A report on

Various Methods Of Load Balancing in Cloud Computing

Done by

Names of Students	Roll No
Aviral Nigam	B090871CS
Snehal Chauhan	B090850CS
Varsha Murali	B090484CS

Guide
Vinod Pathari
(Asst. Professor)

Department Of Computer Science And Engineering
National Institute of Technology Calicut
Calicut, Kerala 673 601

2012 - 2013

Department Of Computer Science And Engineering

National Institute of Technology Calicut

Certificate

This is to certify that this is a bonafide record of the project presented by the students whose names are given below during 2012-2013 in partial fulfilment of the requirement of the major project.

Names of Students	Roll No
Aviral Nigam	B090871CS
Snehal Chauhan	B090850CS
Varsha Murali	B090484CS

Guide

Vinod Pathari
(Asst. Professor)

Date

Acknowledgments

We would like to express our sincere gratitude to our guide, Mr. Vinod Pathari, Asst. Professor, Computer Science and Engineering Department, for his invaluable help, support and guidance throughout the course of the project. We would also like to thank the faculty members of Computer Science and Engineering Department for their suggestions and inputs which helped us in completing the project. Our sincere appreciation to our fellow students for their co-operation and support.

Aviral Nigam
Snehal Chauhan
Varsha Murali

Abstract

As Cloud Computing is an emerging field, many improvements are being proposed to provide users with better services and facilities. This work deals with the illusion of infinite resource availability on demand, one of the new aspect in Cloud Computing. Initially, few proposed algorithms for Cloud Computing have been implemented to understand load balancing in cloud computing and a new hybrid algorithm has been proposed. Further, a combination of forecasting models and game theoretic approaches have been proposed so as to continue providing this illusion without any glitches. This implementation is a new way of looking at the problem and with co-ordination from different providers it becomes possible for each provider to decide his best strategy. This work provides an efficient way for the cloud provider to decide on his strategies to execute a job i.e., whether to use his own services to execute (self-execute) or to pay rent for the services to other cloud providers. Forecasting has been done to get an idea of previous demands. Game theory is a concept that is generally applied for economical undertakings, generally for the current period, and is a good way to engage it in deciding the strategies adopted by an organization. Hence the design considers both the previous as well as current demand to decide on the provider's strategy making the results more accurate. This work combines two different approaches and based on their results decides upon a strategy that has minimum deviation. The results obtained from this utility function show an almost equal distribution of Rent and Self-execute strategies. This work can be enhanced by including more factors, especially of financial importance, in the utility and providing methods for scalability.

Contents

Chapter 1

Introduction

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [1]. This model enables users to access supercomputer-level computing power elastically on an on-demand basis, freeing the users from the expense of acquiring and maintaining the underlying hardware and software infrastructure and components. "A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers" [1]. Cloud computing, as a current commercial offering, started to become apparent in late 2007. It was intended to enable computing across widespread and diverse resources, rather than on local machines or at remote server farms. Where these clusters supply instances of on-demand Cloud computing; provision may be comprised of software (e.g. Software as a Service, SaaS) or of the physical resources (e.g. Platform as a Service, PaaS) [2].

Load Balancing is the process of distributing the load among various nodes of a distributed system when it becomes difficult to predict the number of requests that will be issued to a server. It considers factors like execution time, resource availability and requirement among others to improve job response time, throughput, etc. In order to provide better service-level agreements, the cloud provider has to provide such improvements to the user. Load Balancing is a method to distribute workload across one or more servers, network interfaces, hard drives, or other computing resources.

1.1 Report Organization

The report consists of two phases- different load balancing algorithms and a new approach to load balancing.

In Phase I, a basic introduction on various cloud service models has been provided, followed by a literature survey on the existing load balancing algorithms, their advantages and limitations. The phase then continues onto a detailed description of the work done in studying, implementing and analysing three load balancing algorithms. This also includes a hybrid algorithm that has been proposed by us. In the last part of this phase, the basic idea for a new approach along with its motivation have been recorded.

In Phase II, one of the problems in Cloud Computing has been tackled in a new way, the illusion of infinite resources, considering load balancing factors. An introduction on this problem and the Quality of Service protocols that can be achieved has been described. A literature survey has been done on the various approaches of this problem . In the later part of this phase, the design, implementation and analysis of the proposed approach has been explained in detail followed by the limitations of this approach.

Part I

Phase - I

Chapter 2

Introduction

Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services. Cloud service delivery is divided into three models. The three service models are [3] :

2.1 Cloud Software as a Service (SaaS)

The capability provided to the consumer is to use the providers applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser.

2.2 Cloud Platform as a Service (PaaS)

The capability provided to the consumer is to deploy onto the cloud infrastructure, consumer created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure but has control over the deployed applications and possibly application hosting environment configurations.

2.3 Cloud Infrastructure as a Service (IaaS)

The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications.

Load balancing is used to make sure that none of the existing resources are idle while others are being utilized. To balance load distribution, migration of the load from the source nodes (which have surplus workload) to the comparatively lightly loaded destination nodes can be done.

2.4 Problem Statement

- A study of the existing load balancing algorithms in Cloud Computing and implement a new hybrid load balancing algorithm. This will include an analysis of these above algorithms and to draw conclusions based on their execution time.

Chapter 3

Literature Survey

A distributed solution is required, as it is not practical or cost efficient, in many cases, to maintain idle service/hardware provision merely to keep up with all identified demands. Indeed it is not possible, when dealing with such complexity, to fully detail all system future states; it is thus necessary to allow local reasoning, through distributed algorithms, on the current system state. Thus efficient load balancing cannot be achieved by individually assigning jobs to appropriate servers as the Cloud computing systems scale up and become more complex; rather jobs must inevitably be assigned with some uncertainty attached. This gives some immediate possible methods for load balancing in large scale Cloud systems, which are discussed here [3].

