نموذج رقم (1)

إقـــــرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

Efficient Adaptive Load Balancing Algorithm for Cloud Computing Under Bursty Workloads

أقر بأن ما اشتملت عليه هذه الرسالة إنما هو نتاج جهدي الخاص، باستثناء ما تمت الإشارة إليه حيثما ورد، وإن هذه الرسالة ككل أو أي جزء منها لم يقدم من قبل لنيل درجة أو لقب علمي أو بحثي لدى أي مؤسسة تعليمية أو بحثية أخرى.

DECLARATION

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification

Student's name: Sally Fouad Issawi اسم الطالب: سالي فؤاد العيسوي

Signature: التوقيع:

Date: 25. 4. 2015 التاريخ: 25. 4. 2015

بسم الله الرحمن الرحيم

Islamic University – Gaza
Deanery of Post Graduate Studies
Faculty of Information Technology

الجامعة الإسلامية – غزة
عمادة الدراسات العليا
كلية تكنولوجيا المعلومات

Efficient Adaptive Load Balancing Algorithm for Cloud Computing Under Bursty Workloads

By

Sally Fouad Issawi

Supervised By:

Dr. Alaa Al Halees

A Thesis Submitted as Partial Fulfillment of the Requirements for the Degree of Master in Information Technology

1436 H – March 2015

بسم الله الرحمن الرحيم

الجامعة الإسلامية – غزة
The Islamic University - Gaza

مكتب نائب الرئيس للبحث العلمي والدراسات العليا هاتف داخلي: 1150

الرقم ج.س.غ/35/..... Ref

التاريخ 2015/04/12م Date

نتيجة الحكم على أطروحة ماجستير

بناءً على موافقة شئون البحث العلمي والدراسات العليا بالجامعة الإسلامية بغزة على تشكيل لجنة الحكم على أطروحة الباحثة/ **سالي فؤاد عبدالحليم العيسوي** لنيل درجة الماجستير في **كلية *تكنولوجيا المعلومات*** **برنامج تكنولوجيا المعلومات** وموضوعها:

إنشاء خوارزمية متكيفة لتوزيع الأحمال للحوسبة السحابية تحت عبء العمل المتقطع

Efficient Adaptive Load Balancing Algorithm for Cloud Computing Under Bursty Workloads

وبعد المناقشة العلنية التي تمت اليوم الأحد 23 جمادى الآخر 1436هـ، الموافق 2015/04/12م الساعة العاشرة والنصف صباحاً بمبنى اللحيدان، اجتمعت لجنة الحكم على الأطروحة والمكونة من:

أ.د. علاء مصطفى الهليس	مشرفاً و رئيساً	.....
د. إياد محمد الأغا	مناقشاً داخلياً	.....
د. سناء وفا الصايغ	مناقشاً خارجياً	.....

وبعد المداولة أوصت اللجنة بمنح الباحثة درجة الماجستير في **كلية *تكنولوجيا المعلومات*/ برنامج تكنولوجيا المعلومات.**

واللجنة إذ تمنحها هذه الدرجة فإنها توصيها بتقوى الله ولزوم طاعته وأن تسخر علمها في خدمة دينها ووطنها.

والله ولي التوفيق،،،

مساعد نائب الرئيس للبحث العلمي والدراسات العليا

أ.د. فؤاد علي العاجز

P.O. Box 108, Rimal, Gaza, Palestine fax: +970 (8) 286 0800 فاكس Tel: +970 (8) 286 0700 ص.ب. 108 الرمال، غزة، فلسطين هاتف
public@iugaza.edu.ps www.iugaza.edu.ps

Abstract

Cloud computing is a recent emerging technology in IT industry. It is an evolution of previous computing models such as grid computing. It enables a wide range of users to access a large sharing pool of resources over the Internet. In such complex system, there is a tremendous need for efficient load balancing scheme in order to satisfy peak user demands and provide high quality of services. Load balancing is a methodology to distribute workload across multiple nodes over the network links to achieve optimal utilizing of resources, minimizing data processing time and response time, and avoid overload. One of the challenging problems that affect the load balancing process is bursty workload. Burstiness occurs in workloads in which bursts of requests aggregate together during short periods of time and create periods of peak system utilization. This can lead to dramatically degradation in system performance. Several load balancing algorithms had been proposed which focus on key elements such as processing time, response time and processing costs. However these algorithms neglect cases of bursty workload. In the same time, research works which deals with the problem of bursty workload are quite a few. Motivated by this problem, this research comes to handle the load balancing problem in cloud computing under bursty workload by predicting the variation in the request rate and apply the suitable load balancing algorithm according to the predicted load status. In turn, the selected load balancing algorithm assign the received request to a virtual machine based on information supplied by a fuzzifier. The proposed algorithm has been tested and the experiments results showed that our algorithm improves the cloud system performance by decreasing the response and the data center processing time compared with other algorithms. The decrement is about 2ms when the instruction length is 250 Byte, while this improvement becomes more obvious by decreasing the response time about 10ms and the processing time about 5ms when the instruction length is increased to 1000 Byte.

Keywords: *Cloud Computing, Load Balancing Algorithm, Burstness, Fuzzifier, CloudAnalyst, Response Time, Processing Time.*

Acknowledgement

First, I thank Allah *for guiding me and taking care of me all the time. My life is so blessed because of his majesty.*

I would like to thank ***My Family*** *especially* ***My Parents*** *for encouraging and supporting my all the time.*

Also, I would like to take this opportunity to thank all ***My Teachers*** *and My research supervisor,* ***Dr. Alaa Al-Halees*** *for giving me the opportunity to work with him and guiding and helping me throughout this research and other courses.*

Very special thanks to My Teacher ***Dr. Mohammed Radi*** *for guiding and supporting me by his experience and valuable advices to accomplish this research.*

I wish to express my considerable gratitude to my special ***friend and colleges,*** *for there supports.*

Thank you all for being always there when I need you most. Thank you for believing in me and supporting me.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DC	Data Center
ESCE	Equal Spread Current Execution
FIS	Fuzzy Inference System
GAE	Google App Engine
IaaS	Infrastructure as a Service
IT	Information Technology
ms	Mille Second
PaaS	Platform as a Service
QoS	Quality of Service
RR	Round Robin
SaaS	Software as a Service
SLA	Service Level Agreement
TLB	Throttled Load Balance Algorithm
UB	User Base
VM	Virtual Machine

Chapter 1

Introduction

Chapter 1 Introduction

Appling load balancing algorithm which can work efficiently under different workload cases (high, low, burst… etc.) is an important issue in cloud system to insure delivering high quality of service for cloud users. This research came to handle this important issue. In this chapter we give a brief overview about the area of this research. Then we highlight the problem statement and the objectives of this research. Also we talk about the scope and the limitation of this work. After that we will present the research methodology which had been followed in order to accomplish this research. Finally we will present the organization of the thesis.

