CMPE 281 - Cloud Technology

Instructor: Jerry Gao, Ph.D.

Semester: Spring, 2015

CMPE 281:

Mobile Infrastructure as a Service for MTaaS

TEAM - 9

KAVYA CHANDRADHARA

MANASVINI SURYANARAYANA

NIKITHA RAVIRAJ

NAVYA GANDHI

Table of Contents

Mobile Infrastructure as A Service for Mobile Testing As a Service

1. INTRODUCTION

Cloud computing is a technology which uses a network of remote servers to cache, manage and process the data, instead of using a local machine. Using cloud computing we can deploy applications on the machines and thus access them via Internet. Cloud provider provides the users the machines to deploy applications and are managed by them. The most robust element of Cloud Computing is giving the users infinite resources and thereby increasing the scalability of the application.

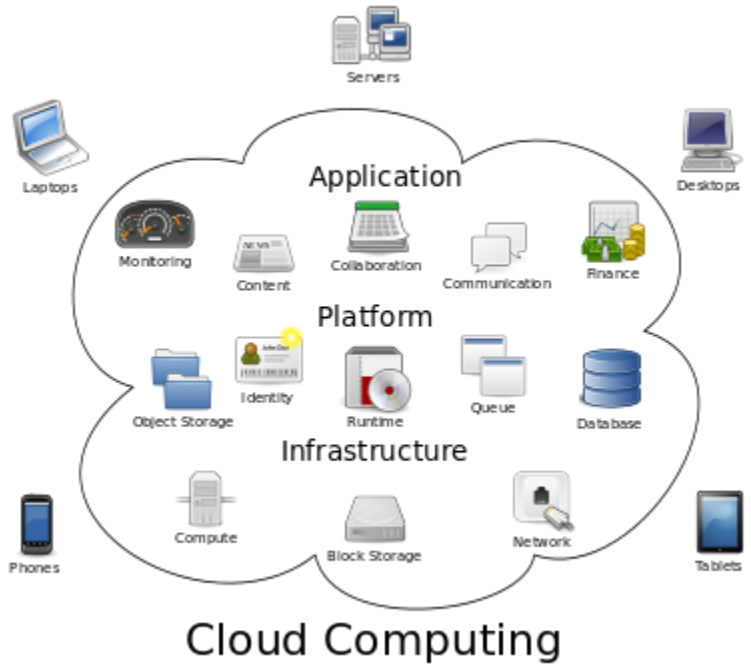
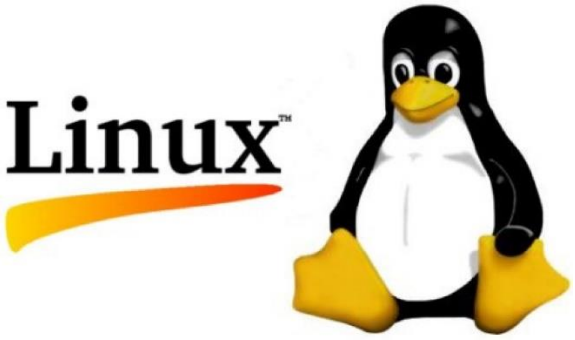


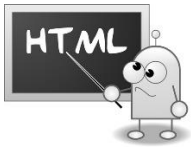
Figure 1. Cloud Architecture

In this project we are designing a mobile Infrastructure as a service mode for Mobile Testing as a service users. Infrastructure As a service is model where the needs an external support from the service provider and hence outsources the requirements like storage, network components, hardware etc. to the service providers, the providers will here be responsible casing and maintaining it. Infrastructure as a service is a provisional model which uses the Pay as you use model. Infrastructure as service model offers many features like On Demand Self Service, resource pooling, rapid elasticity, measured service etc.

With a brisk increase in the mobile industry, comes a necessity to develop and the validate the applications deployed on the mobile web. Users can submit on demand requests to the mobile testing as service resources. The resources can range from mobile devices, mobile Operating systems, servers, hubs etc. In this project we have developed a Mobile infrastructure as a service which gives us support like monitoring, provisioning and billing of the resources.

2. Technologies Used





Amazon Web Services

Amazon cloud is the technology we have used to build the infrastructure for our project. Amazon EC2 cloud compute gives its users the privilege to run their own applications on their virtual machines on the basis of a fixed pricing. A virtual machine can be created by the user through a web service which allows the users to boot an Amazon Machine Image. There by Amazon Elastic Cloud Compute web service allows scalable deployment of applications. A user can create as many server instances he wants to and the instances are charged on an hourly basis. The elasticity in the service here is the control over the geographical locations of instances.

Other Technologies

Front End: HTML, CSS, JSP, JAVA SCRIPT, PHP
Back End: Java and Python
Environment: Eclipse
Cloud Watch: Python
Database: MySQL

3. Functional Component Overview

There are a total of 5 components in our project:

- **Mobile Service Request Generator**: The user requests for resources required. This is used by the request loader component to generate multiple requests for different cloud resources. The resources requested can be computing servers, hubs, emulators, mobile devices etc. The generated requested will be pooled in the load balancer.
- **Load Balancer**: This component queues the requests generated and distributes the load of the requests to the resource provisioner.
- **Resource provisioner and manager:** Once this component receives the request from the load balancer, automatic provisioning of the resources happens. When the resources are released by the users, the deprovisioning of the resources happens. An optimized algorithm will be used in this process.
- **Monitoring and Dashboard**: The allocated resources like emulator, hubs, mobile device will be here monitored. They will be here monitored based on time frame they were used for and on the flavor of resource. Cloud watch API is used for this purpose.
- **Billing Services:** Taking the account of the monitored data, cost metrics for each resources are estimated.

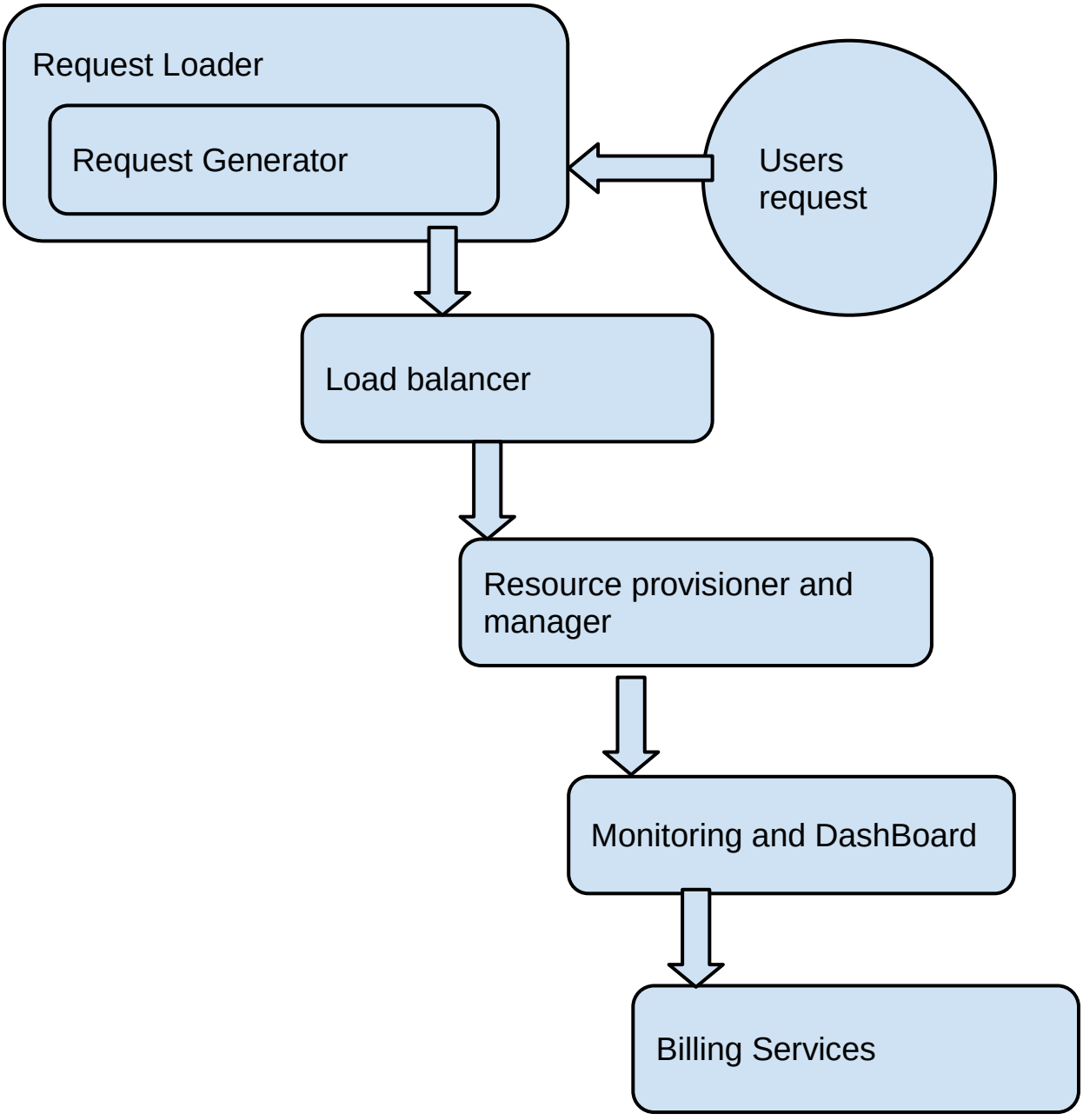


Figure. 2. Functional Components Overview

4. Request Generator and Request Management

This component accepts request from users, stores it into database and updates the load balancer queue with the detalis requested by the user.The user congigures the request parameters and the resource we will provide for the users are Mobile device (IOS or Android),Emulator,CPU,storage,Mobile device version,Location,duration and HUB of the service.All these parameters are specified by the user according to his requirement.