In biased random sampling, a nodes in-degree (node capacity) is mapped to its free resources. When a node receives a new job, it will remove one of its incoming edges to decrease its in-degree and indicate that its available resources are reduced. In the same way, when the node finishes a job, it will add an edge to itself to increase its in-degree. The process of adding and removing incoming edges is done by random sampling. Random sampling is the process whereby the nodes in the network are randomly picked up with equal probability. The sampling walk starts at some fixed node, and at each step, it moves to a neighbor node chosen randomly according to an arbitrary distribution. Then, the last node in the course of the sampling walk will be selected for the load assignment [2].

Honeybee Foraging is one of a number of applications inspired by the believed behavior of a colony of honeybees foraging and harvesting food. Forager bees are sent to search for suitable sources of food; when one is found, they return to the hive to advertise this using a display to the hive known as a waggle dance. The suitability of the food source may be derived from the quantity or quality of nectar the bee harvested, or its distance from the hive. This is communicated through the waggle dance display. Honey bees then follow the forager back to the discovered food source and begin to harvest it. This approach has been used for various applications in computing [2].

Experiments have been conducted to assess and analyze these procedures for load balancing claiming that none of the load balancing methods are mutually exclusive and it is possible that combinations could be used to further improve the performance of the algorithms [4].

Chapter 4

Work Done

The algorithms that have been considered for the project are stated below :

- Biased Random Sampling
- Honey Bee Foraging
- Hybrid Algorithm

4.1 Theory

4.1.1 Biased Random Sampling

In this type of approach we need to construct a network comprising of virtual nodes which represent all the resources present on the server to represent the total load. The indegree of the node corresponds to the free resources of the server such that [5]:

- Whenever a node executes a job, it deletes an incoming edge, which indicates reduction in the availability of free resources.
- After completion of a job, node creates an incoming edge which indicates an increase in availability of the resources.

The working of the algorithm can be understood by the following pseudo code :

```
b = log n  //n - network size

For any node that recieves a new process
    /*Create and send token to
    randomly selected neighbour*/
    job.walklength = 1
    /*Select a neighbour using
    probability based function*/
    Randomselect(node)
    /*Send job to selected neighbour*/
    Send(job,neighbour)

For any node that recieves token
    /*Update token if needed and send it
    to neighbour selected by the function*/
    if job.walklength < b then
        neighbour = Randomselect(node)
        Send(job,neighbour)
    else
    /*Execution of job on node
    indiacted by the token*/
        Execute(job)
    endif
```

4.1.2 Honeybee Foraging

The first job, on being sent into the network of servers acts as a scout providing information on the availability of resources at each server. This information is published on an advert board which is referenced by the incoming jobs. Based on the fitness function, jobs attach themselves to the server and simultaneously update the advert board.

The working of the algorithm can be understood by the following pseudo code :

```
/*Scout on entering network*/
TraverseNetwork(scout)
CreateAdvertBoard()

For all incoming jobs
    /*Checking for the best server*/
    node = fitness(job)
    /*Job Execution*/
    Attach(node,job)
    /*Updating advert board with
     current status of resources*/
    UpdateAdvertBoard(node,job)
```

4.1.3 Proposed Hybrid Algorithm

This algorithm is a combination of biased random sampling and honeybee foraging. The incoming jobs are scheduled on the basis of their job size.

It has been experimentally observed while working on the above two algorithms that the biased random sampling gives a more consistent result during the execution of jobs of size one.

The working of the algorithm can be understood by the following pseudo code :

```
/*Jobs on entering network*/
if jobsize==1 then
    BiasedRandomSampling(job);
else
    HoneyBeeForaging(job);
endif
```

4.2 Implementation

An array of 100 jobs with random jobsize and execution time and an adjacency matrix of a fixed network of servers are given as inputs.The server network being created by the adjacency matrix forms a directed graph which allows for easy traversal between the servers.

- **Nodequeue:** This is a list of all the jobs being executed by the particular server including the job size, execution time and the time at which the job was allocated to the server.
- **NodeArrayList:** The list of all the servers in the network and their accompanying node queue are contained here.

The increaseindegree(), common to all the algorithms, traverses the array list associated with each job and checks if any job that was being executed by each server has been completed. This is done by comparing the current time with the sum of the execution time required for the job and the time at which it was allocated to the server. On completion of the jobs, the array list is updated accordingly and the resources available with the server are incremented.

The timer functions are as follows: jobs are sent every one second and if the job cannot be allocated presently to any server due to lack of resources then it waits for two seconds before trying again.

4.2.1 Biased Random Sampling

- **BiasedRandomWalk:** The selectwalk() initially checks for the availability of the resources with the help of increaseindegree() and uses the hopcount which gives the length of the walk. It randomly chooses a server and checks if the job can be allocated to be followed by the invocation of selectneighbour(). If no resources are available, then it randomly chooses another server and continues the above process. The selectneighbour() probabilistically chooses the next best server based on the resources available in each of the neighbor. This neighbor selection is done till the hopcount value is reached. The send() compares the resources between the last two servers and updates the token with the better value. It then updates the resource information of the server to which the job has been allocated and finally returns the server node.

- **GraphBiased:** The creategraph() contains the server network (represented by an adjacency matrix) and also the information pertaining to the availability of resources with each server. An object of the BiasedRandomWalk class is created and we check for the availability of resources in the network. If resources are unavailable, then the next job to be allocated waits. Only when resources are made available the process of random selection starts.

4.2.2 Honeybee Foraging

- **Fitness:** The class contains select() which initially calls increaseindegree() to update the current resource status of the servers. A traversal of the server network is done whereby for each server we calculate the difference in the resources currently available and the resources requested. The job is then allocated to the server for which the difference is small. The resource available for that server is decremented accordingly and it finally returns the best fit server.

- **GraphHoneyBee:** The server network is represented with an adjacency matrix. The creategraph() forms the network and calculates the available resources with each server. This is called when the scout job is sent. The fitness() is invoked which does the job allocation. If job could not be allocated because of lack of resources, then the current job waits and calls the increaseindegree() to get updated resource information.