1.1 Overview

Nowadays most development in IT industry comes to meet the demand for utilizing more resources in lower costs. This technological trend has enabled the evolution of a new computing model called cloud computing, in which resources and services are available on Internet and can be leased and released on demand fashion [1]. According to The National Institute of Standards and Technology's (NIST) [2] cloud computing is defined as "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."

The core idea behind cloud computing is not a new one. Cloud computing is an evolution of previous computing models starts from grid computing, utility computing, and finally software as a service. It brings together a set of existing technologies such as virtualization and utility-based pricing, and leverages these existing technologies to meet the technological demand for solving variety of technological problems and issues [1].

The main concern for cloud computing users is to get high quality of service with low costs. However there are many issues that affect the quality of the provided service such as load balancing, performance, security, and fault tolerance. In this research the main concern is load balancing in cloud computing.

Load Balancing is a method to distribute workload across one or more servers, Virtual Machine (VM), network interfaces, hard drives, or other computing resources. It is used to make sure that none of your existing resources are idle while others are being over utilized [3]. One of the most challenging problems that dramatically degrade the performance of load balancing process is burstiness in workload.

Bursty traffic refers to an uneven pattern of data transmission: sometime very high data transmission rate while other time it might be very low [4]. Burstiness occurs in workloads in which bursts of requests aggregate together during short periods of time and create periods of peak system utilization. Figure 1.1 presents bursty wokload.

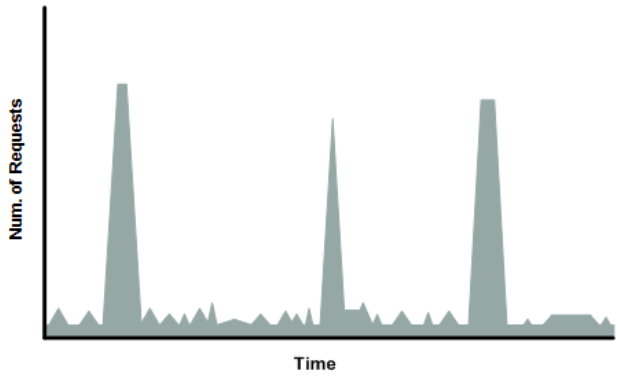


Figure 1.1 Bursty Workload

In this work, we present a new load balancing algorithm for cloud computing environment which can overcome the performance degradation results from bursty workload. The proposed algorithm mainly depends on detecting the start and the end of the burst interval so that it can apply the suitable load balancing algorithm according to the status of the load. Besides that, a fuzzifier is used to decide the degree of balance for every Virtual Machine (VM) based on CPU speed and the assigned load so that this information can enhance the process of assigning the load to the suitable VM.

1.2 Statement of the Problem

Complex systems such as cloud systems where number of access users increased, suffer from burstness in workload. Bursty workload degrades the efficiency of the load balancing algorithm and thus adversely affects the performance of the cloud computing system. However, this problem is not well covered by researchers.

1.3 Objectives

1.3.1 Main objective

Our main objective in this research is to propose an adaptive load balancing algorithm that works efficiently under both normal and bursty workloads.

1.3.2 Specific objectives

The specific objectives of our research are:

- Find the best configuration that generates a bursty workload.
- Evaluate the traditional scheduling algorithms under bursty workload.
- Find the best load balancing algorithm that can work efficiently in normal load.
- Find the best load balancing algorithm that can work efficiently in high load.
- Identify the start point and the end point of burst.
- Develop a suitable algorithm that can swap between the load balancing algorithms.
- Build an agent using fuzzy logic which is responsible for gathering information about VMs in order to help the load balancing algorithm in selecting a suitable VM to handle the received request.

- Implement the proposed algorithm.
- Evaluate the proposed algorithm using two metrics: Response Time and Processing Time.
- Compare the proposed approach with other approaches to make evaluation.

1.4 Significance of the thesis

The significant of this research comes from the following main points:

1. This research is considered as one of the few researches done on load balancing under bursty workload.
2. Adapting the used load balancing process according to the status of the workload.
3. Dealing with bursty workload case which is an important problem that causes degradation in the performance of the load balancing mechanism.
4. The proposed algorithm can works well under bursty and normal workload.
5. Enhance the performance of the cloud system under bursty workload.

1.5 Scope and limitations of the project

This research aims to propose an efficient adaptive load balancing algorithm which can respond to the change in the workload. The work is applied with some limitations and assumption such as:

- The main concern in this research is to deal with bursty workload problem.
- The research will focus on resource utilization and response time while other issues such as reducing cost and fault tolerant is not considered.
- The proposed algorithm is a dynamic, non-distributed load balancing algorithm.
- The performance of the proposed algorithm will be measured using CloudAnalyst simulator.
- The proposed algorithm will be compared with Round Robin, ESCE, and Random algorithm while Throttled algorithm will be excluded.

1.6 Research Methodology

The methodology of research which had been followed in order to complete this research and achieve our goal presented in Figure 1.2:

Figure 1.2 Research Methodology Steps

1) **Literature Review**
   Firstly, a review had been done for current techniques used for load balancing in cloud computing environment and how they address the burstiness problem.
2) **Study burstiness**
   Hard study for burstness and its cases and consequences on the performance of distributed systems in general and on cloud systems in specific had been done.
3) **Understand CloudAnalyst Simulator**
   CloudAnalyst had been used to simulate the cloud environment in this work. Thus, its architecture and classes had been studied in order to know how to make our configuration and add the classes of our proposed algorithm so we can test and evaluate it.
4) **Test different load balancing algorithms on CloudAnalyst.**
   Different load balancing algorithms had been tested on CloudAnalyst simulator to study its performance under burst and non-burst cases.
5) **Develop our proposed solution**
   The proposed algorithm for efficiently distributed workload and overcome burstiness problem had been developed.
6) **Implement the proposed algorithm**
   The proposed algorithm had been implemented and tested on CloudAnalyst.

7) **Evaluate and comparing results**

The evaluation of the system had been done using two metrics: response time and processing time. These two metrics had been used to compare the proposed algorithm with three popular algorithms which are: Round Robin, ESCE, and Random.