The services provided for Users,

1) Mobile Devices – (Samsuang, iphone).
2) Emulator for Android.
3) Hub Services.
4) Storage.
5) Server OS(windows,ubuntu,Redhat).
6) CPU size(t2.micro,t2.medium).
7) Duration.

AWS instance utilized for our project:

T2 instances have expandable performances and provide basic CPU performance at first creation.Cpu Credits monitor the baseline performance and burst time for an instance.Each instance receive CPU credits at a certain rate based on the size of instance.Cpu Credits are accurue when ideal and used when instance is active.With very less burst frequency t2 instance are good where CPU is not used fully.

Features:

- Intel Xeon high frequency Processors operating at 2.5GHz and turbo of 3.3GHz .
- CPU credit governing, Controlled CPU bursts and consistence performance.
- Network resource, memory and computation balance.

Mobile cloud:

We have considered each instance as a mobile cloud to make it simple for explanation. Each instance is capable of hosting and supporting 6 emulators/simulators and represent 6 device launched on cloud. So it 6 instances for 6 device. So for 30 android emulators there are 30 logical instances.

Request generation details and strategy:

Input : User request for service provided.

Output:Request flows to the load balancer module.

The requests are embedded in the queue on the FIFO based service.The latest requests are feed to the end of the queue.

WEB User Interface Dashboard.

The web user dashboard is designed for the users is built using javaservelets,JSP,HTML,CSS.

- HTML and CSS for front end.
- Javasrevelets and JSP for middleware.
- Mysql DB for the backend.

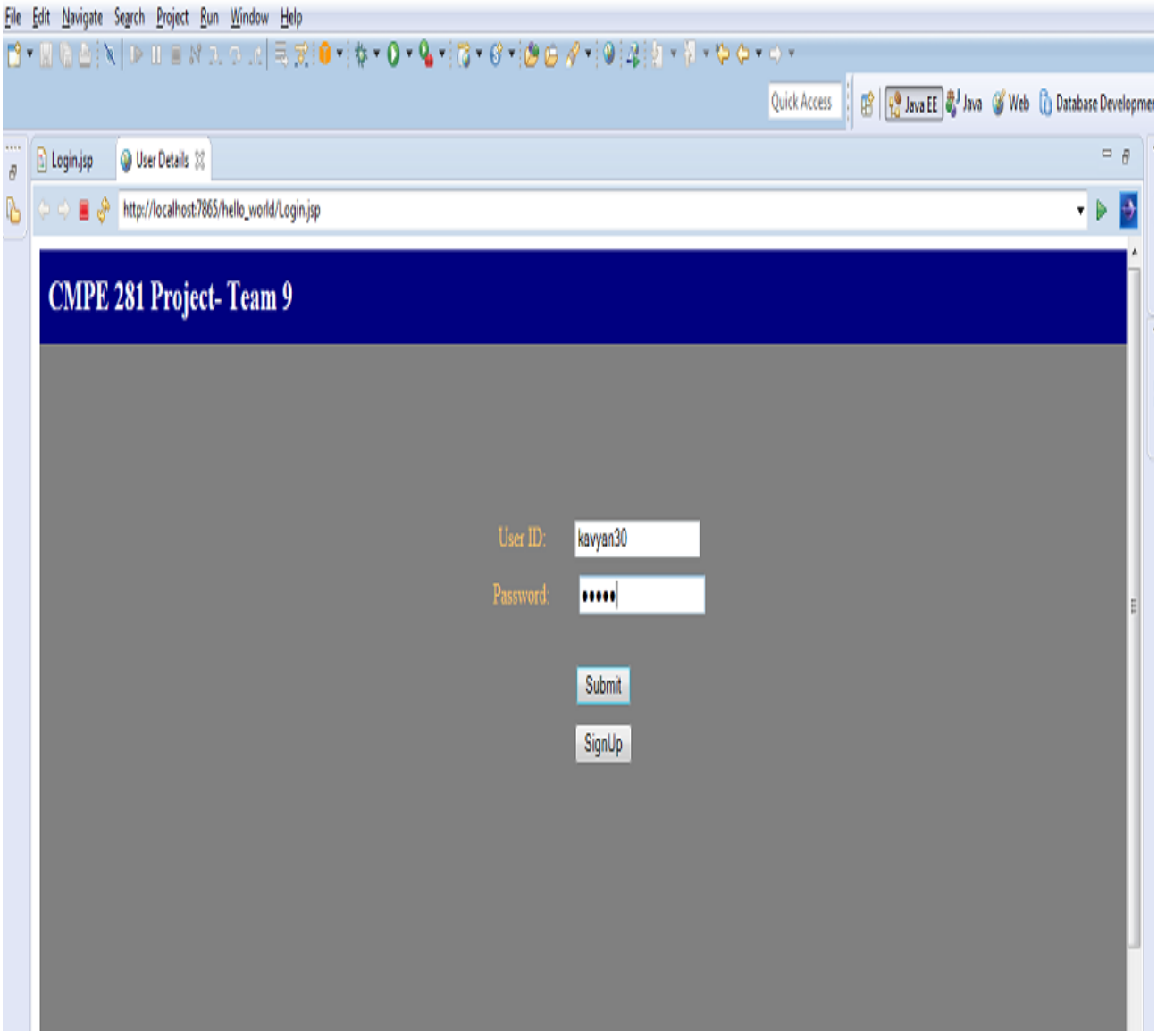
Dashboard has login screen first.The user can login using the sign in button.It verifies with the data base ,if the validation with database is successful ,user can navigate to the next screen.It also gives the user the sign up option.

Next screen contains the services provided and the user can enter the details about the service request.Once user enters the data and clicks submit project button the database is updtaed with the latest service request.The data from the database is assigen to the queue of the load balancer.Later the load balancer allocates the request to the particular reagion based on the number of requests served by the location.If the region is filled with 100 requests irrespective of the request type, then we are not serving the request for sometime till some of the requests are completed and resources released.

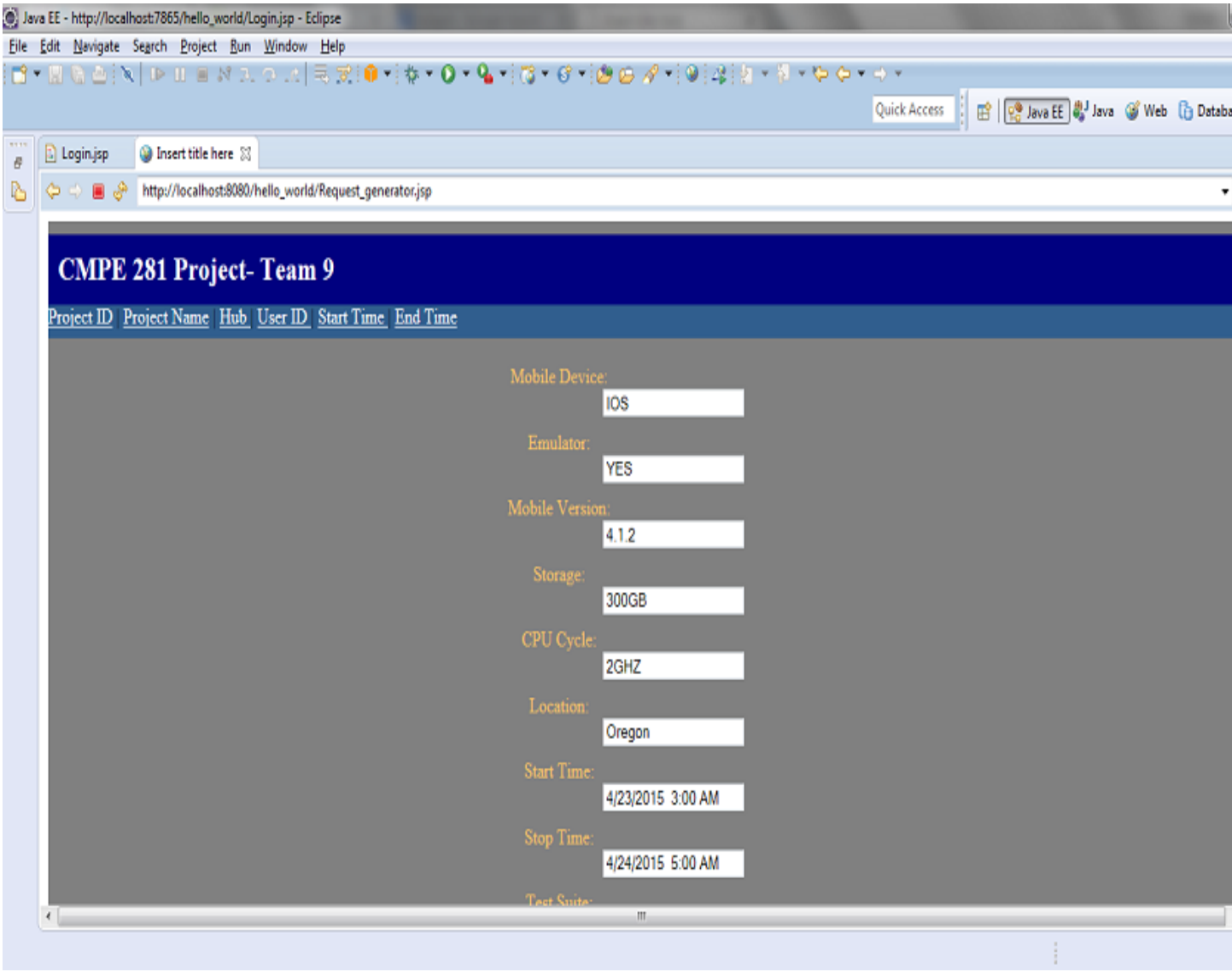
Dashboard Details.