4.2.3 Proposed Hybrid Algorithm

This algorithm makes use of the BiasedRandomWalk and Fitness classes from the above two algorithms.

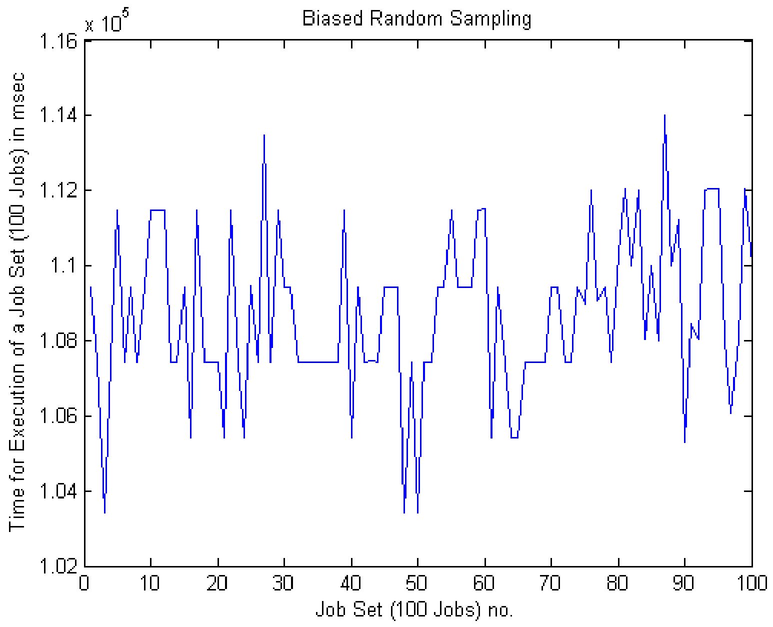
- **Hybrid:** The creategraph() contains the server network (represented by the adjacency matrix) and also about the resources available with each server. Objects of BiasedRandomWalk and Fitness classes are created. The job() receives each job and then checks if any resources are available in the network and then calls the selectjob() (after waiting, if resources were unavailable). If the job requires only one resource, then the biased random sampling algorithm is used while the honeybee foraging algorithm is used otherwise. The selectjob() chooses the algorithm for each particular job and then sends the job to the respective functions in BiasedRandomWalk or Fitness.

The set of 100 jobs is executed by the algorithm and its total execution time is recorded. This process is repeated for 100 times and an average execution time is obtained for each algorithm.

4.3 Analysis

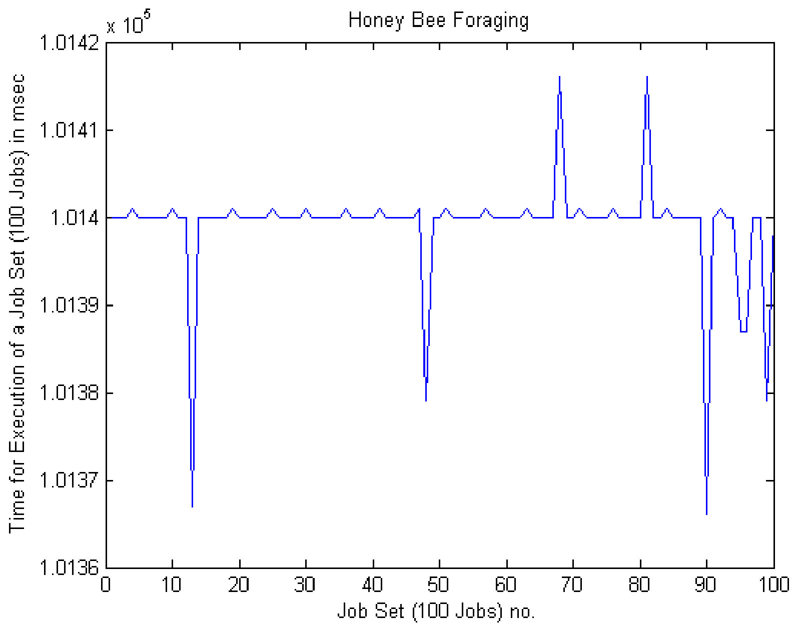
4.3.1 Biased Random Sampling

In the biased random sampling algorithm, for every batch of hundred jobs, small changes in the range of seconds were obtained. The change which occurred is because of the delay being caused due to unavailability of resources. A random node is selected only initially and the walk is taken based on the probabilistic function and hence the walk only goes up to the value of hop-count (which is the same) for every allocation of job. The time taken for load balancing of every job varies by seconds because the algorithm looks for a server with maximum resources available and then tries to allocate the job. So, overheads are obtained because of finding the path for every job followed by comparing it; even though only finally the job size is checked.



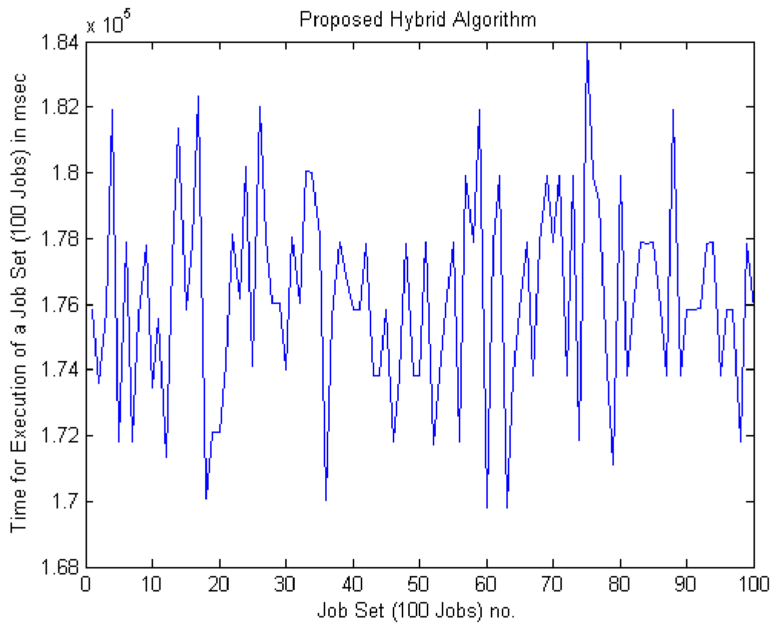
4.3.2 Honeybee Foraging

In the honey bee foraging algorithm, the running time for the batch of jobs is almost same only varying by milliseconds. This change is due to the delay caused by the unavailability of resources since the other jobs are utilizing all the servers in the network. The portions of consistent timing are due to the traversal of the same advert board by every job being sent into the network. Since the advert board is a graph traversal with a list of jobs attached with each node, the algorithm takes almost similar time for the traversal in each case since the job size does not vary drastically.