1.7 Thesis Overview

The rest of the thesis is organized as follow: Chapter two includes a literature review of the load balancing in cloud computing and the burst problem; and a survey on the current approaches which addressed this problem. Chapter three defines in detail the proposed approach including the tools and the mechanisms used in developing this approach. Chapter four includes experiments and results. Finally chapter five presents the conclusions and future works.

Chapter 2

Literature Review and Related Works

Chapter 2 Literature Review and Related Work

"Best performance in lower cost", is a remarkable principle in IT industry nowadays which leads the evolution of computing model to Cloud Computing. However, despite the fact that Cloud computing experiencing a rapid advancement both in academia and industry, the development of cloud computing technology is currently at its infancy, with many issues still to be addressed such as load balancing. In this chapter we will define cloud computing model and present its types and provisioning services levels. In addition we will talk about load balancing issues in cloud computing and specifically we will focus on burst workload problem and how it degrades the efficiency of the load balancing algorithm and affect the performance of the cloud system. Also we will discuss some related work done in this research area.

2.1 Cloud Computing

The fundamental idea of cloud computing was pronounced way back in 1960 by Professor John McCarthy, as; "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility. The computer utility could become the basis of a new and important industry" [5].

Cloud computing is a type of parallel and distributed system. It consists of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources. The services delivered to the consumer are based on service-level agreements (SLA) which established through negotiation between the service provider and consumers [6].The objective of the cloud computing is to provide secure, qualitative, scalable, quick, more responsive, on demand, cost-efficient and automatically provisioned services just like: computation services, storage services, networking etc. Although those services are geographically distributed all over the world, they are provided in a transparent way (location independent) [5].

Cloud computing can help to improve business performance by making a contribution to control the cost of delivering IT resources to any organization. It minimizes the overhead of buying, managing and controlling IT resources. The financial model applied in cloud computing is "Pay-per-Use" so the consumer only pay for his needs.

2.1.1 Cloud Service Model

The principle of cloud computing is to deliver different computing services on Internet which are available as subscription-based services in a pay-per-use model to consumers. These services are essentially categorized under three classes as hierarchy as can be seen in Figure 2.1.

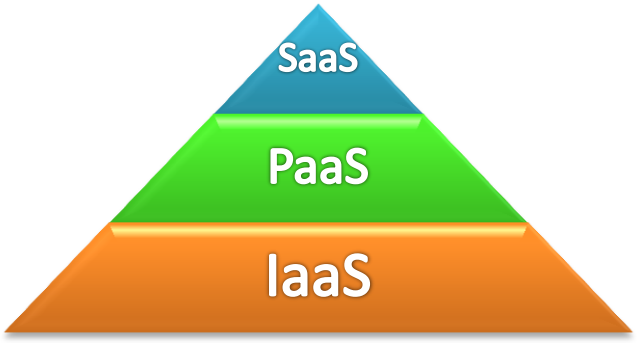


Figure 2.1 Services Provided by Cloud Computing

1. **Software as a Service (SaaS):**
   In this highest level, different types of applications running on cloud environment are provided to the customer. The user can access those applications from various devices through a thin client interface such as a web browser[7]. For example, Gmail is a SaaS where Google is the provider and we are consumers [8].
2. **Platform as a service (PaaS):**
   This intermediate level provide a platform for developers to deploy there applications which are built using programming languages and tools supported by the provider[7] such as: Google App Engine (GAE), Microsoft Azure, IBM Smart Cloud, Amazon EC2, and salesforce.com [8].
3. **Infrastructure as a Service (IaaS):**
   The last level provides a fundamental computing resources such as processors, storage, and resources [7]. For example: Storage services provided by AmazonS3, and Amazon EBS; and Computation services provided by AmazonEC2, and Layered tech [8].

2.1.2 Cloud Deployment Model

Cloud systems can be deployed in four forms which are: private, public, community and hybrid cloud depending on access allowed to the users as shown in Figure 2.2. They are classified as follows:

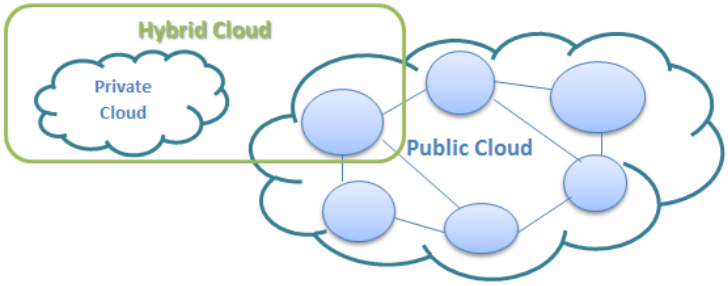


Figure 2.2 Types of Cloud Computing Model

1. **Public Cloud:**
   This model allows cloud environment as openly or publically accessible. It is available from a third party service provider through the Internet. This deployment model is implemented for general users and it is managed and controlled by an organization selling cloud services. It is very cost effective for small and midsize business (SMBs) to deploy IT solutions [8] [9] [5].

2. **Private Cloud:**
   This deployment model is built specifically to provide the services within an organization and is exclusively used by their employees at organizational level. It is managed within an organization and this makes it more secure than the public one. This deployment model is suitable for large enterprises [8] [9] [5].
3. **Hybrid Cloud:**
   Hybrid cloud is an amalgamation of private and public cloud. The participating clouds interact together by some standard protocols. In this deployment model, the organization can deploy some services in their own private cloud while other services can be provided by a public cloud [8] [5].
4. **Community Cloud:**
   This cloud model is shared and managed by number of related organizations with shared concerns such as security requirements, mission, and policy considerations [8] [5].

2.1.3 Cloud Components

Cloud computing solution is made up of three main elements as shown in Figure 2.3: clients, data centers, and distributed servers. Each one of these elements plays a specific role in service provisioning process [10].

