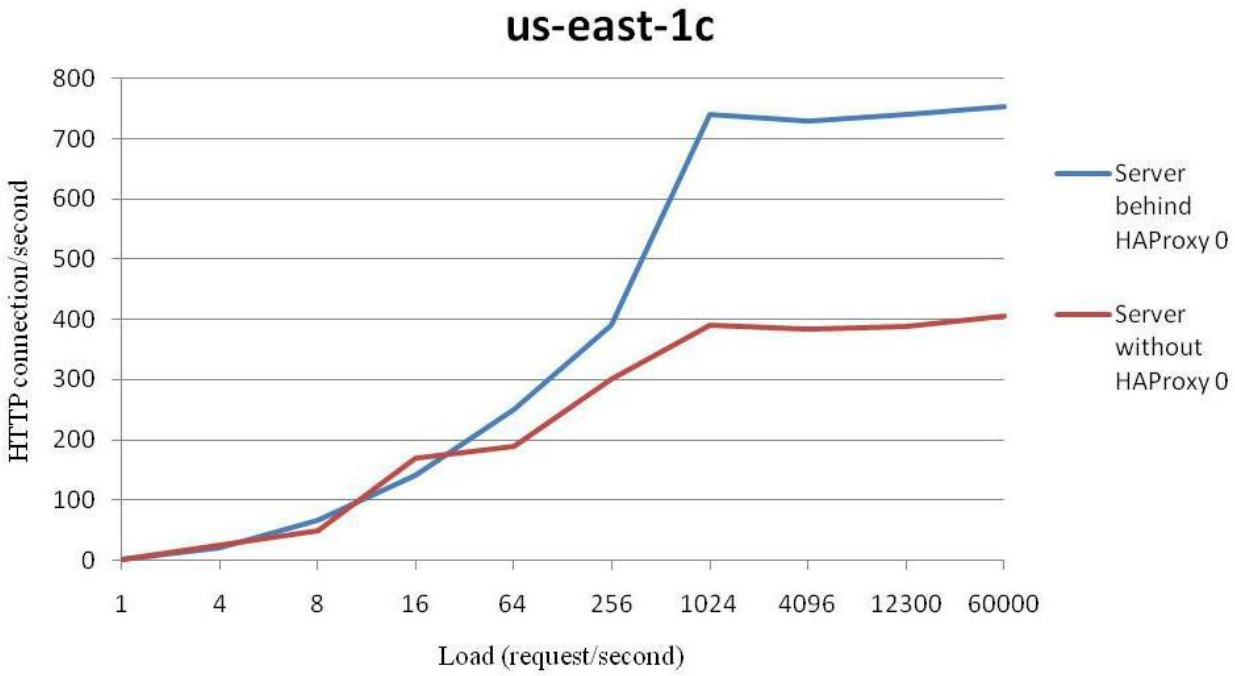
A Cloud established in USA, has been examined for the performance checking. It shows three kinds of servers. US-East-1a, US-East-1b, US-East-1c. These three separate servers indicates several connections as per their requests. These are as follows:



(**Figure 20 (a): Graph between load and HTTP connection/second of US-East-1a)**



(Figure 20 (b): Graph between load and HTTP connection/second of US-East-1b)