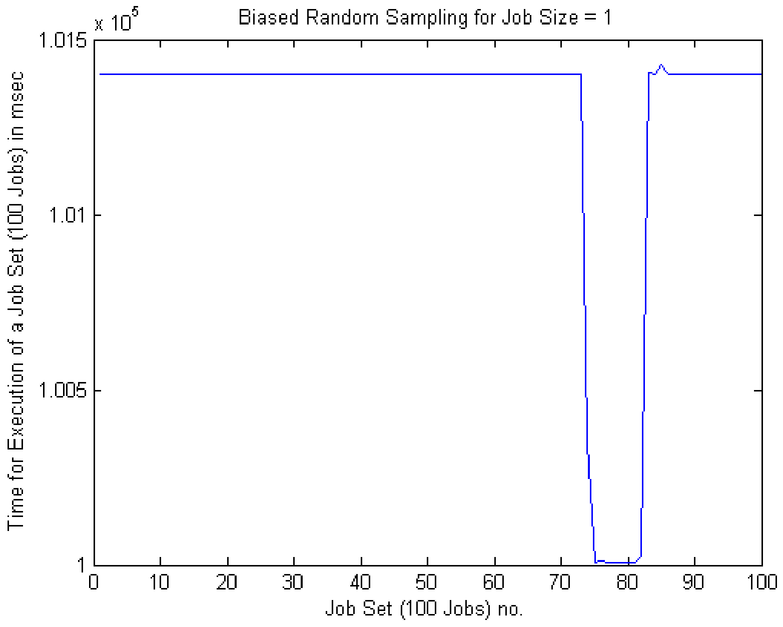


4.3.3 Proposed Hybrid Algorithm

In the proposed hybrid algorithm, the timing pattern shows a divergence in the range of seconds with almost no perfectly same running time in a batch of ten. This variation is due to the unavailability of resources which causes a delay. This is because of the server being chosen, for a particular job, by both the algorithms. While biased chooses the server with the maximum number of resources, honey bee goes in for a more optimal selection taking a best fit approach.



The hybrid algorithm uses the biased random sampling only for a probability of 1/5 (job size has been bound to a maximum of 5) to get a better consistent timing pattern as has been experimentally observed below; whereas the honeybee algorithm is selected with a probability of 4/5. If we consider a case where a biasing function has been applied such that the probability of choosing either of the algorithm is same, then a more inconsistent graph with larger number of peaks and dips are obtained because of the above explained contradiction.



All the three algorithms show changes only in the range of milliseconds to seconds for a batch of hundred jobs being run hundred times. While biased takes 10870 ms, honey bee runs in 10140 ms and hybrid in 17617 ms for a batch of 100 jobs on an average. This clearly shows that the honey bee algorithm is the best approach of all the algorithms considered. Though honey bee has a more consistent run time (the inconsistency occurs due to reaching a state where no resources are available) as compared to biased (the inconsistency being caused by the delays due to the checking of job size only finally), the hybrid algorithm has a highly inconsistent run time graph because of entering the state more due to the contradiction in server selection between the two algorithms.

Chapter 5

Proposed Work

The problem, till now, has been addressed by scheduling one job at a time to the best server and thus following the same strategy for a group of jobs. The problem can also be viewed as scheduling a group of jobs to the best servers at one instant of time. This approach has been taken because it may not be necessary that the best allocation for each job is the best for the given group of jobs i.e., it is possible to have a collective best outcome which may be better than a collection of individual best outcomes. Hence this problem can be looked at from the technique of game theory which fundamentally states the same.

Game theory is the study of strategic decision making. Strategies and payoffs are calculated and a payoff matrix is formed which will contain the values obtained on applying a utility function. This approach has been widely used in various fields of computer science. The matrix given below can be obtained for load balancing problem where each job can have different affinity towards different servers.

Job/Server	S_1	S_2	S_3
J_1	5	2	5
J_2	9	1	3
J_3	4	1	8

Such kind of representation has motivated us to give a game-theoretic approach to this problem. In the coming months, the objective will be to study game theory and to use it to obtain optimal solution for load balancing in Cloud computing.