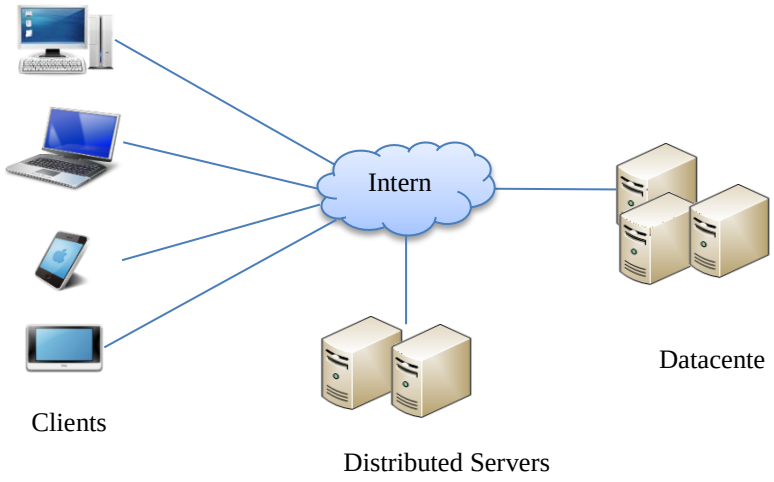


Figure 2.3 Three Components Make Up Cloud Solution [10]

1. **Clients**
   Clients are the devices that the end users interact with to manage their information on the cloud. They might be computers, laptops, tablet computers, mobile phones, or PDAs [10].
2. **Datacenter**
   The datacenter is the collection of servers where the application to which you subscribe is housed [10]. In order to minimize the number of physical servers needed to host tremendous numbers of applications, virtualization technology is used. This technology allows one physical server to contain a large number of virtual servers. In this way, power and costs of running dozens of servers can be minimized [10].

3. **Distributed servers**
   Those servers are distributed in geographically disparate locations but they act together to appear to the cloud subscriber as they are all next to each other [10].

2.1.4 Virtualization

Cloud paradigm offers remarkable advantages through reduction operation costs, decreasing power consumption, and server consolidation. One of the most important technologies that enabled these features is virtualization, and more particularly machine virtualization [11].

Machine virtualization (also known as processor virtualization) allows a single physical machine to emulate the behavior of multiple machines, with the possibility to host multiple and heterogeneous operating systems (called guest operating systems or guest OSs) on the same hardware. This means that several logical servers run on one physical machine so costs of deployment are reduced [11] [12].

There are three main characteristics make virtualization technology ideal for cloud computing [13]:

1. **Partitioning:** many virtual machines are supported in a single physical machine by partitioning the available resources.
2. **Isolation:** every virtual machine is isolated from other virtual machines so if one virtual machine crashes, it doesn't affect the other virtual machines. This prevents the cloud system from being down.
3. **Encapsulation:** cloud system provides different applications and services. Encapsulation can protect each application so that it doesn't interfere with another application.

2.1.5 Issues and Challenges of Cloud Computing

The research on cloud computing is still at the early stage. Several new challenges keep emerging from industry applications, and many issues need to be properly addressed. Some of the challenging research issues in cloud computing is as follows:

1. **Security and Privacy**
   One of the more obvious cloud concerns is how to address security and privacy issues while accessing data and applications by different users. Organizations have critical concerns about protecting its data from being vulnerable to different attacks as the data is stored, processed and moved outside the control of the organization [5].

2. **Performance**
   According to the survey of International Data Corporation (IDC), performance is the second biggest issue in cloud adoption after security. Performance is a competitive feature for cloud provider as cloud must provide improved performance when a user moves to cloud infrastructure. Several issues may degrade the cloud performance such as disk space, limited bandwidth, lower CPU speed, and memory and network connections [5].

3. **Resource Management and scheduling**

   Resource management is a very important issue in cloud computing, as large numbers of users shared the same resources. So in order to meet Quality of Service (QoS) standards and insure best resource utilization, proper resource management mechanisms should be used in deferent levels, such as management of memory, disk space, CPU's, cores, threads, VM images, I/O devices etc. Resource provisioning can be defined as allocation and management of resources to provide desired level of services. Job scheduling is an essential process in resource provisioning where the order of the job execution is established in order to optimize performance parameters such as response time, processing time, waiting time … etc. [5]. One of the most important issues in job scheduling is load balancing which is our interest in this work.

2.2 Load Balancing

The scale up in demands make load balancing a major concern in cloud computing. It is defined as 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 your existing resources are idle while others are being utilized [3]. Some of the main goals of load balancing as pointed out by [14] [6] are:

1. Achieve overall improvement in the system performance.
2. Decrease the response time.
3. Increase the system throughput.
4. Decrease overhead on the cloud system components.
5. Maintain system stability in emergency situations such as sudden surge of arrivals.
6. Optimize resource utilization.

2.2.1 Mathematical Definition:

The mathematical model of load balancing is defined as follows:
Let say that there are n set of Load or requests need to be scheduled given as[15, 16]:

$$L = \{L_1, L_2, \dots\dots\dots\dots\dots, L_n\} \qquad (1)$$

And there are K set of Virtual Machines in a Datacenter given as:

$$V = \{V_1, V_2, \dots\dots\dots\dots\dots, V_k\} \qquad (2)$$

The current Datacenter load is the set of load for all virtual machines in this datacenter given as:

$$DL = \{VL_1, VL_2, \dots\dots\dots\dots, VL_k\} \qquad (3)$$

We need to find a function $f(L)$, which the set of load L can be mapped to the set of Virtual Machines V, making the Load VL_i of each Virtual Machine V_i be essentially equal, that is:

$$VL_1 \approx VL_2 \approx \cdots \approx VL_k \qquad (4)$$

Let us use τ_o to reflect the time needed for executing task L_o on the Virtual Machine V_i, so the time needed for executing all the tasks on the Virtual Machine V_i is as follows:

$$t_i = \sum_{0 \in f(L_i)(i=1,\dots,n)} \tau_o \qquad (5)$$

When $k=1$, that means there is only one Virtual Machine, and all the tasks should be executed serially on this Virtual Machine, so the time needed is the sum of all the time, which can be represented as T_1 shown below:

$$T_1 = \sum \tau_o \ (o = 1, \dots, n) \qquad (6)$$

When $K > 1$, that means there is more than Virtual Machine, and the tasks can be shared to multiple server nodes for dealing with in parallel, the time needed is represented as T_k shown below:

$$T_k = max_{i=1,\dots,n} \ t_i \qquad (7)$$

Thus, the goal of the load balancing is to solve the mapping $f(L)$ to get the minimum of T_k in case of $VL_1 \approx VL_2 \approx \cdots \approx VL_k$

2.2.2 Categories of Load Balancing Algorithms

Basically there are 2 types of load balancing algorithm depending on their implementation method:

1) **Static Algorithms**
   In this type, the load is divided equivalently between nodes. This algorithm depend on Prior knowledge of the system it does not consider the current state of the node and this will degrade the performance of the system. This type of algorithms is announced as round robin algorithm [3, 17].
2) **Dynamic Algorithms**
   Dynamic algorithms make decisions based on current state of the system. No prior knowledge is needed[3]. So workloads can be distributed efficiently over nodes. Dynamic load balancing can be done in two ways: [18]

   - **Distributed dynamic load balancing:**
     In the distributed one, all nodes in the system execute the dynamic load balancing algorithm and the task of load balancing is shared among them. The advantage of this way is that if one or more nodes in the system fail, the system performance will be affected to some extent, but it will not cause the total load balancing process to halt.