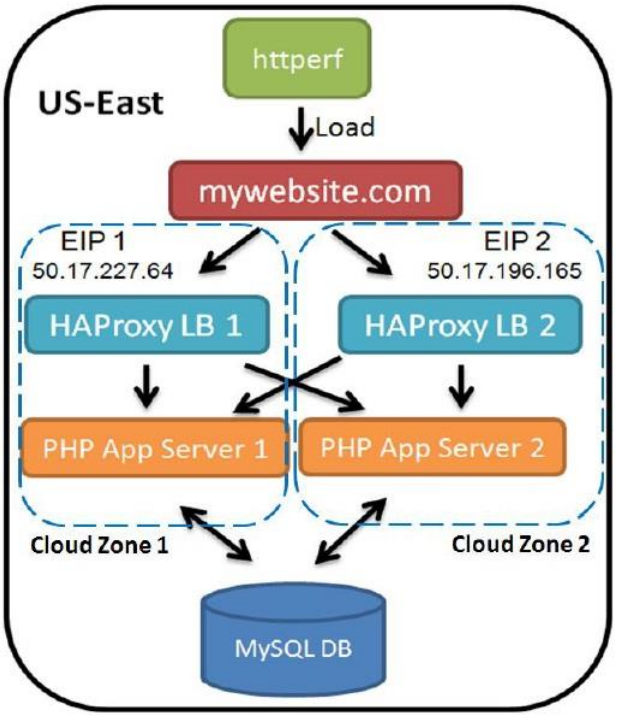
(Figure 20 (c): Graph between load and HTTP connection/second of US-East-1c)

US East-1A shows the graph between request/second and HTTP connections in server-1.

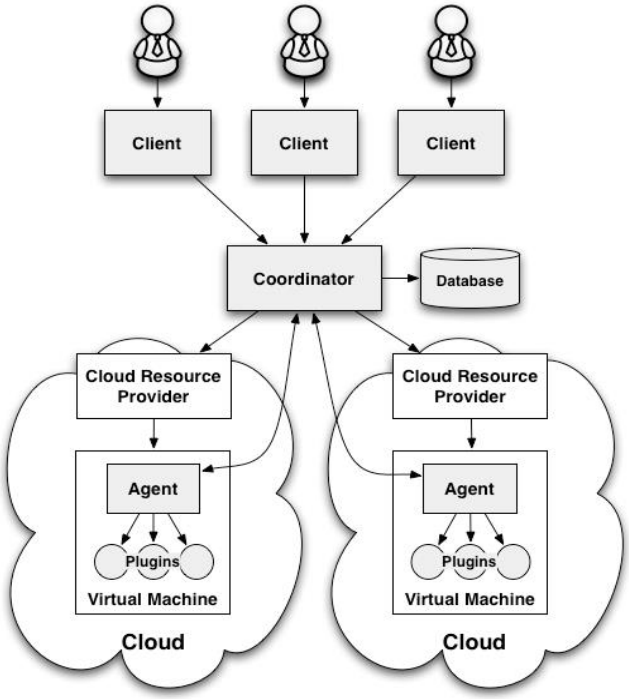
US East-1B shows the graph between request/second and HTTP connections in server-2.

US East-1C shows the graph between request/second and HTTP connections in server-3

All these servers are established in US and the difference between these server is respect to their architecture specified in architecture comparison.



(Figure 20 (d): Flow-Chart of a web-application to examine the cloud in US)



(Figure 21: Cloud Management through Coordinator over Several Clients)

Request Scheduling and Pending Latency:

App Engine routes each request to an available instance. If all instances are busy, App Engine starts a new instance.

App Engine considers an instance to be "available" for a request if it believes the instance can handle the request in a reasonable amount of time. With multithreading disabled, this definition is simple: an instance is available if it is not presently busy handling a request.

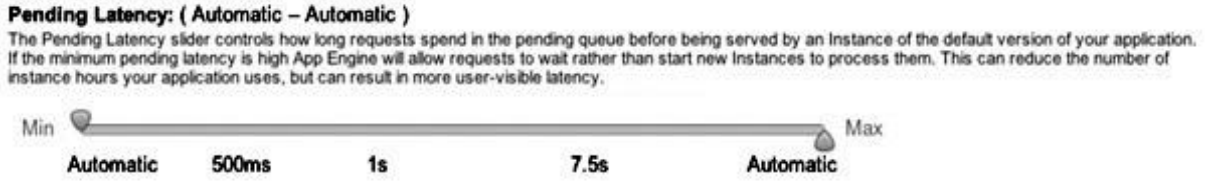
With multithreading enabled, App Engine decides whether an instance is available based on several factors. It considers the current load on the instance (CPU and memory) from its active request handlers, and its capacity. It also considers historical knowledge of the load caused by previous requests to the given URL path. If it seems likely that the new request can be handled effectively in the capacity of an existing instance, the request is scheduled to that instance.