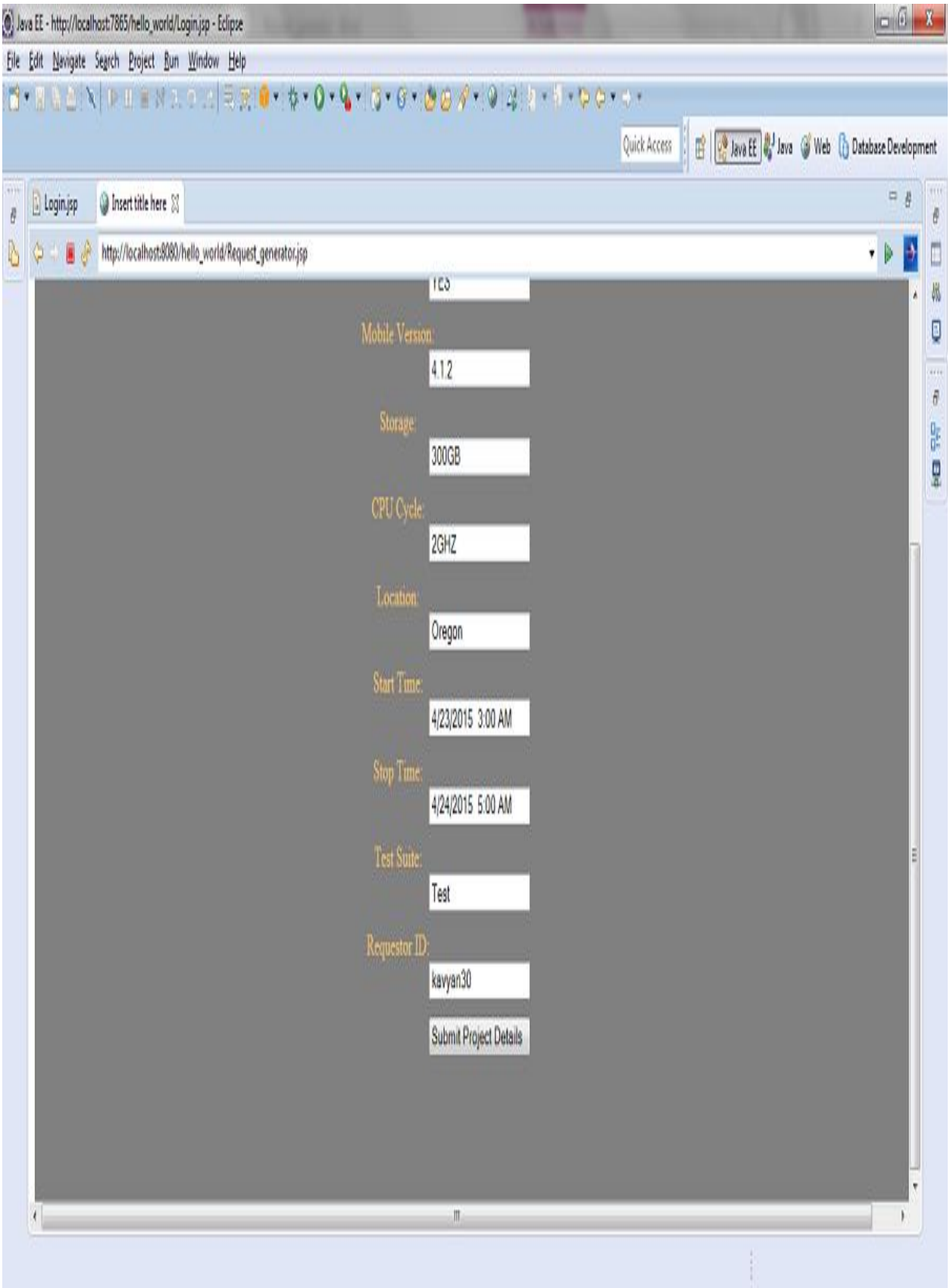
1) Login Screen.

Accepts the input and validates with the database if user has not yet registered it requests for sign up.



2) User request




Java EE - http://localhost:7865/hello_world/Login.jsp - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Java EE
Java
Web
Database Development
Login.jsp
Insert title here
http://localhost:8080/hello_world/Request_generator.jsp
123
Mobile Version:
4.1.2
Storage:
300GB
CPU Cycle:
2GHZ
Location:
Oregon
Start Time:
4/23/2015 3:00 AM
Stop Time:
4/24/2015 5:00 AM
Test Suite:
Test
Requestor ID:
kavyan30
Submit Project Details

6. Load Balancer

Definition

Load balancing is as one of the component in cloud projects as it is used for distributing a large processing resource to smaller processing nodes for boosting up the overall performance of the application. An excellent load balancing algorithm will help the system to avoid overloading or under loading of any particular computing node.

It is not easy to come up with a load balancing algorithm in cloud computing project since it involves several other criteria such as security, reliability, throughput etc which should be taken care of while preparing a good algorithm. So, the main aim of load balancing algorithm is to reduce the response time of completing the task by distributing the total load of the application to different computing nodes.

Working types of Load Balancing

Load balancing can be made to work in 2 ways:

1) Cooperative
2) Non-Cooperative

Cooperative Mode: In this mode, all nodes work simultaneously to achieve the goal of reducing overall response time.

Non-Cooperative: Each tasks run independently in order to reduce response time of the tasks.

Types of Load Balancing Algorithm

Load balancing algorithm can be divided into 2 types:

1) Static load balancing algorithm

2) Dynamic load balancing algorithm

Static load balancing algorithm: These algorithms do not take into consideration things like previous state or behaviour of a node while distributing the tasks.

Examples of static load balancing algorithm:

1) Round-Robin Load Balancer

2) Min-Min

3) Max-Min

4) Load Balance Min-Min

5) Load Balance Max-Min-Max

Dynamic load balancing algorithm: These algorithms check the previous state of the node before distributing the tasks. This algorithm is applied either on distributed or non-distributed systems. In dynamic load balancing system, nodes interact with each other by passing messages, thus generating huge number of messages when compared to static.

Advantage of using dynamic load balancing algorithm over static load balancing algorithm is that if any node fails, it will not stop the system from working but only harm the performance of the system.

Examples of dynamic load balancing algorithm:

1) Equally Spread Current Execution

2) Throttled Load Balancer

3) Honeybee Foraging Algorithm

4) Biased Random Sampling

5) Active Clustering

6) Join-Idle Queue

Polices are used by dynamic load balancer to keep record of all the information. There are 4 policies that are used by the dynamic load balancer and they are as follows:

1) Transfer policy: When an elected task needs transfer from a local to a remote node this policy will be used.

2) Selection policy: This policy is used when processors need to communicate messages between them.

3) Location Policy: This policy is used for choosing a destination node for the transfer.

4) Information Policy: This policy is used to keep all information and status of nodes in the system.

There are several ways for comparing the load balancing algorithms. Few of them are as followed:

- Throughput: This can be calculated by estimating the total number of tasks that has been completed within a fixed duration of time, without taking, creation and destruction of virtual machines into consideration.
- Execution time: This can be calculated by estimating the time taken for completing the task for creation of virtual machines, processing time of a task and destruction time of virtual machine.

Basics steps for any cloud computing algorithm are as follows:

- End-user requests the cloud coordinator.
- Cloud coordinator distributes the work into cloudlet and sends it to data centers.
- Data center coordinators work on scheduling the work. This also selects the algorithm, which will be used for scheduling work.