   - **Non-distributed dynamic load balancing:**
     In the non-distributed one, the load balancing algorithm is executed by a single node of the system and the task of load balancing is dependent only on that node. Unlike distributed algorithm, the failure in one node will cause the total load balancing process to halt.

In comparison between static and dynamic algorithms, static (Round Robin) algorithms are based on simple rule in dividing the loads among nodes but this leads to more loads conceived on servers and thus imbalanced traffic discovered as a result. However; dynamic algorithm predicated on query that

can be made frequently on servers, but sometimes prevailed traffic will prevent these queries to be answered, and correspondingly more added overhead can be distinguished on network [17].

2.2.3 Popular Load Balancing Algorithms for Cloud Computing

1. **Round Robin**

   Round Robin is one of the simplest scheduling techniques. It is a static algorithm, which distributed the load equally to all the nodes[18]. Using this algorithm, the allocation of VMs to nodes is done in a cyclic manner. The scheduler starts with a node and moves on to the next node, after a VM is assigned to that node. This process is repeated until all the nodes have been allocated at least one VM and then the scheduler returns to the first node again. As an example, if there are three nodes and three VMs are to be scheduled, each node would be allocated one VM, provided all the nodes have enough available resources to run the VMs [19].

   Problem with this kind of algorithm is that these algorithms are not able to handle bursty workloads. Even these algorithms do not consider the current situation of each node of the system [18].

2. **Equal Spread Current Execution (ESCE)**

   It is spread spectrum technique in which the load balancer spread the load of the job in hand into multiple virtual machines [20]. It focuses on preserving equal load to all the virtual machines connected with the data center. In this algorithm [21] the load balancer maintains an index table of VMs as well as number of requests currently assigned to the VM. When the data center receives a request, it scans the index table for the least loaded VM. If there are more than one VM has the least load then the first founded is selected. The load balancer returns the Id of the reserved VM to the data center. Then the load balancer updates the index table by increasing the allocation count of the reserved VM by one. When VM completes the assigned task, the load balancer decreases the allocation count of the VM by one. If there is a VM that is free and there is another VM that needs to be freed of the load, then the balancer distributes some of the tasks of that VM to the free one so as to reduce the overhead of the former VM [20].

3. **Simple Random**

   Random is a very simple algorithm. In this algorithm [22] the received task is assigned randomly to the available VM without any considerations of the status of the VM. Hence, this may result in the selection of a VM under heavy load and the job requires a long waiting time before service is obtained. On the other hand random algorithm has low complexity as it does not need any overhead or pre-processing to select a VM.

4. **Throttled Load Balance Algorithm (TLB)**

   In this algorithm the load balancer maintains an index table of virtual machines as well as their states (Available or Busy) [21]. When the data center receives a task, it makes a request to the load balancer to find a suitable virtual machine (VM) to perform the recommended job. The load balancer scans the index table from top until the first available VM is found, and then the load

balancer accepts the task and allocates that VM. If there is no VM available then the load balancer returns -1 and the task is queued.

In our Adaptive algorithm, based on experiments results, Round Robin will be used in low workload cases while Random will be used when burst occurs. For evaluation process, the performance of the Adaptive algorithm is compared with three algorithms: RR, ESCE and Random while TLB is excluded because when there is no VM available the task is not processed and it is queued in the Data Center. So at the end of simulation time the number of processed tasks is not equal to the Adaptive algorithm and the other three algorithms and this may affect the comparison results.

2.2.4 Evaluating Load Balancing Algorithm

There are many criteria for evaluating the performance of load balancing algorithm [23]:

- Average response time per unit time.
- Average waiting time per unit time.
- Average processing time per unit time.
- Workload(requests) to be serviced per second(Mbps) or a unit of time
- Throughput (Req / Sec), this criterion will be recovered recovery, buffering capacity and processing power factors
- Percentage of CPU utilization
- The number of requests executed per unit time
- The number of requests per unit time buffer
- The number of rejected requests per unit time.

The response time criterion considered as the most important one as it covers all other factors completely [23].

2.2.5 Bursty Workload

One of the most challenging problems that dramatically degrade the performance of load balancing process is burstiness in workload.

Bursty traffic refers to an uneven pattern of data transmission: sometime very high data transmission rate while other time it might be very low [4]. Burstiness occurs in workloads in which bursts of requests aggregate together during short periods of time and create periods of peak system utilization. Figure 2.4 shows three different level of burstiness: strong, week, and no burstiness.

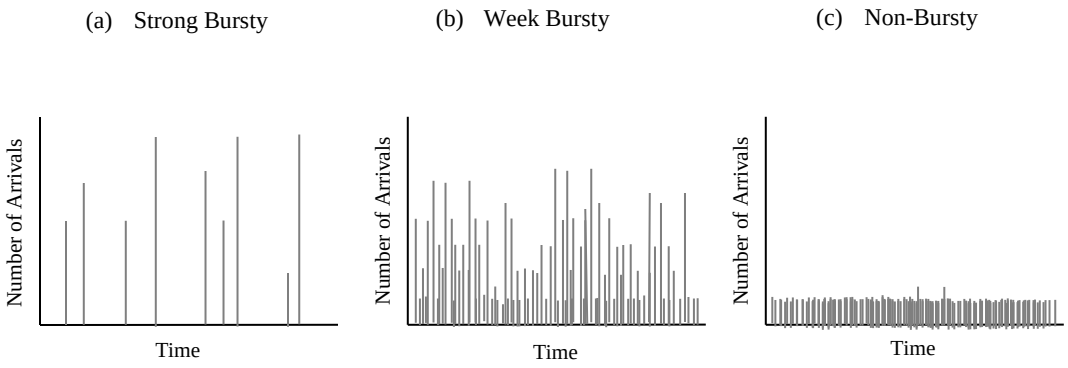


Figure 2.4 Three Different Levels of Burstiness

This problem is often observed in systems including web based applications[24], grid services[25], multitier architectures [26], large storage systems [27]. It can dramatically degrade the system performance, make the system unavailable and lead to a total failure. Burstiness considered the most complex problem nowadays in cloud computing as the number of users that uses cloud services, increases day by day. Therefore load balancer must consider the performance of each instance under both bursty and non-bursty workloads for best performance.