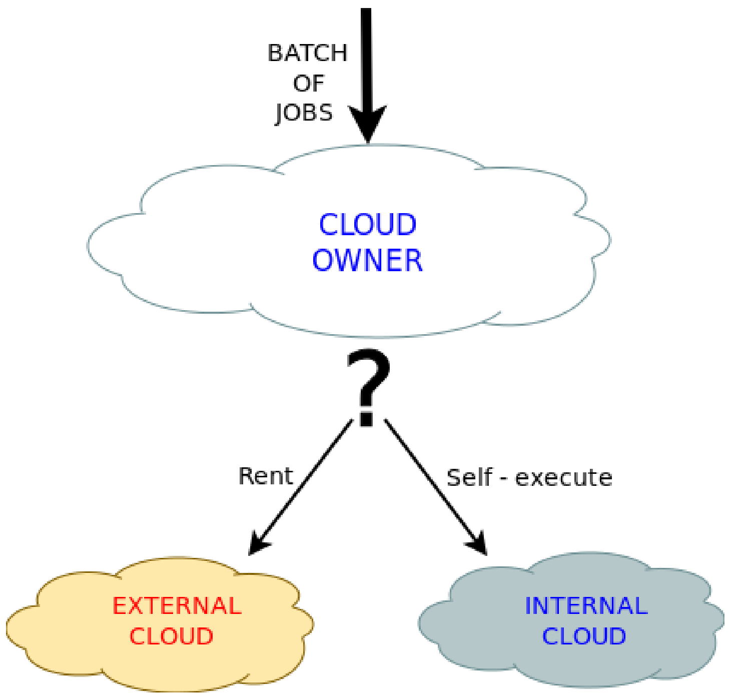
Part II

Phase - II

Chapter 6

Introduction

Quality of Service (QoS) is the resource reservation control mechanism in place to guarantee a certain level of performance and availability of a service. It provides a level of assurance that the resource requirements of an application are strictly supported [6]. It is possible that the resource requirement of a user may not be supported by the cloud provider. In such scenarios, the cloud provider has to provide a means of executing the user's load. The provider has to decide the appropriate strategy such that the user's needs are met. One of the most interesting aspects in Cloud Computing is the feeling of availability of 'infinite' computing resources that the cloud provider tries to distribute to the user in an elastic way [7]. The user does not fully realize the internal allocations while demanding for more resources.



For this infinite demand of the user to be met, the cloud provider has to find ways to do it without incurring any loss. One of the approaches to do it is to calculate the number of times the provider does not have the capacity to execute the load and based on that draw up an agreement with another provider to execute the load. This calculation can be predicted from past data through forecasting and by game theory (to include the current data also). The method of standard deviation can help in deciding the strategy of the cloud provider.

6.1 Problem Statement

Identification of a Rent or Self-execute strategy in Clouds for an incoming batch of jobs. The strategy adopted defines when a Cloud owner decides whether a batch can be executed using an internal Cloud or has to pay rent to execute it in an external Cloud provider.

- Use of forecasting methods to predict the strategy.
- Applying a game theoretic approach to decide the strategy.
- Use the above two approaches to decide on one final strategy.

Chapter 7

Literature Survey

An aspect of Cloud Computing is the illusion of infinite computing resources available on demand, thereby eliminating the need for Cloud Computing users to plan far ahead for provisioning. For this, realizing the economies of scale afforded by statistical multiplexing and bulk purchasing requires the construction of extremely large data-centers. Building, provisioning, and launching such a facility is a hundred-million-dollar undertaking [8].

A system composed of a virtual network of virtual machines capable of live migration across multi-domain physical infrastructure have been constructed. By using dynamic availability of infrastructure resources and dynamic application demand, a virtual computation environment is able to automatically relocate itself across the infrastructure and scale its resources [9]. Thus the QoS improvements can be met using this virtual machine setup. Depending on the type of application, the generated workload can be a highly varying process that turns difficult to find an acceptable trade-off between an expensive over-provisioning able to anticipate peak loads and a sub performing resource allocation that does not mobilize enough resources. These properties can be leveraged to derive a probabilistic assumption on the mean workload of the system at different time resolutions [10].

There are many proposals that dynamically manage Virtual Machines by optimizing some objective function such as minimizing cost function, cost performance function and meeting QoS objectives. The objective function is defined as Utility property which is selected based on measures of response time, number of QoS, targets met and profit etc. [9]. This Utility property can then be adjusted to include factors of individual importance. They have been modified to meet needs according to current demand. But they may prove complex and may not work in a practical virtualization cloud system with real workload. These approaches work best for stationary demands and may not give optimal solution for dynamic resource requirements. The utility property can be modified to include dynamic demands and allocation and hence has to be formulated accordingly.

Chapter 8

Work Done

8.1 Theory

In Cloud Computing, an organization has to decide whether it has to Self-execute or pay Rent for the computing facilities. Considering factors like execution time, the number of resources required, number of jobs in a batch and the waiting probability, forecasting is done to determine the strategy. Since these data are generally random in a Cloud environment, using forecasting methods that depend on seasonal data cannot be considered. Methods that rely on level and trend based data have been used [12]. Further, considering only current data, a game theoretic approach is applied which decides the optimal strategy for any situation.

8.1.1 Utility Function

A utility function translates outcomes into numbers such that the expected value can be used to calculate certainty equivalents for alternatives in a manner that is consistent with a decision maker's attitude toward risk taking [11].

$$Utility = \frac{\frac{e}{t_{avg}} + n^{1-p}}{s}$$

where:
e - Execution time of current job.
t_{avg} - Moving average of the execution times of all the previous jobs in the batch.
n - Number of resources required by the current job.
p - Waiting probability.
s - Batch size.
The strategy adopted is to maximize the utility. Taking into consideration resource utilization, execution time and throughput, all the factors of load balancing have been given weightage.

(a) $\frac{e}{t_{avg}}$ deals with execution time and average previous throughput.

(b) n^{1-p} deals with resource allocation. If number of resources required is less then waiting probability tends to be less. The job gets executed faster because of lesser waiting probability. So to maximize utility; if $\frac{e}{t_{avg}}$ is less, then this factor need to be higher to increase the utility value. Hence n has been raised by a factor of $(1 - p)$.

The above terms are added because of this compensation, i.e., if one factor is less then the other factor is higher and vice-versa, and also because it is a consideration of all the factors of load balancing, this operation gives each of these factors a non-trivial weightage in the utility function. Division by the batch size helps in maintaining uniformity between batches.

8.1.2 Forecasting Methods

Forecasting involves making projections about future performance on the basis of historical and current data. The need for forecasting stems from the time lag between awareness of an impending event or need and the occurrence of that event. Forecasting methods are applied to predict the future and depending on the data we can classify it into level-based, trend-based and seasonality-based.

Exponential Smoothing

Exponential smoothing forecasts are forecasts for an integrated moving average process; however, the weighting parameter is specified by the user rather than estimated from the data. Experience has shown that good values for the weighing parameter $= \alpha$ are between 0.05 and 0.3 [16].
The single exponential smoothing operation is expressed by the formula :

$$FORECAST(t+1) = \alpha * demand(t) + (1 - \alpha) * FORECAST(t)$$

where:
$demand(t)$ - Current actual value of the series.
$FORECAST(t)$ - Forecast value at the current period.
$FORECAST(t+1)$ - Forecast of $demand(t+1)$.