Load Balancing Algorithms Used In Project

In our project we have compared two load balancing algorithm. They are as followed:

1) Random Priority based Algorithm

2) Artificial Honey Bee algorithm.

Random Priority Based Algorithm:

Random priority based algorithm is a type of static load balancing algorithm. This algorithm assigns random priority for the incoming resources and pushes them to the region which falls in its range.

Figure 1 shows the flowchart that describes the working of random priority based load balancing. This algorithm does not take into account the previous load state of the node at the time of allocating tasks.

We have taken 3 regions in our project and created 10 VMs in each region. Each region is assigned with a range such that the random number generated falls in one of those ranges. When we get end user request, each of the request is assigned with a random number and then the resources are sent to the regions which is in the range of the random number associated with that resource.

Advantage of this algorithm is that it is very easy to create and understand. It works great if the random numbers generated do not repeat frequently.

Disadvantage of using this as load balancing algorithm is that there is no control of the random numbers generated. This may affect the performance of the application since there are chances of a particular node to be heavily loaded while compared to others.

If this algorithm is altered to come up with a nice formula to create random numbers such that the load is equally distributed to different regions, then it may improve the performance of the application.

We approached this problem by implementing Artificial Honey Bee load balancing algorithm in our project.

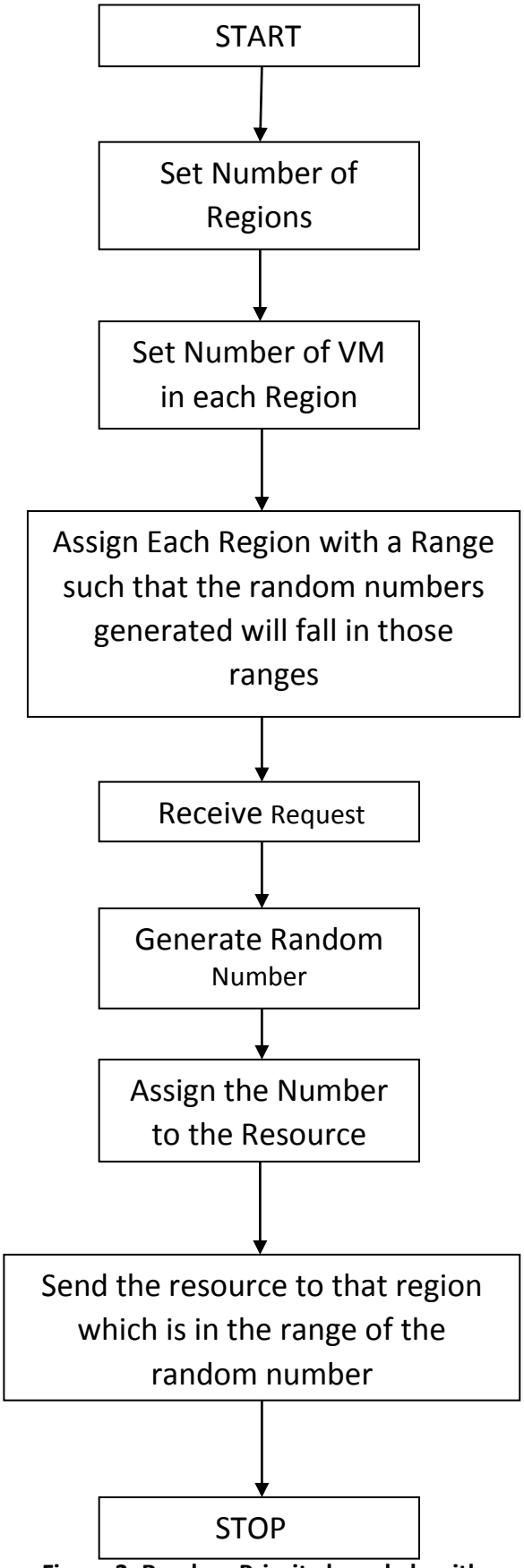


Figure 3: Random Priority based algorithm

The Bee Algorithm

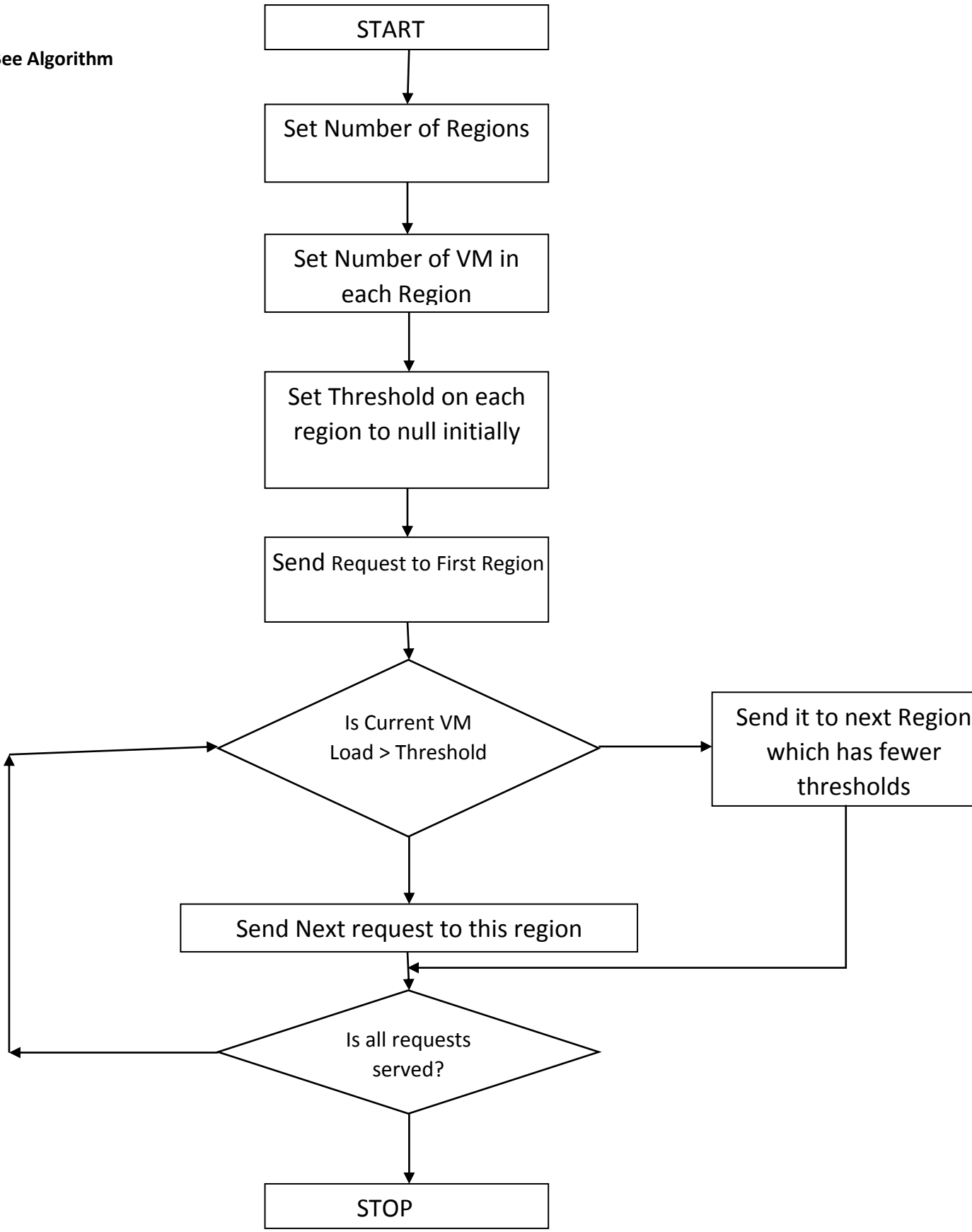


figure 4: Artificial Honey Bee algorithm

The Bee Algorithm

This algorithm was made by noticing the behavior of the bee colony. It is a very good optimization algorithm. It belongs to family of dynamic load balancing algorithms. This algorithm also balances the tasks based on priorities on the system, such that the total waiting time of the tasks in the queue is minimized. It maintains overloaded and under-loaded CPUs. It also provides very good QOS.

Figure 2 is the flowchart that describes the working of bee algorithm that we have implemented in our project. We have taken 3 regions in our project and created around 10 VMs in each region. The threshold value is about 5 per region. When we receive a request from our end-user, we go to the first region and check if it meets the threshold level, if it does not then we will send all requests to be served by that region till the threshold is reached, else we will go to the next region and check its threshold. This cycle repeats till all the requests are served.