Incoming requests are put on a *pending queue* in preparation for scheduling. App Engine will leave requests on the queue for a bit of time while it waits for existing instances to become available, before deciding it needs to create new instances. This waiting time is called the *pending latency*.

You can configure a maximum pending latency and a minimum pending latency for your app. By default, App Engine will determine appropriate latency bounds automatically.

The *maximum pending latency* is the most amount of time a request will wait on the pending queue before App Engine decides more instances are needed to handle the current level of traffic. Lowering the maximum pending latency potentially reduces the average wait time, at the expense of activating more instances. Conversely, raising the maximum favors reusing existing instances, at the expense of potentially making the user wait a bit longer for a response.

The *minimum pending latency* specifies a minimum amount of time a request must be on the pending queue before App Engine can conclude a new instance needs to be started. Raising the minimum encourages App Engine to be more conservative about creating new instances. (This minimum only refers to creating new instances. Naturally, if an existing instance is available for a pending request, the request is scheduled immediately.)



RESIDENT INSTANCES:

Instances stick around for a while after finishing their work, in case they can be reused to handle more requests. If App Engine decides it's no longer useful to keep an instance around, it shuts down the instance. An instance that is allocated but is not handling any requests is considered an *idle instance*.

Instances that App Engine creates and destroys as needed by traffic demands are known as *dynamic instances*. App Engine uses historical knowledge about your app's traffic to tune its algorithm for dynamic instance allocation to find a balance between instance availability and efficient use of resources.

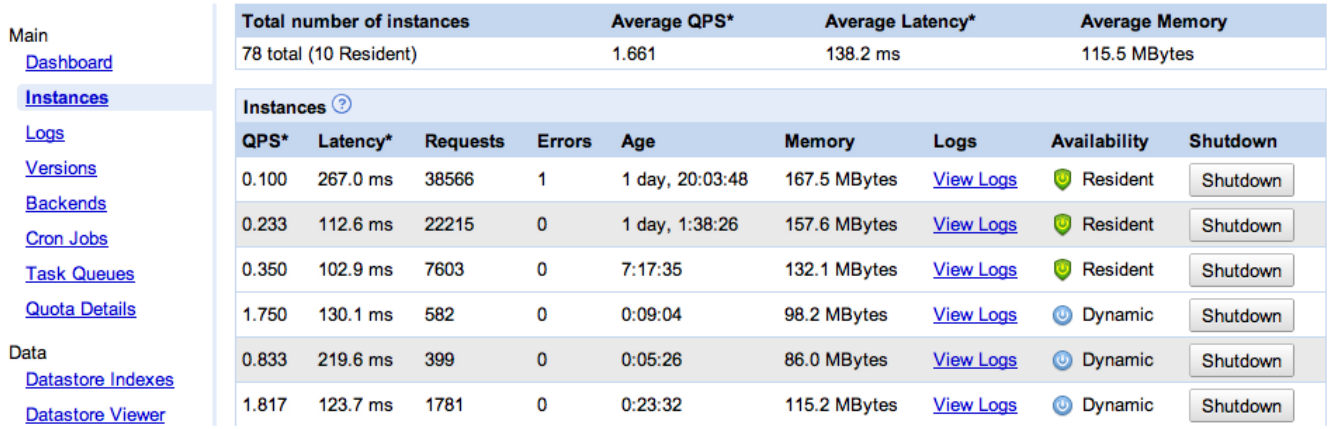
You can adjust how App Engine allocates instances by using two settings: minimum idle instances and maximum idle instances.

The *minimum idle instances* setting ensures that a number of instances are always available. The actual number of idle instances will still fluctuate with traffic, but increasing this number will cause App Engine to be more aggressive about starting new instances as it tries to ensure the minimum.

Setting a nonzero minimum for idle instances also ensures that at least this many instances are never terminated due to low traffic. Because App Engine does not start and stop these instances due to traffic fluctuations, these instances are not dynamic; instead, they are known as *resident instances*.

