A report on

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

Monsoon Semester

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 Monsoon-2012 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

Date

Vinod Pathari
(Asst. Professor)

Abstract

Cloud computing is the use of computing resources (hardware and software) that are delivered as a service over a network. Load balancing is a computer networking methodology to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to optimize resource utilization, maximize throughput, minimize response time, and avoid overload. With Clouds becoming one of the most important concept in the field of internet and resource sharing, it is of utmost importance to optimally balance the tasks and loads in the Cloud. The objective of this work is to study various load balancing algorithms used in Clouds and to design a new algorithm for tackling this issue.

Contents

Chapter 1

Introduction

The Cloud Computing 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”. 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).

1.1 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.
- Comparison of this implementation with a game-theoretic approach to load balancing

1.2 Report Organization

At the outset, the report contains a brief review on the existing works done in the field of Cloud computing focussing on the area of load balancing. A detailed study on two of the popular load balancing algorithms and the design of a new algorithm which is a combination of the above follows. A comparative analysis of the implenmentation of these algorithms has been explained. The report concludes with a plan on the future action to be done in the following semester.

Chapter 2

Literature Survey

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].

Service Models

- **Cloud Software as a Service (SaaS):** The capability provided to the consumer is to use the providers applications running on a Cloud infrastructure.
- **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.
- **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.

Load Balancing is a method to distribute workload across one or more servers, network interfaces, hard drives, or other computing resources. 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 [3].

Goals Of Load Balancing

- To improve the performance substantially.
- To have a backup plan in case the system fails even partially.
- To maintain the system stability.
- To accommodate future modification in the system.

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 behaviour 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 3

Work Done

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

- Biased Random Sampling
- Honey Bee Foraging
- Hybrid Algorithm

3.1 Theory

3.1.1 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 :

- 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
```

3.1.2 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)
```

3.1.3 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.