Comparison Matrices for Random Priority based Algorithm and Honey Bee Algorithm

Servers No.	Response time(in ms)	
	Bee Algorithm	Priority Algorithm
0	234	187
1	218	171
2	202	156
3	187	140
4	171	124
5	156	109
6	140	93
7	124	78
8	109	62
9	93	46

Figure 5: Response time of Honey Bee algorithm and priority algorithm for 10 servers

No. of Servers	Average response time	
	Bee Algorithm	Priority Algorithm
10	163.4	116.6
20	211.0	231.4
30	266.9	295.9
40	349.43	272.98
50	291.32	258.24

Figure 6: Average response time of Honey Bee algorithm and Priority Algorithm

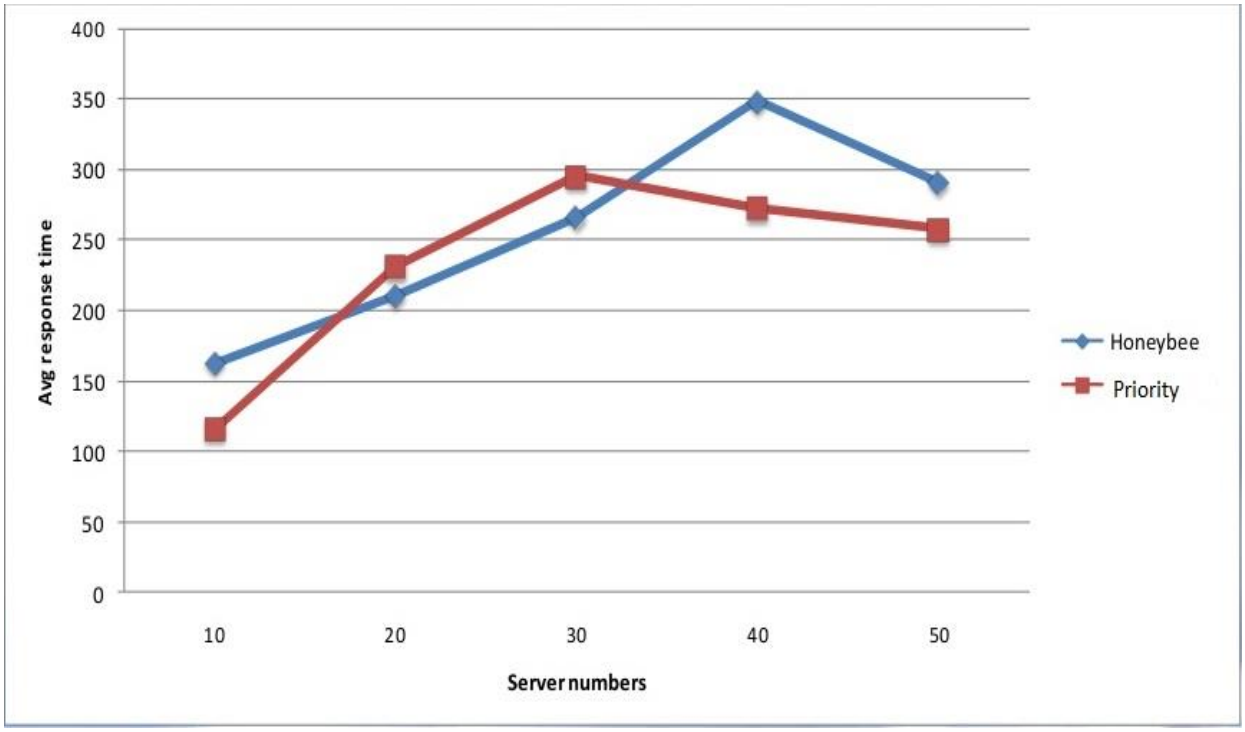


Figure 7: Response time Vs Servers number graph

Servers No.	Bill pay	
	Bee Algorithm	Priority Algorithm
0	1	4
1	1	5
2	1	6
3	4	6
4	5	6
5	6	6
6	6	8
7	6	8
8	6	8
9	8	9

Figure 8: Bill pay of Honey bee Algorithm and Priority Algorithm

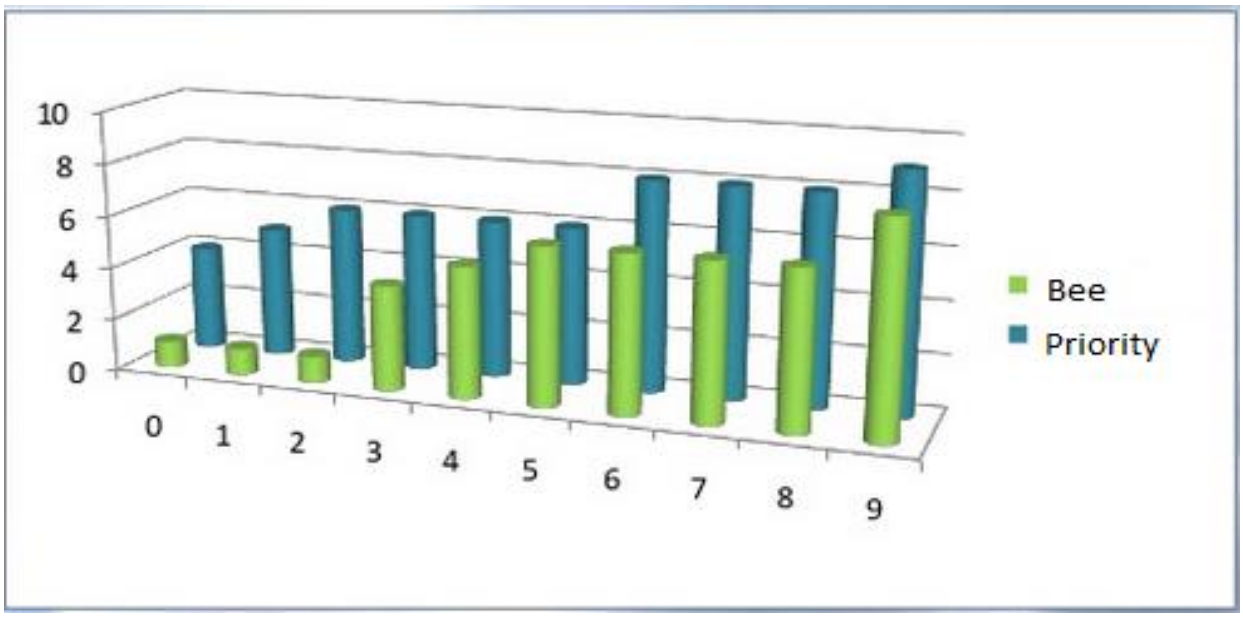


Figure 9: Graph of Bill Vs Servers number

7. Resource Allocation and Resource management

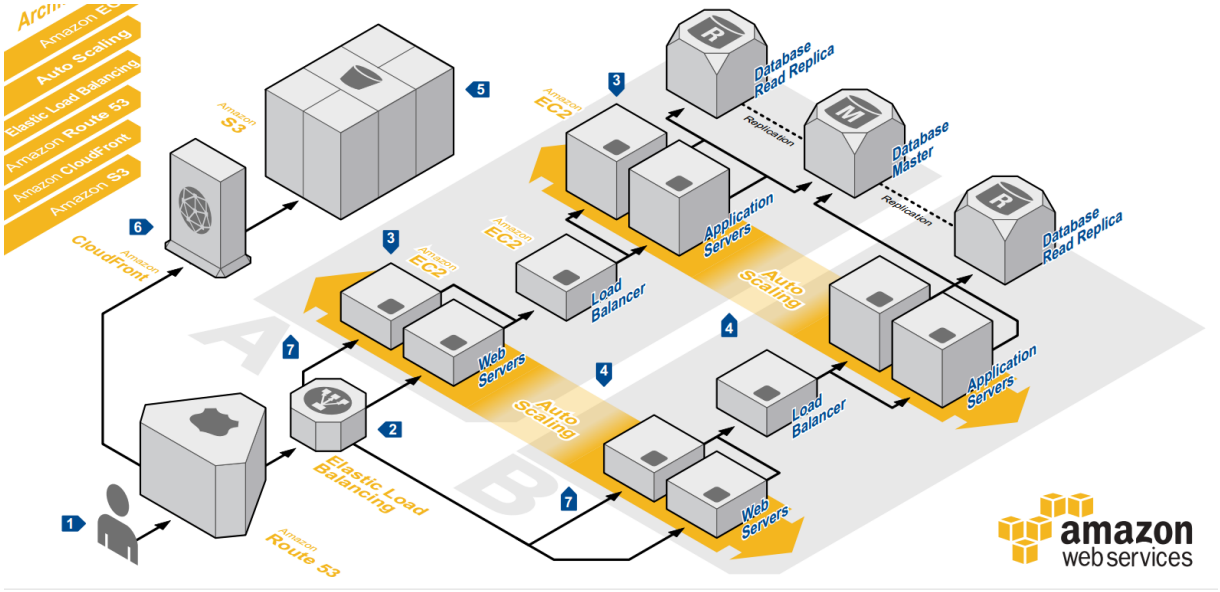


Figure 10. Resource Allocation

Given the types of resources that should be served to the user, resource allocation component will serve those many different types of resources to the user. Hence main goal of resource allocation is to provide resources to the user.