You *must* enable warm-up instances to set the minimum idle instances to a nonzero value.

Reserving resident instances can help your app handle sharp increases in traffic. For example, you may want to increase the resident instances prior to launching your product or announcing a new feature. You can reduce them again as traffic fluctuations return to normal. You might also reserve instances prior to executing a large batch job, or keep instances available for task queues, so spikes in load are less likely to affect end users.



CHAPTER – 4 METHODOLOGY AND SOFTWARE TOOLS USED

4.1 METHODOLOGY

As the performance issue being a major issue for cloud computing in java applications, it needs to be solved as soon as possible for a better future of cloud computing.

There are following several possible ways through which the removal of performance issue can be done:

4.1.1 Optimization in Framework:

As general java applications are developed using a particular framework and each application has its different characteristics. A Term, Application Filtering, is used to distinguish the application on the basis of the framework which is to be used in developing the application. Application Filtering performs the task to filter the applications as per their requirements and development. Application Filtering is also a beneficial step to reduce performance issue. If each application specification is known to developers while developing the application, the application performance could be better. But in most of the cases, it is not possible for that kind of cases, application filtering is useful.

To reduce performance issue, the best way is framework optimization. As spring framework is customized for clouds (Nov 2012), all the frameworks should be customized. Struts framework has not been customized yet now. It will have to customize before cloud computing covers the IT-Sector.

DeadlineExceededException is thrown in spring framework in clouds if the application takes long time to load and the control is shifted to Framework. As the application is called first and loaded into memory, the framework coverts configuration metadata (data of data) and creates the application context object which includes all the application resources and java-classes. While requesting for the application or its services, if the existing instance could not serve in 60 seconds, framework constructs a new instance (if the range is not crossed to create instances).

Far from framework-optimization or customization, developers can also reduce this issue by optimize their code with respect to the framework used and its configuration files. But it is not a big solution for such a big problem. Framework customization is a must for this major issue.

4.1.2 Reducing the Component/Module Scanning:

As it is discusses earlier, component scanning is also a cause to performance issue. To resolve performance issue, it should be managed at a particular manner. It should be avoided until unless it is compulsory to use. If it is mandatory, the usage should be lesser.

In Spring Framework, the component scanning is handled as follows:

```
<!-- Component scanning will significantly slow down application initialization time -->
<context:component-scan base-package=””/>
```

```
<bean id=”myComponentBean” class=”org.foo.MyComponent”/>
<bean id=”myOtherComponentBean” class=”org.bar.MyComponent”/>
```

To reduce relationship-autowiring:

```
@Controller
@RequestMapping("/movie")
public class MovieController {
  MovieDaoInterface movieDao;
  public void setMovieDao(MovieDaoInterface movieDao) {
    this.movieDao = movieDao;
  }
}
```

```
<bean class="com.example.controller.MovieController">
  <property name="movieDao" ref="movieDao"></property>
</bean>
<!-- movieDao is defined elsewhere in the container configuration -->
<bean id="movieDao" class="com.example.dao.MovieDao" />
```

```
<context:annotation-config />  <!-- Turn on Autowired -->
<bean class="com.example.controller.MovieController" autowire="byType">
</bean>
```

Thus the component scanning can be ignored.

These all are the possible ways through which the performance issue can be resolved.