Holt's Method

Holt's Method is a double exponential smoothing method used for forecasting, when there is a linear trend present in the data. The method requires separate smoothing constants for slope and intercept.

$$FORECAST(t+1) = a(t) + b(t)$$

$$a(t) = \alpha * demand(t) + (1 - \alpha) * (a(t-1) + b(t-1))$$

$$b(t) = \beta * (a(t) - a(t-1)) + (1 - \beta) * b(t-1)$$

where:
$a(t)$ - Level which represent the smoothed value up to and including the last data.
$b(t)$ - Slope of the line representing the trend.
$demand(t)$ - Current actual value of the series.
$FORECAST(t+1)$ - Forecast of $demand(t+1)$.

8.1.3 Forecast Design

For each batch, a forecasting method is applied with the existing utility values to compute the utility value for the next batch. The forecasting methods can depend on the trend present in these input utility data. If no trend is present, exponential smoothing can be applied and the presence of trend will shift the forecasting to Holt's Method.

```
/*Choosing Forecasting Method*/

if $trend == true$ then
    $method \leftarrow Holt's\ Method$
else
    $method \leftarrow Exponential\ Smoothing$
end if
```

The computed forecasted value is compared with the actual utility value of the batch (an average of the utility values of all the jobs in the batch) to understand the computing facilities required to execute the batch.

```
/*Choosing Batch Strategy by Forecasting*/

if $forecast\ value < average\ utility$ then
    $strategy \leftarrow Rent$
else
    $strategy \leftarrow Self - execute$
end if
```

When the value of forecast is obtained, the system prepares itself to provide computing facilities atmost equal to that of the utility value forecasted. So if the forecasted value is less than the actual utility value, then the system was not prepared to handle that load as it forecasted only lesser facilities . So the cloud owner pays rent to an external cloud who can provide the necessary computing facilities for executing the batch. If found otherwise, the cloud owner will execute the batch with the facility he owns.

8.1.4 Game Theoretic Design

Game theory is the formal study of conflict and cooperation. Game theoretic concepts apply whenever the actions of several agents are interdependent. The concepts of game theory provide a language to formulate, structure, analyze, and understand strategic scenarios.

In this approach, two utility values are calculated for two waiting probabilities, p_1 and p_2 where $0 <= p_1 <= 0.5$ and $0.5 < p_2 <= 1$.The strategies of Rent and Self-execute then randomly take the u_1 and u_2 values:

```
$/^{*}On\ Random\ Coin - toss^{*}/$

if $Coin - toss == 1$ then
    $Rent \leftarrow u_1$
    $Self - execute \leftarrow u_2$
else
    $Self - execute \leftarrow u_1$
    $Rent \leftarrow u_2$
end if
```

The above random strategy can be interchanged without much effect. So, for each batch of jobs a matrix $M_{BatchSize\,X\,2}$ is created with utility values for Rent and Self-execute strategies. The maximum utility values for each strategy is then selected alongwith its corresponding rows resulting in a matrix $M'_{2\,X\,2}$. A method of mixed strategy calculation is applied to M' which chooses the strategy with the higher probability p.

8.1.5 Standard Deviation

Standard deviation (represented by the symbol sigma, σ) shows how much variation or "dispersion" exists from the expected value. Standard deviation is commonly used to measure confidence in statistical conclusions.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

The strategy given by forecasting and game theory may be different. In such a case, further decision has to be taken as to which strategy should be adopted. The method of standard deviation in each conflicting batch is applied about the points:

$\mu_1 = forecasted\ value \quad /^{*}Forecast\ approach^{*}/$
$\mu_2 = equilibrium\ value \quad /^{*}Game\ Theory\ approach^{*}/$

where equilibrium value is obtained as:

$$p * utility_p + (1 - p) * utility_{1-p}$$

The final strategy is given by the approach that has minimum standard deviation.

```
/*In case of Conflict of Strategies*/
$\sigma_1 \leftarrow$ Standard Deviation about $\mu_1$
$\sigma_2 \leftarrow$ Standard Deviation about $\mu_2$
if $\sigma_1 < \sigma_2$ then
    $strategy \leftarrow strategy_{Forecasting}$
else
    $strategy \leftarrow strategy_{GameTheory}$
end if
```

8.2 Implementation

A batch of jobs (maximum 25 jobs per batch) having randomly generated values for execution time, number of resources required and waiting probability is initially created and 1000 such batches are initialized. The first seven batches are made to randomly adopt a strategy because of the window size taken in the approach.

The utility value for each job is calculated using the utility function and the average of these values are obtained for a batch. While applying a forecasting model, if a linear trend is observed in the average utility values for a window size of 7 consecutive batches, Holt's method is used. Otherwise exponential smoothing is used to calculate the forecast value for the next batch of jobs. A relaxation of 3 values has been provided in the forecasting model so that an almost equal distribution of Holt's and exponential model is observed. This deviation was experimentally chosen. The smoothing constants for the models were taken as $\alpha = 0.2$ and $\beta = 0.1$ to maintain stability of the forecast. The forecast value obtained from this approach is then used to decide the strategy using forecast by the algorithm for choosing batch strategy by forecasting.

Further for game theory, the utility values for p_1 and p_2 probabilities are calculated for each batch and then by the selection process a $2\,X\,2$ matrix is obtained which undergoes the mixed strategy calculation. The strategy with higher probability is the strategy using game theory.