In this project, we classifies three types of resources to be provided to the user, namely, Servers, Emulator and Hubs. Hence once the user requests any or all such resources, request on passing through load balancer reaches resource allocator. Then the resource allocator will create a new resource or use existing resource based on the request and sends back the resource information for the user.

Input: Request parameter, region

Output: Resources up and running for the user

The main component of Resource Allocation are,

- Identifying the resource type
- Checking whether the resource is available
- Create/Use resource requested
- Storing the request information for the dashboard

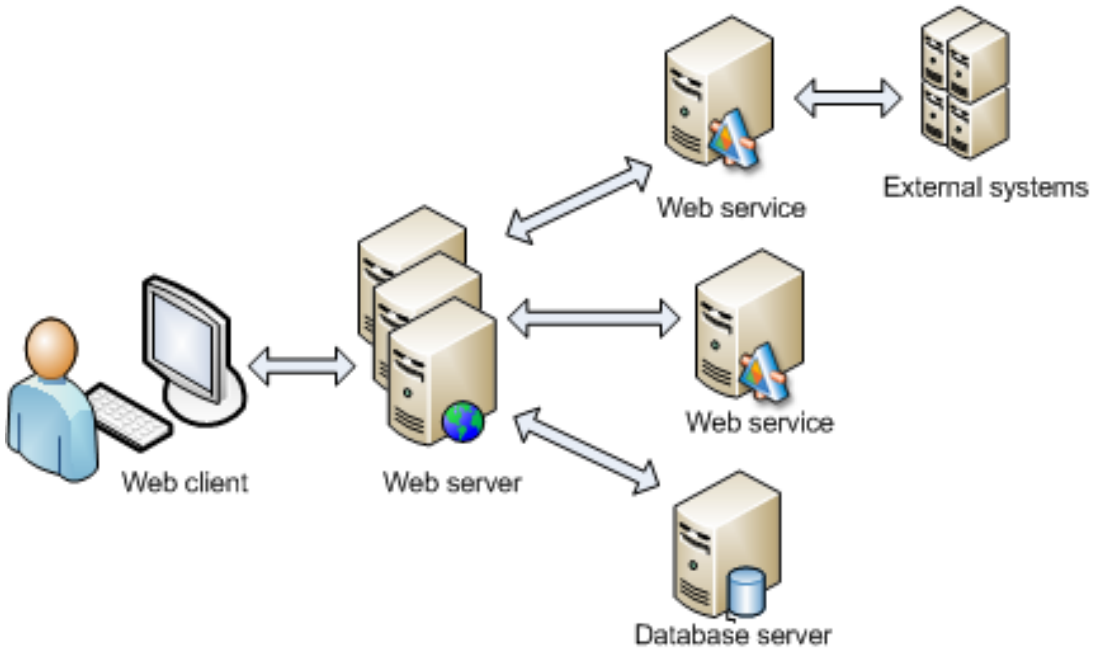
Each request was being served by the components which was listed above. Along with these document, an algorithm is used before we allocate any resource. All these components and algorithm will be explained in detail in the following sections.

Identifying the resource type:

Once the request comes to resource allocator, the first step is to find out the resource type requested for. Before identifying the resource type, below given is the brief information of types of resources we are provided in our project. The main goal of this project was to provide cloud infrastructure resources using which user can test his mobile applications. Hence any user can be able to use such resource to run/configure for his/her mobile applications and then test the application. Because of this definition, we classified the resource types as Server, Emulator and Hub.

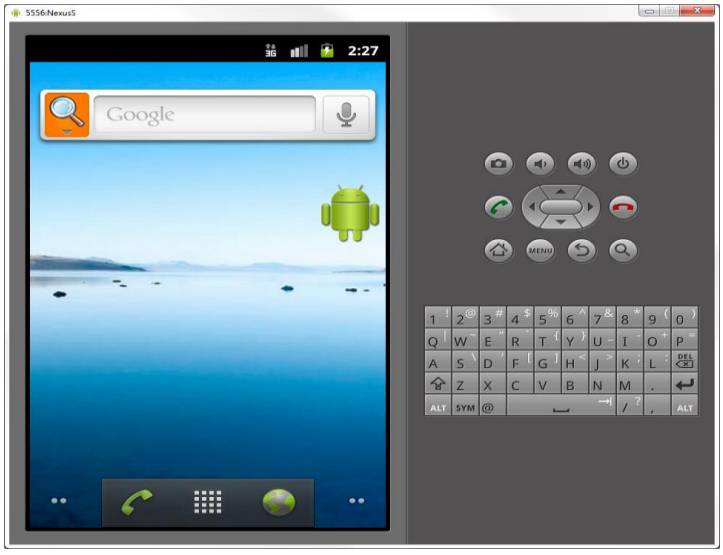
Servers:

Servers are nothing but machines running in the cloud, so that user can login to those machines and use it as server for any client-server types of applications. We are targeting on different OS based servers here.



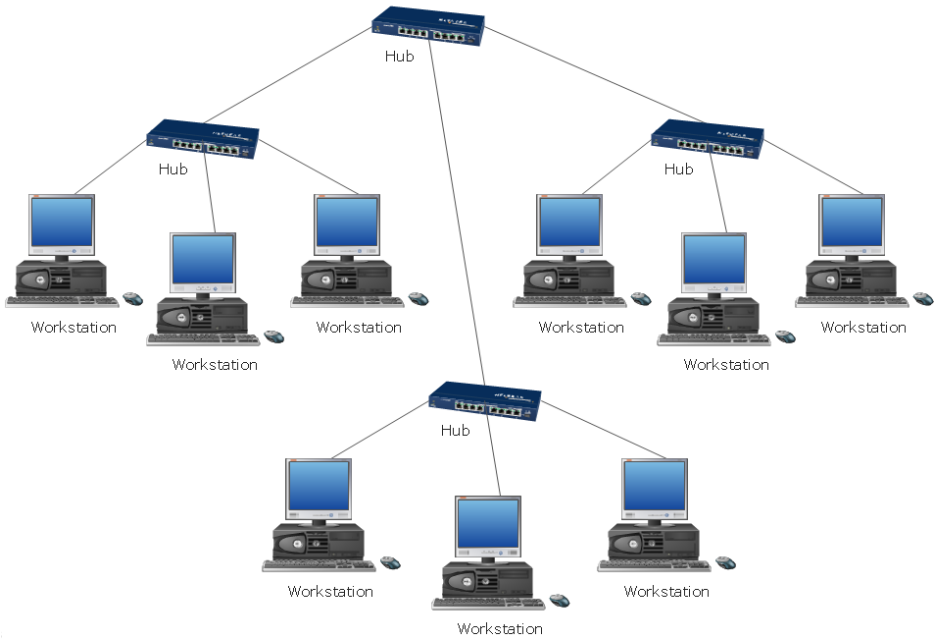
Emulator:

Emulator are nothing but a virtual android devices. These are virtual android/ios devices which will be installed on any machine. The user can deploy the mobile application to these emulator and test the behavior. In this project we are deploying an android emulator on our local machine.



Hub:

The third resource type is hub. Hub is used for connecting multiple IP devices together and then can establish secure connection between them. In this project, mobile devices, connect to server hub and then the user can use this hub and connect to other mobile devices present in the same mesh. We have deployed Hub service on top of Amazon cloud ubuntu instance. By this we are able to provide Hub service on cloud. This will achieve multi tenancy in the cloud. We have developed an apache tomcat webservice to connect multiple IP devices to the Hub. User will be given a portal to acknowledge the connection between cloud based Hub and the local device.



Hence with these types of resource available user can request for any such resource or all resource. This component helps in identifying the requested resource type. The request parameters have the fields Emulator, Hub which takes the values True/False. Based on the value of this field, identifier component determines which resource type requested for.

Checking whether the resources is available:

Once the resource type is identified, the resource allocator should find the requested resource. It should be able to determine whether the requested resource is already available or need to create new resource. Here we use dynamic allocation strategy which will be explained later. So this component checks for the request resource type already running, here we check whether there are already servers available. Here the available server may be of different types, like ubuntu, windows or redhat, we also check whether the requested type of server is available. Once we found the server available, we should also checked whether the instance is running.

Checking the status of the existing server is handled through the AWS API's. We used one of the API which gives us the status of instance whether running/stopped. Based on this result we determined whether the requested resource is available or not. These operations will be explained in later sections.

Create/ Use resource requested:

If the instance is running, we need to create new server and assign the newly created server. Else use the existing server which is stopped and re run the instance. Both these operations are handled from the API's available in amazon AWS. This strategy of creating a new instance is mainly considered as dynamic allocation strategy. Because new instance will be created based on the demand. If the demand decreases, the available instance will be used and assigned to the user. But if the demand increased we can allocate the resources by creating new resources. Hence we can accommodate as many request as amazon allows to create instance.

Once the requested resource is created/re-started, we will wait till the instance state gets changed to running from pending before we give the information back to the user. This is because, we need some information about the instance like public ip address which will be generated after the status changes to running. Hence we wait till the instance state is running.