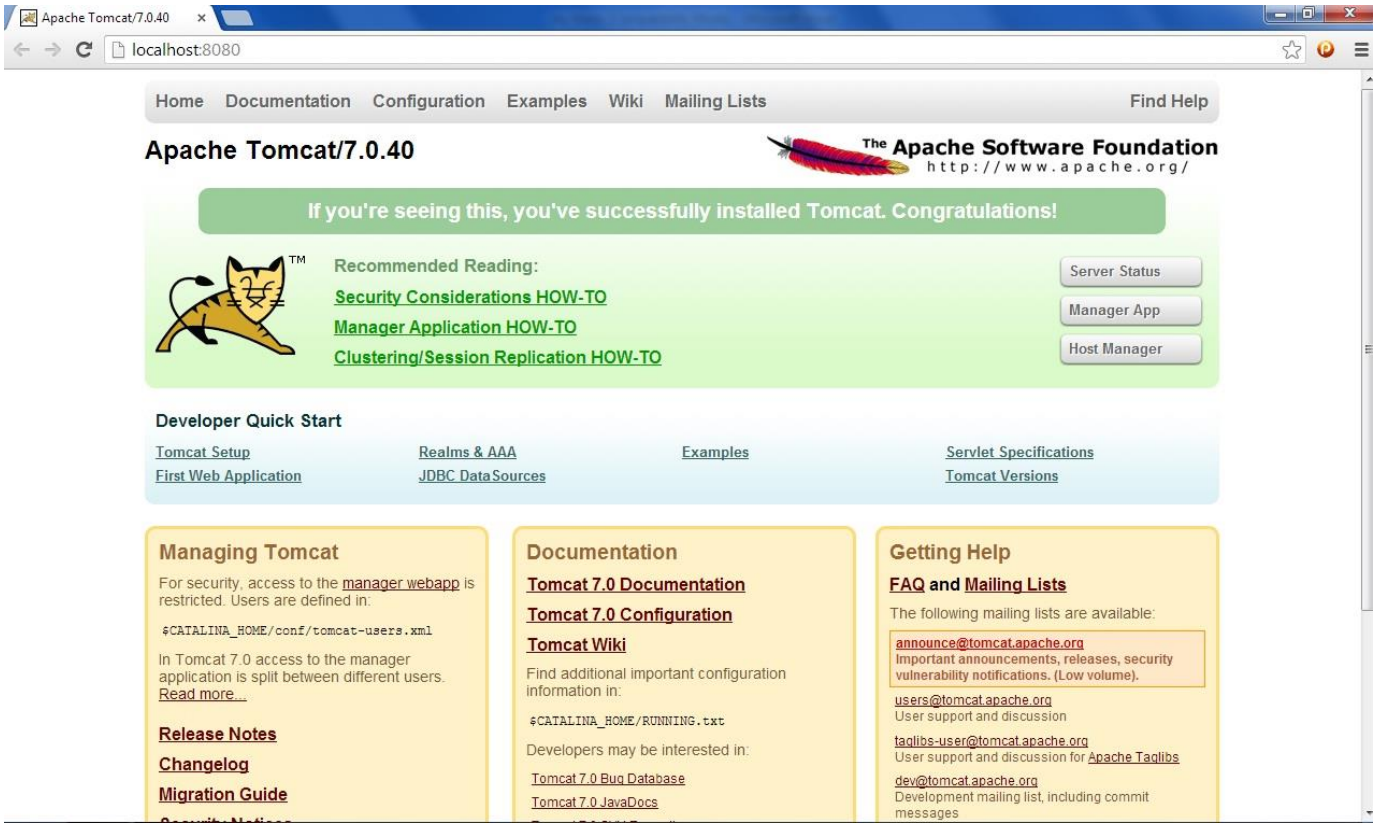
4.2 TOOLS AND SOFTWARE PACKAGES USED

4.2.1 OVERVIEW

To examine the performance issue in cloud computing and to propose the methods to resolve it, many software and tools are used which are explained as follows:

i.) **JDK (Java Development Kit) and JRE (Java Runtime Environment)**: To develop and compile java applications, JDK is used. To execute the java applications, JRE is used. To deploy the sample java application, JDK and JRE were used.