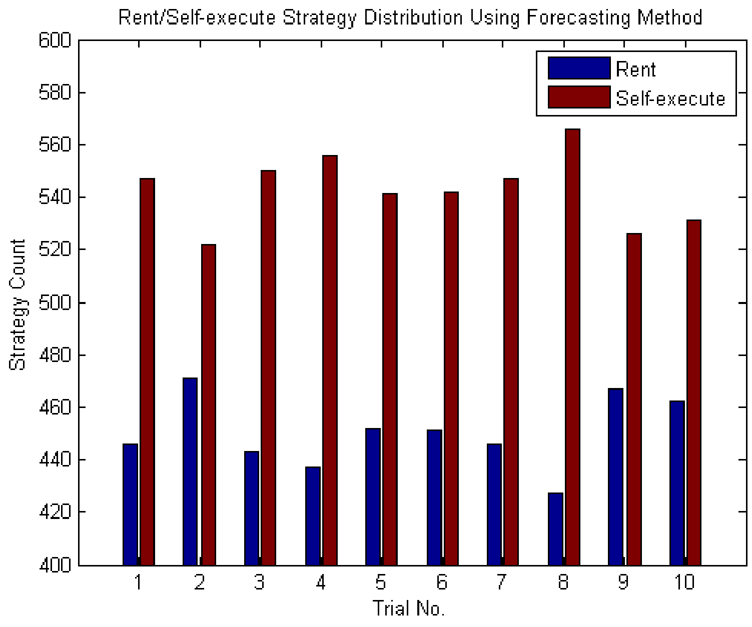
Job/Strategy	Rent	Self-execute
J_i	U_{i1}	U_{i2}
J_j	U_{j1}	U_{j2}

From the above two approaches, two strategies are obtained. Whenever there is a conflict in the strategies obtained from the approaches, the strategy which gives minimum standard deviation with its respective means is finally chosen.

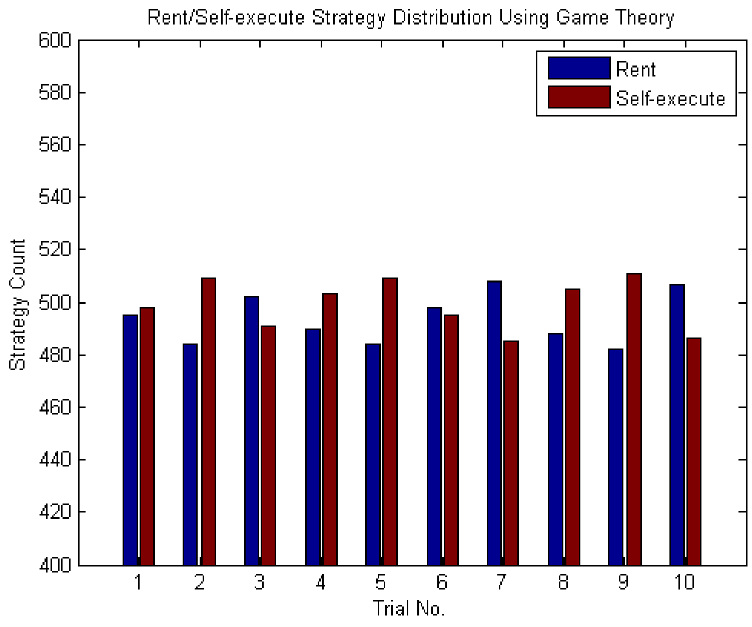
For example, consider that the cloud provider receives the i^{th} batch, then the forecast is done for the $(i+1)^{th}$ batch utility value which is the average of the utility values of all the jobs in the batch. If a trend is observed in these average values of $(i-7)^{th}$ batch to i^{th} batch, then the forecast for the $(i+1)^{th}$ batch utility is obtained using Holt's Method; else exponential smoothing is adopted. Then when the $(i+1)^{th}$ batch is received by the provider, the average actual utility of the $(i+1)^{th}$ batch is computed and is compared with the forecasted value previously obtained. Based on the algorithm for choosing batch strategy by forecasting, either Rent or Self-execute strategy is chosen. At the same $(i+1)^{th}$ batch, two utility values are then calculated for each job in this batch using p_1 and p_2 probabilities and a matrix is constructed. The selection process is then applied to this matrix to obtain the $2\,X\,2$ matrix and finally the mixed strategy calculation is done to decide the Rent or Self-execute strategy using game theory. Thus the strategies for the $(i+1)^{th}$ batch is obtained by the two approaches. Consider a scenario whereby the Rent strategy is chosen through forecasting and Self-execute through game theory (or vice versa) for the $(i+1)^{th}$ batch, standard deviation is then applied and by the last algorithm on conflict of strategies, the final strategy to be adopted by the cloud provider is chosen for the $(i+1)^{th}$ batch.

8.3 Results and Analysis

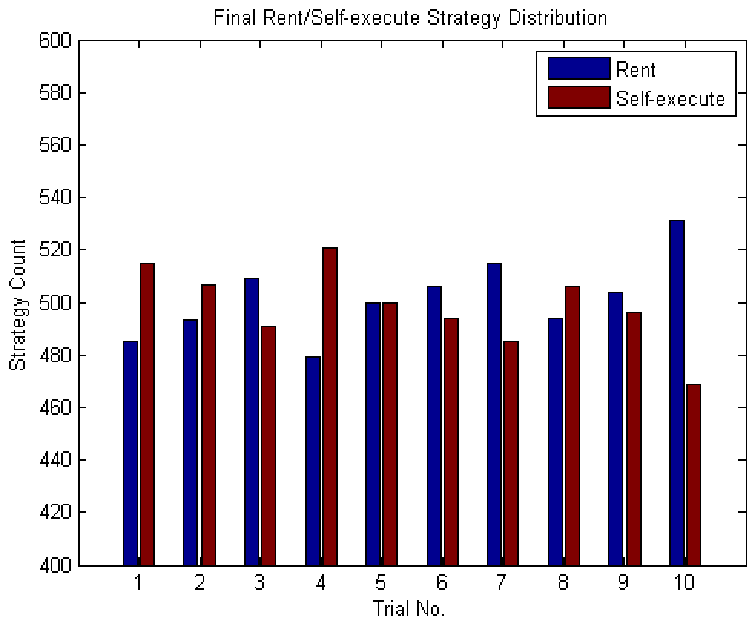
During forecasting, the strategy adopted is seen to be biased towards the Self-execute strategy. The utility values are observed to lie within the range 1 to 10 for the values generated. So when a sudden increasing trend in the utility values is seen (observed as a greater slope), then the next forecasted value will be much higher as it takes into consideration the difference between the values while forecasting in a linear trend. This will result in the Self-execute strategy being chosen frequently. The decreasing trend phenomenon does not happen frequently as this sudden decrease cannot be seen from a high value because of the common range observed. Hence however big a decreasing trend is observed, it still lies within the acceptable range.