Storing the request information for the dashboard:

Once the instance is started, we collect the information about the instance, like key Pair file, public DNS name etc., Once these information are collected, we are storing these values to the database so that we can use these information in dashboard. Dashboard are used by the client to see the status of requested resource and also to stop or start the instance. Along with saving the instance. We have used JDBC connection to store the values in database.

For this project, after researching some of the algorithms, we have used a specific algorithm to allocate resource which is called Dynamic Resource Allocation Strategy (DRAS). Although we found other algorithms like Round Robin, Priority based algorithms align close to the problem definition but those algorithm does not satisfy all the requirements we needed. Hence we choose DRAS among all those.

Dynamic resource allocation algorithm:

In this algorithm, the request are being served dynamically. Which means, if the resource are not available at the time request comes, resources will be dynamically created and assigned. This algorithm is explained clearly by using following pseudo code.

```
Resource Allocator (Request)

List<resourcestype> resourcesTypeList = identifyResourceType(Request)

for each resourcesType in resourcesTypeList

if resourcePool.has(resourcesType)

restartResource(resourcesType)

removeResourcesPool(resourcesType)

else
resource = createResource(resourceType)

end if

end for
```

The algorithm is explained below:

Each time request comes, we first check for the resources type requested for. Once we found out all the request Type, we check for the resource pool whether the resource is already available. If the resource is already in the resources pool restart the instance and return.

If not we are creating new resource and assign it to the user request.

Hence each time a request comes, either the existing resource from pool is fetched or restarted else new resource will be created. Hence as long as the number of requests served are less than the available resources in resource pool no new resource will be created.

This makes sure that no resource will be created as long as requested resource is already available. Hence maximum utilization of available resource will take place. Once the number of requests exceeds the resources available in resources pool, a new resource will be created. Hence this ensures that user requests will always be served. Following is the flow chart for the algorithm.

Flow Chart:

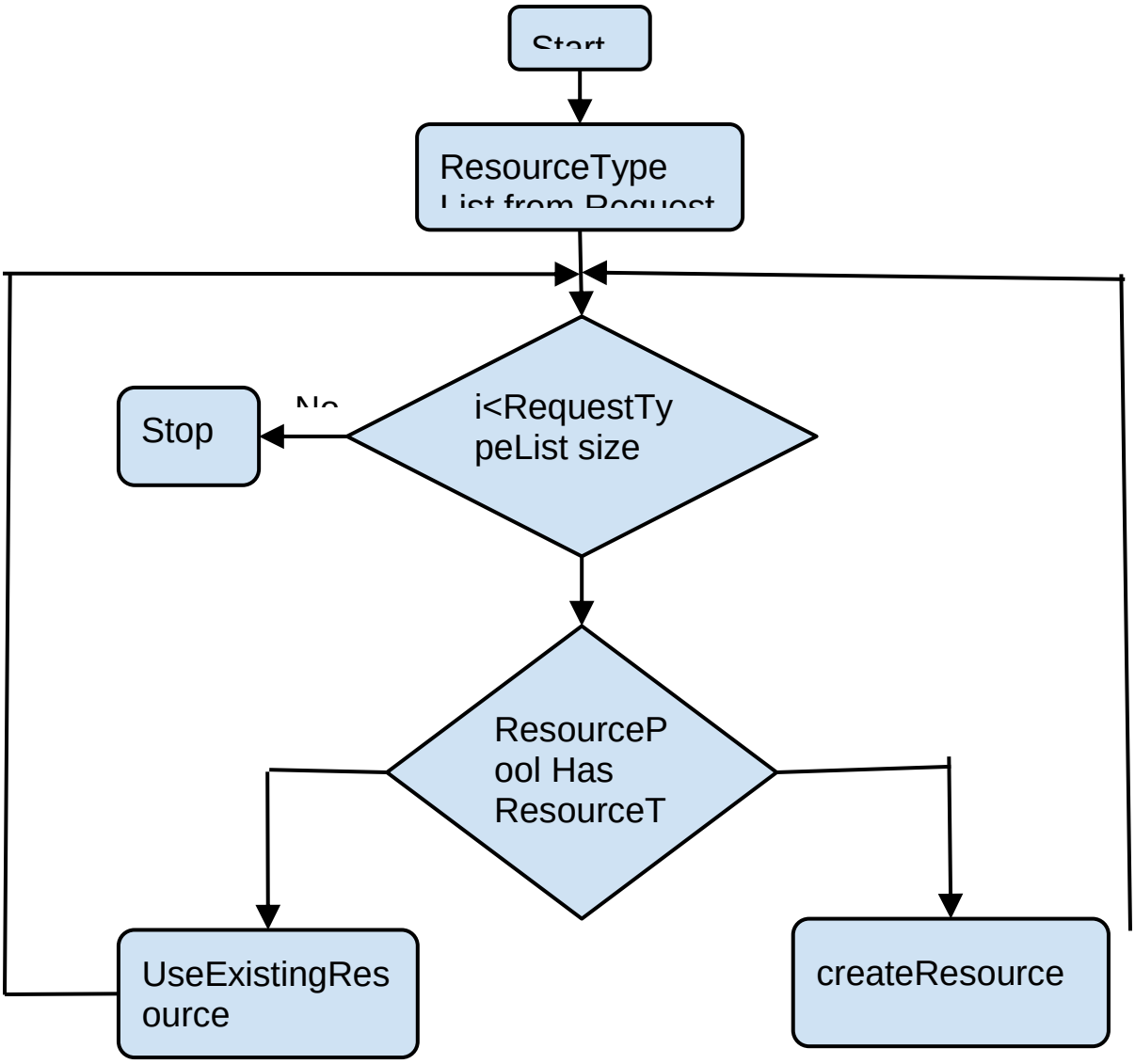


figure 11. Comparison of Resource Allocation Algorithms:

There were couple of algorithms which we considered before choosing the DRAS algorithm. Round Robin and Ant Colony algorithms. Both of this algorithm were suited well when we are having static job which are running. So that we can allocate the resource parallelly and run all these jobs concurrently. Hence we can also use sharing of resource when we have jobs to run which has specific set CPU cycles required.

Ant Colony Algorithm:

The details of the resources in the cloud computing environment and the network topology is Unknown. In fact there is no fixed topology and structure for a given infrastructure, which makes it difficult to determine the location and quality if the computing resources for the data nodes. Ant algorithm can be used to find out the computing resources in an unknown network topology.

Psuedocode:

START

Initialize environment variables

Initialize the default pheromones on the trail selected by the network ants.

Read the connection matrix for the nodes in the network.

WHILE (simulation is running)

START the FORWARD ant from the source node.

CHECK the transiton probability at the node

ASSESS the pheromone levels.

MOVE the ant to the next node based on the calculation.

INCREASE the pheromone at the source node.

IF (ant has reached its destination node)

START the BACKWARD ant from the destination node.

TRACE the path back to the source node

END IF

Update the statistics for the all the resources (regions).

END WHILE

Return back the regions sorted list according to calculation.

END

Round Robin Allocation Algorithm:

Round robin algorithm is a network scheduling kind of algorithm. It has different types of serving in it. We had considered weighted round robin (WRR) method where each request will be considered as one packet of resource queue and instances will be allotted based on the weight given to the particular request type.

Pseudocode for WRR algorithm.

```
// calculate number of requests to be served each round by connections
for each flow f
  f.normalized_weight = f.weight / f.mean_request_size
  min = findSmallestNormalizedWeight
  for each flow f
  f.requests_to_be_served = f.normalized_weight / min

// main loop
loop
  for each non-empty flow queue f
    min(f.requests_to_be_served, f.request_waiting).times do
      serveQueue f.getRequest
```

On the other hand, since we had load balancer component which was using priority based and honeybee algorithm which will disperse the request equally between different regions of Amazon. Hence because of undefined length of user requests, we are using modified version of Round Robin algorithm which is Dynamic Resource Allocation Strategy.

Amazon AWS API List :

We used many Amazon AWS API's to handle operations on instances. Following are the Each API we used in resource allocation component to manage the instances. Amazon instances are used to provide Server, Hub and also Emulator.