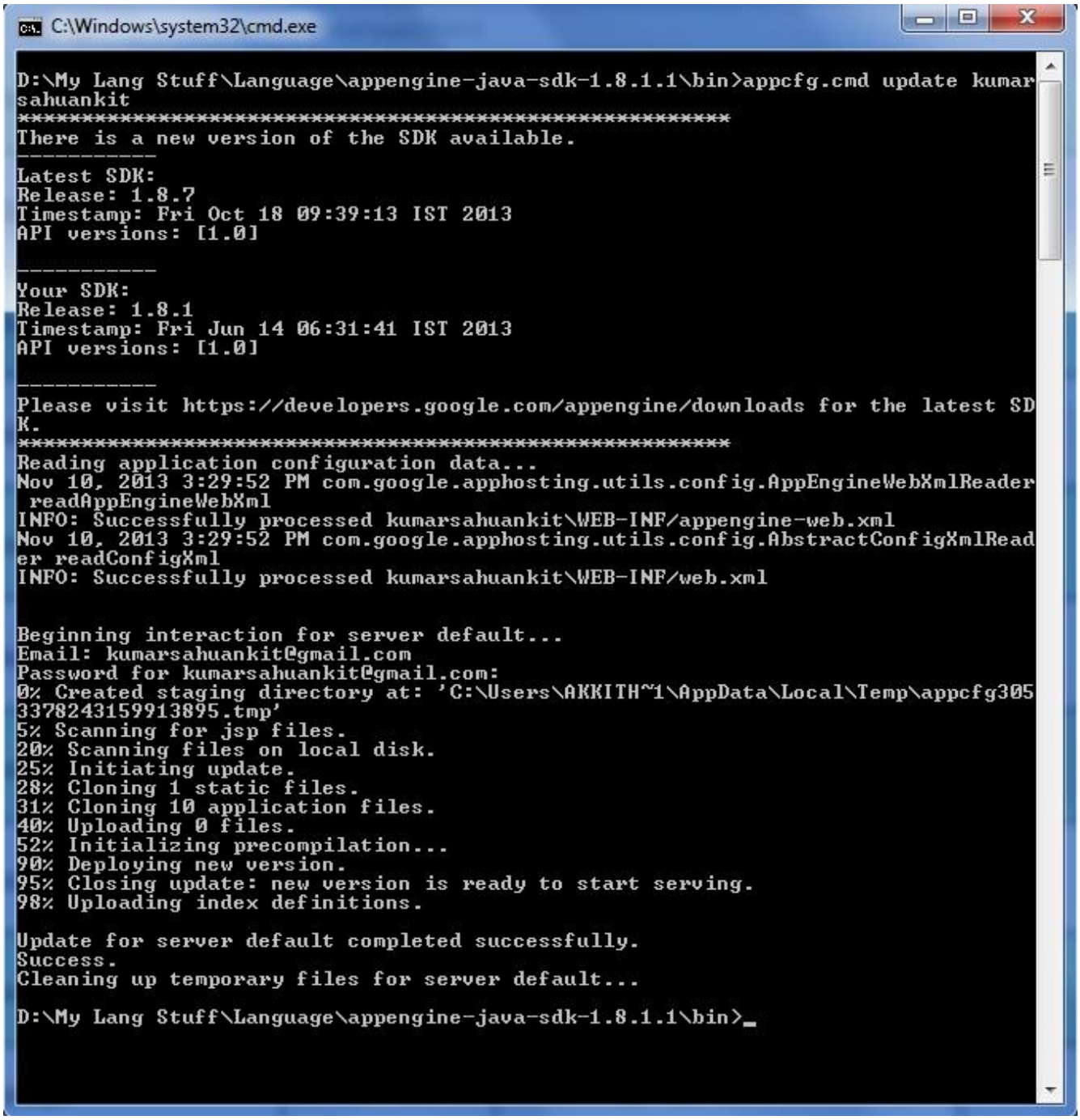
ii.) **Apache Tomcat:** Apache Tomcat is a proxy server which is used at local machine to behave as a server. Using apache tomcat, applications could be tested on local machine. To deploy the sample application on non-cloud environment, tomcat is used to compare non-cloud deployments to cloud deployments.