Adaptive algorithm is supposed to deal with different burstiness levels.

2.3 Fuzzy Logic

In our proposed algorithm fuzzy logic is used in order to decide the balancing degree of VM depending on the processor speed and the assigned load on the VM. The logic of computers in making decision is based on “true or false” (0 or 1) Boolean Logic. However, in real life this logic is not practically applicable. People use uncertainty natural language terms such as “Slightly”, ”Somewhat”, “SortOf”, “A Few”, “Mostly”, … etc. to express their opinion and decisions.

In 1960s, Dr. Lotfi Zadeh of the University of California at Berkeley was working on the problem of computer understanding of natural language. In fact, it is not easy to translate natural language terms into absolute terms of 0 and 1 so Dr. Zadeh introduce the concept of partial truth which indicats that Truth values are between “completely true” (1) and “completely false” (0); and this is the base idea of Fuzzy logic [28].

Fuzzy logic is the soft computing approach to compute based on “degrees of truth” rather than the usual “true or false” logic. Lotfi Zadeh says that fuzziness involves possibilities. For instance, for numbers from 1 to 6, it’s possible that 6 is a large number, while it’s impossible that 1 or 2 are large numbers. In this case, a fuzzy set of possible large numbers includes 3, 4, 5, and 6 [28].

2.3.1 Systems Use Fuzzy Logic

Fuzzy logic methodology can be utilized in solving the problems that are complex to be analyzed quantitatively or are not easy to be modeled mathematically such as estimating, decision-making, and mechanical control systems. There are five types of systems where fuzziness is necessary or beneficial [28]:

1. Complex systems which are difficult or impossible to model.
2. Systems controlled by human experts.
3. Systems with complex and continuous inputs and outputs.
4. Systems that use human observation as inputs or as the basis for rules.
5. Systems which are naturally vague, such as those in the behavioral and social sciences.

2.3.2 Fuzzy System Components

A general Fuzzy System mainly involves the following components as shown in Figure 2.5 [29]:

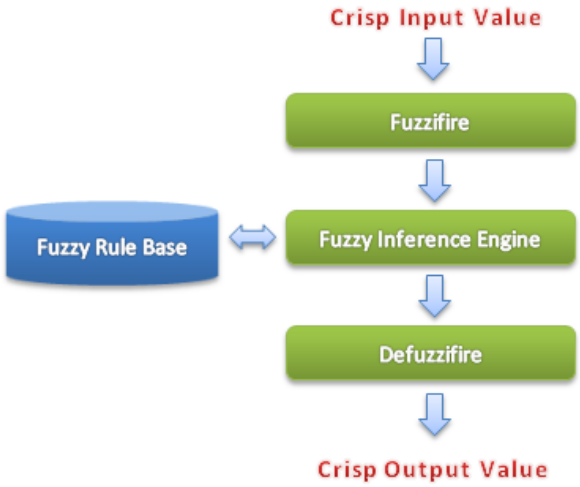


Figure 2.5 The General Architecture of the Fuzzy Logic System [29]

To clarify the concept of fuzzy system and explain the job of each component, we will take air conditioner system as simple example of fuzzy system. The conditioner system has two inputs: current room *temperature* and the *target* temperature; and one output: the result *command.* The linguistic variables for *temperature* input are:

- Cold and its interval [0^o - 10^o]
- Worm and its interval [11^o - 20^o]
- Hot and its interval [21^o - 50^o].

For *target* input the linguistic variables are:

- Cold and its interval [0^o - 10^o],
- Worm and its interval [11^o - 20^o],
- Hot and its interval [21^o - 50^o].

For output, the linguistic variables for *command* are:

- Heat and its interval [25 ^{o} - 45 ^{o}]
- Cool and its interval [3 ^{o} - 20 ^{o}]
- worm and its interval [21 ^{o} - 25 ^{o}]

1. **The Fuzzifier**

   The Fuzzifier is the first station in the fuzzy system. It receives the input value in numerical (Crisp) form, and then it maps the given numerical inputs to fuzzy sets and linguistic variables [29].

   According to our example let's say that the *temperature* = 30^{o}, and the *target* = 10^{o}. So the fuzzifier will find the linguistic variables for the numerical inputs which are: *temperature* = hot, and the *target* = cold.

2. **The Rule Base**

   Includes a set of linguistic rules designed in the form (**If-Then** rules) [29]. For our example the rule base are set in Table 2.1

Table 2.1 Fuzzy Rules for Air Condition System

	Fuzzy Rules
1	**IF** temperature is *hot* and *target* is *cold* **THEN** command is *cool*
2	**IF** temperature is *cold* and *target* is *hot* **THEN** command is *heat*
3	**IF** temperature is *worm* and *target* is *worm* **THEN** command is *worm*

3. **The Fuzzy Inference Engine**

   It is considered the central part of the fuzzy system. In this stage, in order to produce the intended output, the inference engine is applied to a set of rules included in the fuzzy rule base. This procedure involves many steps as follows: first, it matches the linguistic variables of the input with the rules' premises. Second, it activates the matched rules in order to deduce the resultant of each fired rule, and finally it combines all consequents by using fuzzy set union in order to generate the final output which is represented as fuzzy set output [29].

   In our example the inputs match the first rule so it will be fired.

4. **The defuzzifier**

   As described in the previous phase, the output is produced as a linguistic variable, which is fuzzy and can be interpreted in different ways. Therefore, the fuzzy set output in this stage is converted to a crisp output, which is a numerical value [29].

   According to our example, the linguistic for *command* is cool while the final crisp output is 10.

2.4 Related works

Several approaches had been proposed to handle the load balancing issues in cloud computing systems. All these works aimed to improve the process of distributing the workload among cloud nodes and try to achieve optimal resource utilization, minimum data processing time, minimum average response time, and avoid overload. However most of these approaches neglected the effect of burstiness on load balancing process which degrades the performance of the system. This section presenting some researches had been done in this area. The related works is organized under three classes. Firstly, researches which address load balancing in cloud computing. Then researches which deal with burst problem in load balancing in cloud computing. Finally, the researches which deal with burstness problem in deferent distributed systems.

2.4.1 Load Balancing in Cloud Computing Approaches:

Ratan and Anant in [3] proposed an ant colony optimization to initiate the service load distribution under cloud computing architecture. It mimics the use of the pheromone by ants to select the optimal path that leads to their destination. In the same way, selecting a node to process a request is based on a Pheromone Table which contains a probability of each node to be selected to serve the received request. The node with high probability is selected and then the pheromone table is updated by increasing the probability of this node and decreasing other nodes probabilities. The pheromone update mechanism has been proved as an efficient and effective tool to balance the load.