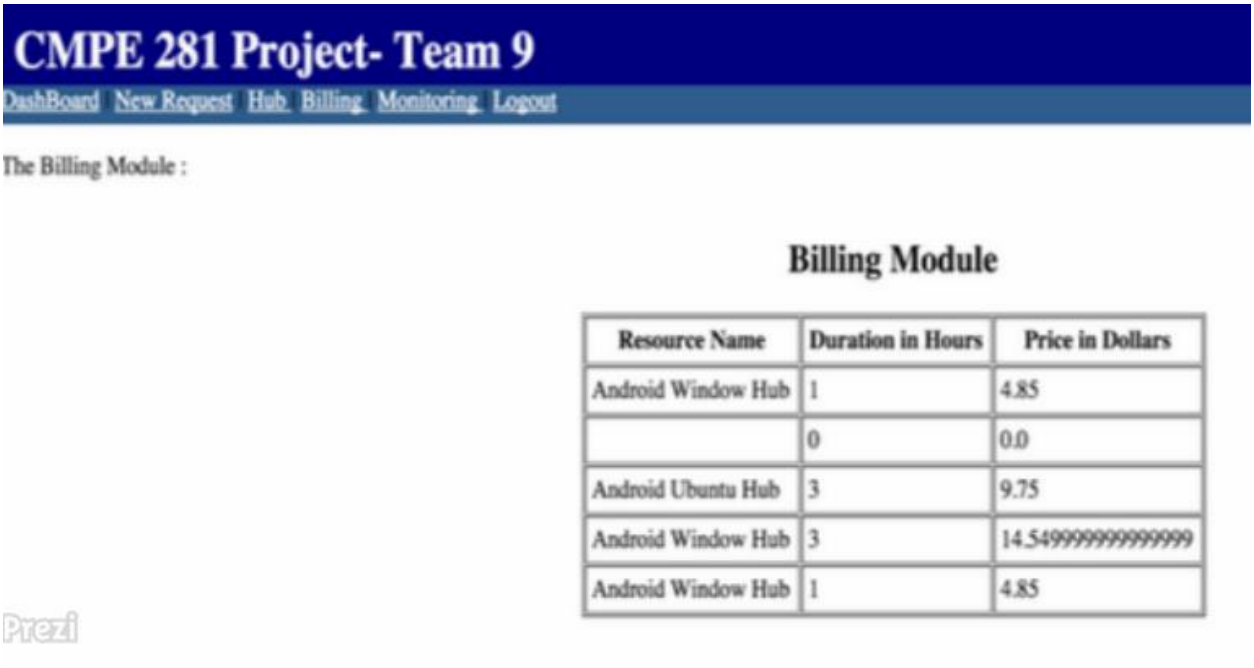
- AmazonEC2Client - This is the main object which we can be used to connect to AWS specific regions.
- ProfileCredentialsProvider - This API will get the credentials file from the path of the AWS login setup
- checkInstanceAvailable - This API checks if the specific instance type is available and if yes,returns true
- checkInstanceRunning- This check if the available instance type is running or in stop state
- DescribeInstancesRequest- This API describes the state of the instance type
- createNewInstance- This method creates new instance of particular type if the instance type is already not present.
- useExistingInstance- This API is used to identify the particular instance type to restart.
- DescribeInstancesRequest- This API gives the list of all the instances of specific type.
- restartTheInstance - This method restarts the stopped instance
- CreateSecurityGroupRequest -This API creates security group for the new instance type
- CreateKeyPairRequest - This API creates new key pair for a instance.
- RunInstancesRequest- This API runs the already created isntance type from stop state.
- stopInstance - Stops the particular instance.
- getConnection - This API attempts to establish connection to the given database URL.

7. Billing Module

Billing and Cost module in AWS is one among the Amazon web services which is used by the users to bill the Amazon instances, monitor the cost incurred and also look at the spent amount. Billing and cost Management model is important so that the users can estimate the costs and plan them accordingly.

How billing module works?

The billing module receives the input from the resource provisioning and monitoring module. Pricing will then be calculated based on the type of resource allocated. The flavor of the resource and the time frame it is used are the values that are considered for the calculating the billing metrics. Emulators and hubs are priced at a higher rate when compared to the other amazon instances/resources.



8. Dashboard and Monitoring.

We have built in a custom AWS dashboard to make it more data visualisable. AWS as a built in dashboard which supports only line chart. Using Cloudviz and google tools data visualization can be achieved.

The 3 main components:

1) AWS APIs to get metrics data.
2) Cloudviz to collect metrics data from cloud.
3) Google dashboard tools to display data.

AWS APIS Used:

- ProfileCredentialsProvider – Provides credentials to login to AWS.
- AmazonCloudWatchClient – Access cloud watch clients for data.
- GetMetricStatisticsRequest- Get metrics statistics like namespace, metrics, statisctics, dimensions, instance type.
- GetMetricStatisticsResult

Below is the sample code to get metrics data:

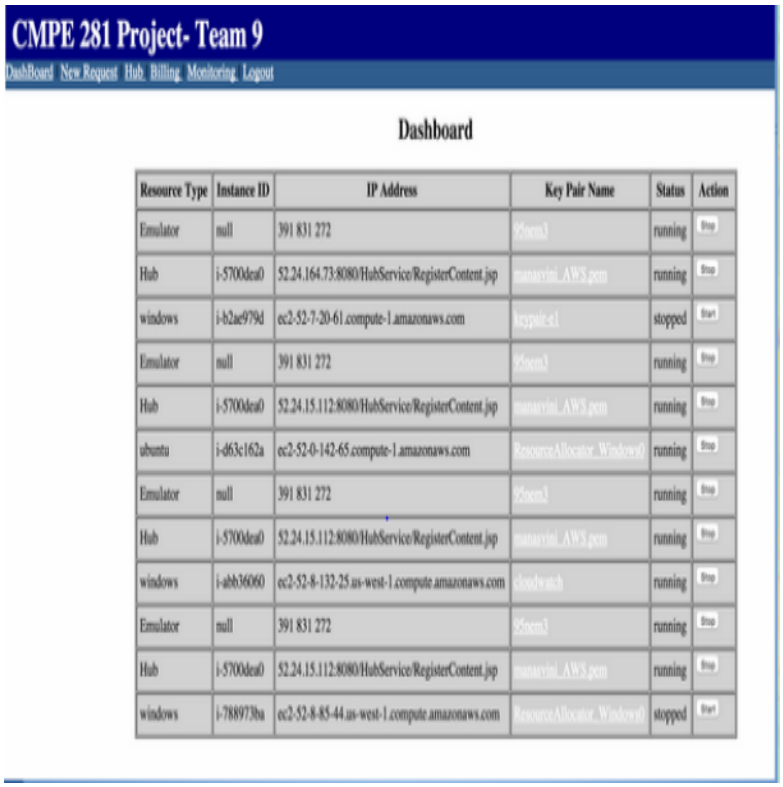
```
public static void main(String[] args) {
                    AWSCredentials credential = null;
                    String instanceId = "i-abb36060";


    try {
       credential = new ProfileCredentialsProvider("default").getCredentials();
    } catch (Exception e) {
       throw new AmazonClientException(
            "Cannot load the credentials from the credential profiles file. " +
            "Please make sure that your credentials file is at the correct " +
            "location (/Users/Mana/.aws/credentials), and is in valid format.",
            e);
    }
                    try {

                    AmazonCloudWatchClient cw = new AmazonCloudWatchClient(credential) ;

                    long offsetInMilliseconds = 1000 * 60 * 60 * 24;

                    GetMetricStatisticsRequest request = new GetMetricStatisticsRequest()
                    .withStartTime(new Date(new Date().getTime() -
offsetInMilliseconds)).withNamespace("AWS/EC2")

                    .withPeriod(60 * 60)

                    .withDimensions(new Dimension().withName("InstanceId").withValue(instanceId))
```

```
.withMetricName("CPUUtilization")

.withStatistics("Average", "Maximum")

.withEndTime(new Date());

GetMetricStatisticsResult getMetricStatisticsResult = cw.getMetricStatistics(request);


double avgCPUUtilization = 1;

List<Datapoint> dataPoint = getMetricStatisticsResult.getDatapoints();

for (Object aDataPoint : dataPoint) {

Datapoint dp = (Datapoint) aDataPoint;

avgCPUUtilization = dp.getAverage();

System.out.println(instanceId + " instance's average CPU utilization : " + dp.getAverage());

System.out.println( avgCPUUtilization);


}

System.out.println( avgCPUUtilization);


} catch (AmazonServiceException ase) {


        System.out.println( "error" + ase.getMessage());
        System.out.println( "error1" + ase.getStatusCode());
        System.out.println( "error2" + ase.getErrorCode());
        System.out.println( "error1" + ase.getErrorType());
        System.out.println( "error2" + ase.getRequestId());

}


}
```

Cloudviz Data:

Cloudviz collects the following data for data vizualisation.

"container": "webcpu",
"cloudviz":
{
"namespace": "AWS/EC2",
"metric": "CPUUtilization",
"unit": "Percent",
"statistics": ["Average"],
"period": period,
"range": range,
"timezone": timezone,
"region": region,
"calc_rate": false,
"cloudwatch_queries":
[
{
"prefix": "dashboard",
"dimensions": { "InstanceId": "i-090adccb"}
},



Conclusion

The main focus of this work is provide a mobile Infrastructure as a service to the mobile testing as a service users. The goal of this project is to provide an anywhere, anytime solution to the MTaaS users, there by scaling out the applications effectively.

References:

[1] Karaboga, D. (2010). Artifical bee colony algorithm. Retrieved December 20, 2014, from http://www.scholarpedia.org/article/Artificial_bee_colony_algorithm

[2] http://www.arubacloud.com/cloud-computing/load-balancers.aspx

[3] http://en.wikipedia.org/wiki/Cloud_computing

[4] http://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html

[5] http://en.wikipedia.org/wiki