(Figure 22: Apache Tomcat)

iii.) **Google App Engine:** Google App Engine, discussed earlier, is a cloud platform for java and python languages. It provides free platform to deploy 10 applications. GAE is used to deploy a java application and test it on website checker and compare with non-cloud environments.

iv.) **App-Engine-Java-SDK Tool for Google App Engine:** appengine-java-sdk tool is used to deploy a java web application on Google App Engine by authenticating user through a google account.



(Figure 23: Deploying a sample application on GAE)

Pingdom, webpagetest, gtmetrix, iwebtool etc.

v.) **Website Load Checkers:** There are several online tools available that analysis a website and results the website load-statics. Some of them are:

a. **Pingdom:** Pingdom shows the load time of the application for the first request along performance grade.

b. **Webpagetest:** It performs 2 tests. Webpagetest shows the load time of the application and first view analysis and repeat view. It also shows Load Time, First Byte, Start Render, DOM elements.

c. **Gtmetrix:** It provides multiple analysis factors like page speed grade, yslow grade, page load time etc.

d. **Iwebtool:** It shows the page load time of the application and average speed per KB.

CHAPTER - 5 RESULTS AND CONCLUSION

5.1 RESULTS

5.1.1 CLOUD COMPUTING IN COMING TIME

Cloud Computing is a technology of tomorrow. The day is very near when all the remote hosting would be replaced with cloud-hosting. As per the discussed topic, it has some milestones but as per the time, these all milestones are going to be discarded from the path of the cloud computing to all the IT-Fields. Cloud Computing is stepping forwards itself day by day.

As per the research area, the major issues are solvable in coming time. Thus the cloud computing can be good technology for the coming time if all the issues are resolved. Cloud Computing heads IT to a large virtualized field. If cloud computing clears all the hurdles and replaces the existing systems into clouds, 80% of the IT-world could be the virtualized.

5.1.2 STUDY OF THE SEVERAL CLOUD PLATFORMS

To examine the performance issue in cloud computing, several cloud platforms need to be studied. Different cloud platforms like Amazon EC2, Windows Azure, Google App Engine, Salesforce.com, rackspace.com, heroku.com, zelasitic.com etc. All the cloud platforms that provide java deployment are facing the same challenge. Performance and reliability issues are major issue for applications and Cost and Security issue are major issue for the organization.

These issues make the cloud computing lesser usable if these issues do not get resolved and in this case, the cloud computing may also lose its place in the future in application deployment.

5.2 CONCLUSION

As the IT-world is stepping forward to virtualization, there are some fields that have already virtualized and some are being. As an important step to virtualization, cloud computing plays a vital role. It is considered that if the cloud computing replaces a big part of existing systems, the IT-Sector would be at peak level of virtualization.

There are several milestones in cloud computing to replace all the existing system which are to be overcome. In the web-applications, the milestone is performance, reliability, cost and security issues. The Performance issue should be handled as soon as possible due to the web-application reliability. All the cloud platforms providing java deployments should be using their best optimization to facilitate user at its best. Web applications are expected to load at fast speed.

For a better stability in IT-Sector, cloud computing needs to overcome all these issues and provide a good environment for users as well as developers. The performance issue can be resolved by proposed methods. The research analyzes the issues in cloud deployment of a java application and proposes the methods to resolve this.

Industry	**Frequency**	**%**
IT	75	23.7%
Manufacturing	63	19.9%
Finance	52	16.4%
Logistics	46	14.5%
Service	44	13.9%
Education	27	8.5%
Location		
Asia	92	29.0%
North America	81	25.6%
Europe	79	24.9%
Australia	34	10.7%
Africa	17	5.4%
Other	14	4.4%
Cloud Services Used		
Saa S	134	42.3%
Iaa S	117	36.9%
Paa S	66	20.8%
Cloud Delivery Methods		
Public	134	43.8%
Private	91	28.7%
Hybrid	72	22.7%
Other	15	4.7%

Hypothesis	Effects	Path coefficient	*t*-value	Results
H1	Trust → Successful cloud deployment	0.251**	10.438	Supported
H2	Technical capability → Successful cloud deployment	0.335**	7.950	Supported
H3	Management capability → Successful cloud deployment	0.269**	8.273	Supported

(Figure 24: Statics of cloud computing in 2012)

REFERENCES

[1] The Basics of Cloud Computing by Alexa Huth and James Cebula.