The researches mentioned that their technique did not consider the fault tolerance issues. In addition burstiness load cases are not considered in the proposed algorithm. Neglecting these important issues degrade the efficiency of this algorithm.

Sethi, et al in [30] designed a new load balancing technique using fuzzy logic based on Round Robin (RR) algorithm to obtain measurable improvements in resource utilization and availability of cloud-computing environment. The proposed technique uses a fuzzifier to perform the fuzzification process that converts two types of input which are processor speed and the assigned load of Virtual Machine (VM) and one output which is a balanced load to create an inference system. The Fuzzy based Round Robin (FRR) load balancer compared to conventional Round Robin (RR) load balancer where the experimental results show that FRR minimize the data center processing time and overall response time.

The problem with Round Robin algorithms in general that it is not able to handle bursty workloads. Even with the proposed enhancement on RR by using fuzzy logic burstiness cases are not considered.

Dave and Maheta in [31] proposed new load balancing algorithm based on round robin algorithm, they made a modification on round robin algorithm by implementing a dynamic time Quantum based on algorithm execution round. The result showed an improvement in response time as compared to normal round robin algorithm. The drawback of this paper that authors had focused only on how to decrease the response time and they ignored talk about processing cost. In addition, the researchers compared their results with only RR algorithm which had been enhanced and improved by many researchers before.

Singhal and Jain in [32] proposed a load balance algorithm using Fuzzy Logic, the algorithm focused on a public cloud. The main idea of the algorithms is to partitioning the Cloud to several cloud partition and

each partition having its own load balancer, and there is a main controller which manage all these partition. And with the idle partition status they used a fuzzy logic and in the normal partition status they used a global swarm optimization based load balancing strategy. The result showed enhancements in resource utilization and availability in cloud computing environment. The drawback of this approach is difficulty of testing the technique in a real environment to make sure that it has achieved good results.

2.4.2 Load Balancing Approaches for Burst Workload in Cloud Computing:

Jianzhe T., et al in [33] proposed a smart burstiness-aware algorithm (ARA) to balance bursty workloads across all computing sites, and thus to improve overall system performance. The presented algorithm predicts the beginning and the end of workload bursts and automatically on-the-fly shift between two schemes "greedy" (i.e., always select the best site) which has better response time under the case of no burstiness, and "random" (i.e., randomly select one) which has better response time under burstiness case. Both simulation and real experimental results show that this algorithm improves the performance of the cloud system under both bursty and non-bursty workloads.

Although this algorithm give good results, but it does not consider an important factor in load balancing, which is the current utilization of available resources.

Mehta, et al in[18] proposed a dynamic load balancing model that considers utilizing resources under burstiness cases. They suggest an architecture which consists of four parts: Cloud controller server, Node controller server, Agents, and Virtual machines. All requests will first go to the cloud controller server then it will be transferred to the load balancer. Finally a virtual instance is selected by the load balancer based on the information supplied by the monitoring agent about CPU usage, memory and storage space usage.

The researchers claimed that this algorithm will ensure the optimum utilization of cloud resources, faster response time, and cut the economic cost for an organization. However they did not do any experiments or evaluations for their work.

Naik and Patel in [34] proposed load balancing under bursty environment for Cloud Computing. They present a dynamic load balancing algorithm which maintains the state of all virtual machine (VM) resources, and based on CPU, memory and storage space utilization, selects the less utilized VM resource to handle the request. They used a monitoring agent to continuously monitor the CPU usage, memory and storage space usage, and the current and the expected load for each virtual machine. Based on this information a Pheromone (or probability) assigned for every VM. When a request arrivers to the datacenter the load balancer transfer the request to the VM which has the least Pheromone.

Authors mentioned that their algorithm improved the performance but they did not provide any comparisons with other load balancing algorithms and they did not put their experiments results.

Ghorbani, et al in [35] proposed an approach to overcome the un-utilized resource provisioning and the power consumption problems under bursty and fractal behavior workload. It consists of two phases for resource utilization provisioning called "predictive and reactive provisioning". Firstly the forecasting module predicts the workload for the next control horizon, and then the controller estimates the

number of necessary resources such as processing cores, for the predictable part of the incoming load. In order to avoid consequences of forecasting errors, the system allocates extra resources that can be used to serve unpredictable load. This allocation is made based on the history of the system operation.

The proposed approach improves the resource utilization and the power consumption. On the other hand, if some prediction error happens beyond the estimation of the extra resources, it would be subject to delay in getting the resource till the system allocates available resources.

2.4.3 Load Balancing Approaches for Burst Workload in Other Distributed Systems:

The problem of bursty workload exists in other fields. For example, Chin, et al in [24] proposed a load balancing algorithm which applies to the local Domain Name Service (DNS) server for bursty web application traffic. The proposed load balancing algorithm is based on a performance indexing of each node in the server farm in order to select the most suitable nodes for the next task distribution. The performance indexing takes into account the current server load and the server performance metric of the server itself.

Zhang, et al in [36] proposed two hybrid schedulers, Hibred-1 and Hibred-2, which addresses the problem of bursty workload on computing grids. The proposed schedulers are using the duplication-invalidation method [37] which makes duplications of the received request. The hybrid scheduling combines the Qlen scheduler with the Rand scheduler. Given a global job, the Hybrid-1 scheduler uses the shortest queue length to locate the site for the first submission, and the remaining duplications are submitted to sites that are randomly selected. The Hybrid-2 scheduler uses the queue length as the ranking criterion to select two sites for submissions, and the remaining duplications are submitted to randomly select computing sites. Once the requested resources are allocated and the application's resource requirement is met, the application starts to run. Hence, all resources that are allocated later become redundant, and should be released.

There experiments showed that using 2 or 3 duplications can allow the hybrid schedulers to obtain the shortest queuing time for most global jobs. Comparing the two schedulers, Hybrid-1 has the advantage if strong burstiness is expected. In weak and non-bursty conditions, Hybrid-2 is preferable.

2.5 Summary

In this section detailed information about cloud computing was presented. In addition, load balancing was defined and the bursty problem which affect the performance load balancing in cloud computing had been clarified. .As shown in this section most algorithms did not consider the bursty workload case. Even those who work on bursty case, they failed to address the performance improvements especially the response time and the processing time. Also some works did not give a clear evaluation for their proposed algorithm to prove their contribution. This led us to think about developing a load balancing algorithm that can work efficiently by minimizing the response and the processing time under bursty workload.