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
```

3.2 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.

3.2.1 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 neighbour. 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.

3.2.2 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.

3.2.3 Proposed Hybrid Algorithm

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

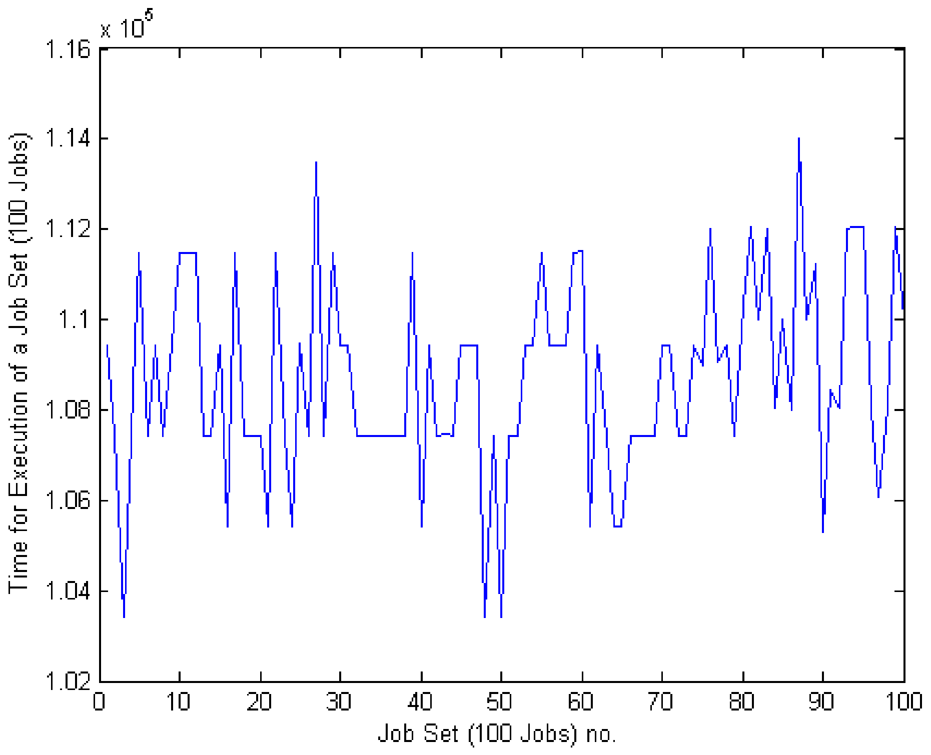
- **Hybrid:** The creategraph() contains the server network (represented by the adjacency matrix) and also about the resources available with each server. Objects of BiasedRandomWalk and Fitness classes are created. The job() receives each job and then checks if any resources are available in the network 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.

3.3 Analysis

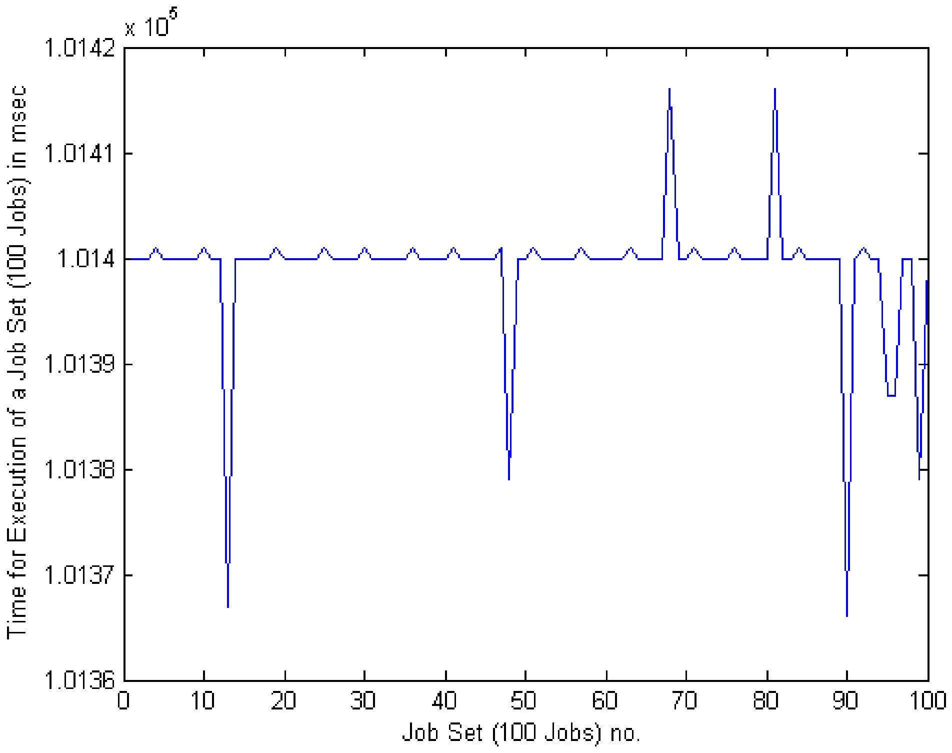
3.3.1 Biased Random Sampling

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.



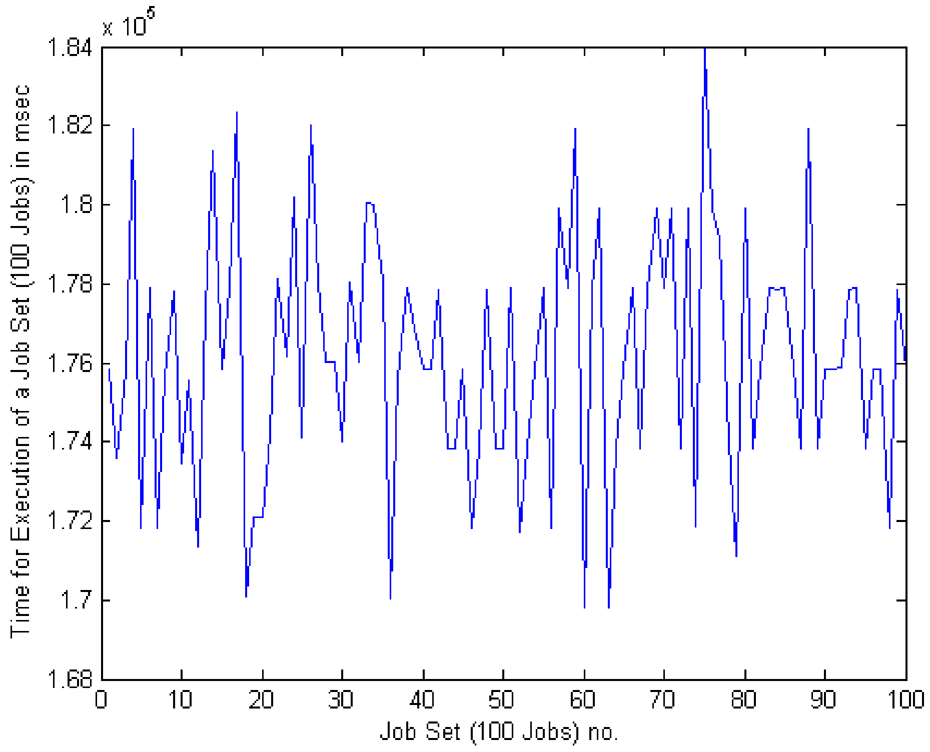
3.3.2 Honeybee Foraging

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.



3.3.3 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. 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 17617ms 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 deadlock 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 deadlock state more due to the contradiction in server selection between the two algorithms.

Chapter 4

Future Work

A complete study and implementation of the algorithms has been carried out over the past three months. In the following semester, the area of load balancing in Cloud computing will be approached from a whole new perspective of game theory. 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	A	B	C
A	5	2	5
B	9	1	3
C	4	1	8

Such kind of representation has motivated us to give a game-theorectic approach to this problem.

Cloud computing, being a new concept, has a lot of scope for research. So, one of the ways to tackle load balancing in Cloud computing is through a game theoretic approach with proper strategies, payoffs and payoff matrices. 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.

References

[1] Peter Mell, Timothy Grance; The NIST Definition of Cloud Computing (Draft); Recommendations of the National Institute of Standards and Technology, `http://http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`

[2] Martin Randles, David Lamb, A. Taleb-Bendiab; School of Computing and Mathematical Sciences, John Moores University, UK; A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing, `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5480636`

[3] Ratan Mishra and Anant Jaiswal; International Journal of Web & Semantic Technology (IJWesT) Vol.3, No.2, April 2012; Ant Colony Optimization: A Solution of Load Balancing in Cloud, `http://airccse.org/journal/ijwest/papers/3212ijwest03.pdf`

[4] Martin Randles, David Lamb, A. Taleb-Bendiab; School of Computing and Mathematical Sciences, John Moores University, UK; A Comparative Experiment in Distributed Load Balancing, `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5395118`

[5] Sheeja S. Manakattu, S. D. Madhu Kumar; ICACCI '12 Proceedings of the International Conference on Advances in Computing, Communications and Informatics; An improved biased random sampling algorithm for load balancing in cloud based systems, `http://dl.acm.org/citation.cfm?id=2345472`