An almost equal distribution of strategies of Rent and Self-execute was observed with the game theoretic approach with no strategy overpowering the other. This is because of the randomness in allocating the utility values to the strategies while constructing the matrix. This results in an unbiased evaluation of the matrix which gives an almost equal weightage to both the strategies.



The overall strategy also shows an almost equal distribution after resolving the conflicts. In case of conflicts, the final strategy obtained from the standard deviation method considers both forecasting and game theoretic approaches. In this case, the utility values of the jobs in the current batch is compared with the utility values obtained from forecasting and game theory. This method finally selects the strategy that does not deviate much from the expected value obtained from the above two approaches, thus making it easier for the cloud provider to provide the services required to execute the batch without deviating much from the services he already provides.



The results obtained above are with respect to the randomly generated data inputs. In the real environment, certain values like waiting probability depends on factors like network congestion, efficiency of the cloud, datacenter locations, etc. So during run-time, the results obtained may vary with respect to dynamic factors. The utility function can also be further enhanced to include economic factors for the cloud provider to get a clearer picture.

Chapter 9

Conclusion

In this work, we have discussed one approach of how the illusion of infinite resources in Cloud Computing can be further optimized without incurring any loss for the cloud provider. QoS can be improved and thus it provides the user with better facilities. The cloud provider can decide whether he wants to pay rent for the load and prevent disappointing the user or just execute only what is possible. Forecasting followed by game theory gives a better approach of including the data available till the current point and will help in correctly deciding the strategy. From these results, a cloud provider can also check the number of times he is paying rent for the load and based on further calculations he can decide to own more facilities such that the frequency of renting will decrease and he may earn more profit. This can be modelled as a Ski-Rental problem and further worked out.

This work provides one method of tackling the problem and it further opens up interesting avenues for improvement. The utility property can be changed to suit the agreement between the provider and the user. This approach considers execution time as one of the factors and this may change depending on the network traffic and other dynamic factors which will further complicate the utility property. There can be different utility functions that can be used considering more parameters. Other game-theoretic concepts and forecasting models can also be used to obtain the results. Since the avenues are deep and the domain is still growing, updates on this problem will keep on increasing until an optimal solution is reached.

References

[1] Mell, Peter, and Timothy Grance. "The NIST definition of cloud computing (draft)." *NIST special publication* 800 (2011): 145.

[2] Randles, Martin, David Lamb, and A. Taleb-Bendiab. "A comparative study into distributed load balancing algorithms for cloud computing." *In Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, pp. 551-556. IEEE, 2010.

[3] Mishra, Ratan, and Anant Jaiswal. "Ant colony Optimization: A Solution of Load balancing in Cloud." *International Journal of Web & Semantic Technology (IJWesT) Vol* 3 (2012).

[4] Randles, Martin, Enas Odat, David Lamb, Osama Abu-Rahmeh, and A. Taleb-Bendiab. "A Comparative Experiment in Distributed Load Balancing." *In Developments in eSystems Engineering (DESE), 2009 Second International Conference on*, pp. 258-265. IEEE, 2009.

[5] Manakattu, Sheeja S., and S. D. Kumar. "An improved biased random sampling algorithm for load balancing in cloud based systems." *In Proceedings of the International Conference on Advances in Computing, Communications and Informatics*, pp. 459-462. ACM, 2012.

[6] Armstrong, Django, and Karim Djemame. "Towards Quality of Service in the Cloud." In *Proc. of the 25th UK Performance Engineering Workshop.* 2009.

[7] Endo, Patricia Takako, Andre Vitor de Almeida Palhares, Nadilma Nunes Pereira, Glauco Estacio Goncalves, Djamel Sadok, Judith Kelner, Bob Melander, and J. Mangs. "Resource allocation for distributed cloud: concepts and research challenges." *Network, IEEE* 25, no. 4 (2011): 42-46.

[8] Armbrust, Michael and Fox, Armando and Griffith, Rean and Joseph, Anthony D. and Katz, Randy H. and Konwinski, Andrew and Lee, Gunho and Patterson, David A. and Rabkin, Ariel and Stoica, Ion

and Zaharia, Matei. “Above the Clouds: A Berkeley View of Cloud Computing.” *Technical report*, no. UCB/EECS-2009-28 (2009).

[9] Vinothina Sr, V. “A Survey on Resource Allocation Strategies in Cloud Computing.” *International Journal* (2012).

[10] Goncalves, Paulo, R. O. Y. Shubhabrata, Thomas Begin, and Patrick Loiseau. “Dynamic Resource Management in Clouds: A Probabilistic Approach.” *IEICE Transactions on Communications* 95, no. 8 (2012): 2522-2529.

[11] Han, Zhu, Dusit Niyato, Walid Saad, Tamer Baar, and Are Hjrungnes. *Game theory in wireless and communication networks: theory, models, and applications.* Cambridge University Press, 2011.

[12] Kalekar, Prajakta S. “Time series forecasting using Holt-Winters exponential smoothing.” *Kanwal Rekhi School of Information Technology* (2004).

[13] Ski Rental problem, `http://en.wikipedia.org/wiki/Ski_rental_problem`.

[14] Standard Deviation, `http://en.wikipedia.org/wiki/Standard_deviation`.

[15] Forecasting Method, `http://nptel.iitm.ac.in/courses/110106045`.

[16] SAS Products and Solution Documentation.