[2] Introduction to the cloud computing by Dialogic.com

[3] appengine.google.com, developer.google.com

[4] Borko Furht Armando Escalante," Hand book of Cloud Computing "by Springer.

[5] G. Juve, E. Deelman, K. Vahi, and G. Mehta, "Scientific Workflow Applications on Amazon EC2," Workshop on Cloud-based Services and Applications in conjunction with 5th IEEE International Conference on e-Science (e-Science 2009), 2009.

[6] Using Google App Engine-O'Reilly Google Press by Charles Severance.

[7] X-as-a-Service: Cloud Computing with Google App Engine, Amazon Web Services, Microsoft Azure and Force.com by Rabi Prasad Padhy, Manas Ranjan Patra and Suresh-Chandra-Satapathy.

[8] Identifying Key Challenges in Performance Issues in Cloud Computing by Ashraf Zia.

[9] Cloud Computing Security Issues by Randy Marchany.

[10] The Cost of a Cloud: Research Problems in Data Center Networks by Albert Greenberg, James Hamilton, David A. Maltz, Parveen Patel.

[11] Cloud Computing: Security and Reliability Issues by Farhad Ahamed, Seyed Shahrestani and Athula Ginige.

[12] A Novel Approach for Handling Security in Cloud Computing Services by Sahar Mohammad Abduljalil, Osman hegazy and Ehab E Hassanein.

[13] SAP Cloud Computing by Joseph Yeruva, MPHASIS.

[14] Cloud Computing in the Public Sector by Russell Craig, Jeff Frazier, Norm Jacknis, Seanan Murphy, Carolyn Purcell, Patrick Spencer, JD Stanley.

[15] Moving from Legacy Systems to Cloud Computing: A Tata Communication White Paper.

[16] GTSI Group, "Cloud Computing-Building a Framework for Successful Transition," White Paper, GTSI Corporation, 2009.

[17] Rajnish Choubey, Rajshree Dubey and Joy Bhattacharjee, "A Survey on Cloud Computing Security, Challenges and Threats", International Journal on Computer Science and Engineering (IJCSE), vol. 3, No. 3, 2011.

[18] Andrew Joint and Edwin Baker, "Knowing the past to understand the present- issues in the contracting for cloud based services", Computer Law and Security Review 27, pp 407-415, 2011.

[19] Michael Miller, "Cloud Computing Pros and Cons for End Users", microsoftpartnercommunity.co.uk, 2009.

[20] Radu Prodan and Simon Ostermann, "A Survey and Taxonomy of Infrastructure as a Service and Web Hosting Cloud Providers", 10th IEEE/ACM International Conference on Grid Computing, 2009.

[21] Google App Engine documentation link. http://code.google.com/appengine/docs/whatisgoogleappengine.html.

[22] Liladhar R. Rewatkar, Ujwal A. Lanjewar, Implementation of Cloud Computing on Web Application, International Journal of Computer Applications (0975 – 8887) Volume 2, No.8, June 2010.

[23] K. Mukherjee, G.Sahoo, Development of Mathematical Model for Market-Oriented Cloud Computing, International Journal of Computer Applications (0975 –8887) Volume 9– No.11, November 2010.

[24] Google App Engine And Performance Of The Web Application By Anjali Jain.

[25] Into the Cloud: An Evaluation of the Google App Engine by Chornyi, Dmitry, Riediger, Julain, Wolfenstetter, Thomas.

[26] Mei, L., Chan, W. K., and Tse, T. H. 2008 A tale of clouds: paradigm comparisons and some thoughts on research issues. In Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference, pp. 464-469.

[27] Raman, T. V. 2008. Cloud computing and equal access for all. In Proceedings of the International Cross-Displinary Workshop on Web Accessibility, 2008, pp. 1-4.