Chapter 3

Proposed Method

Chapter 3 Proposed Method

In this chapter we will present our proposed method for solving performance degradation caused by bursty workload. The main idea of the new algorithm is to apply the suitable algorithm depending on the requests rate received by the datacenter. Adaptive algorithm consists of three main components: Burst Detector, load balancing algorithms, and fuzzifier. Later in this chapter, every step will be discussed in details.

3.1 Adaptive Load Balancing Algorithm

The request rates received by the datacenter are not constant all the time. Sometimes large number of requests aggregated in a small period of time creating a burst. This affect the performance of the load balancing algorithm as it increase the processing time and the repose time of the datacenter. The performance of several load balancing algorithms differs according to the users' requests rate. For example some algorithms work efficiently under low workload while its performance degraded under high workload and vice versa. To overcome burst problem and benefit from different load balancing algorithms advantages we propose a new load balancing algorithm called Adaptive algorithm.

Adaptive algorithm is a load balancing algorithm used by the datacenter to distribute the received tasks efficiently over the virtual machine under bursty workload by swapping between two policies depending on the requests rates. It consists of three main components as follows:

1- Burst detector.
2- Load Balancing Algorithms.
3- Fuzzifier.

When the datacenter receives a request, the burst detector determines the workload state (Normal or Burst). Depending on the burst detector decision, the datacenter will select the appropriate load balancing policy for that state. After that the selected load balancing algorithm will assign the received task to a suitable VM depending on the information supplied by the fuzzifier. When the VM complete its assigned task, it informs the data center. The main steps of the adaptive algorithm are shown in Figure 3.1.

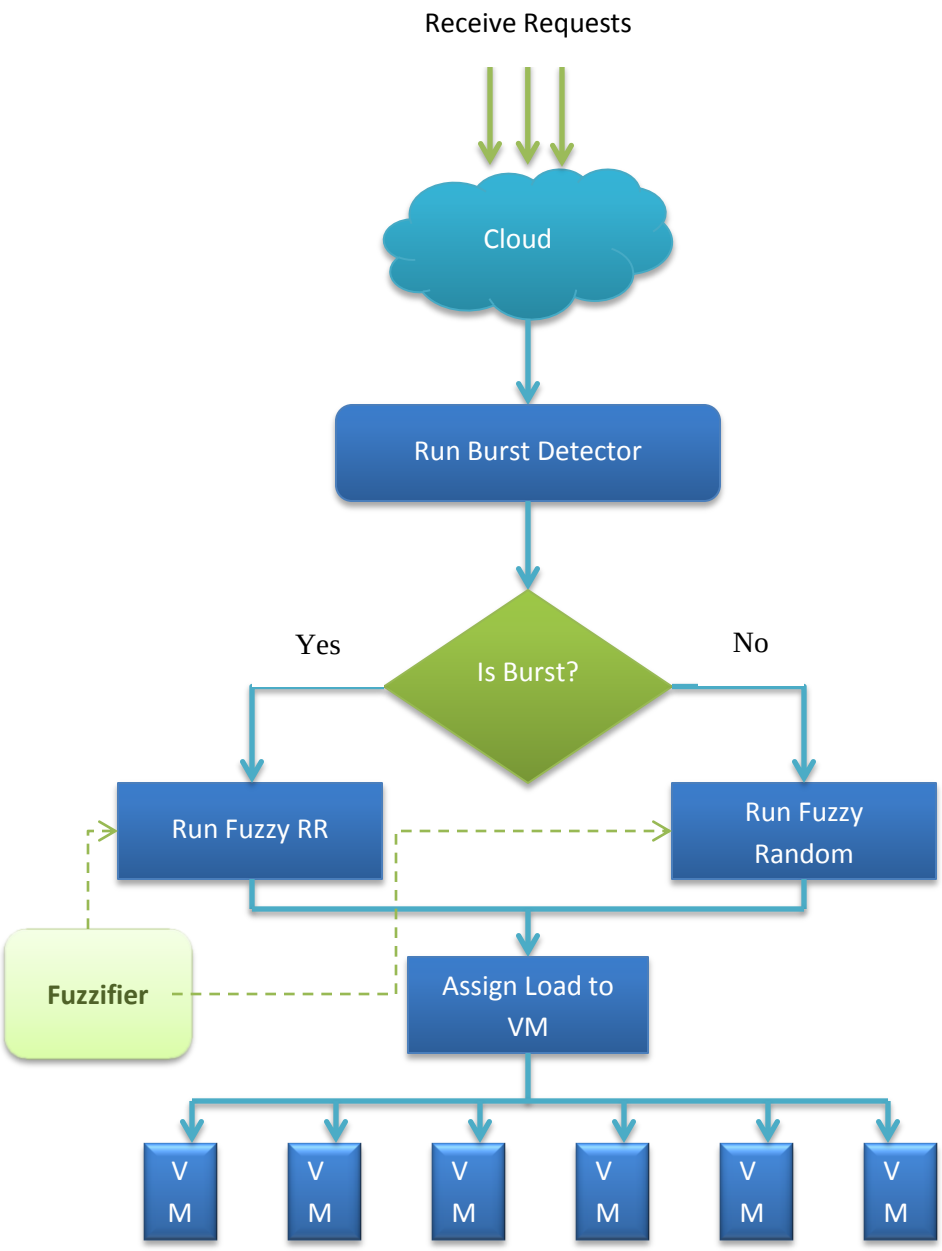


Figure 3.1 Adaptive Load Balancing Algorithm Flow Chart

3.1.1 Burst Detector

The burst detector is responsible for detecting the variation in the workload, and determines whether the state of the workload is burst or not using a specific threshold. When a request arrives the burst detector checks the rate of the requests in the last 15 minutes if it exceeds the threshold so it indicates that the status is burst. Depending on experiments, we found that 15 minutes is a suitable time interval which can give view about the requests rate and give evidence of possible burst occurrence. Depending on the detector decision, the datacenter will select the proper load balancing policy.

3.1.2 Load Balancing Algorithm

The proposed approach uses two load balancing algorithms, one can work efficiently in normal cases and the other one can work efficiently in burst cases. As we will see in experiments chapter, Random policy performs the best in low workload, and Round Robin performs the best in high workload.

1. Random Policy:
   When the burst detector decides that the workload state is normal, the random policy will be applied. The fuzzifier supply the random policy with a candidate list of more balanced VMs in the data center, then the policy will select one of these VMs randomly and assign the received task to it.

2. Round Robin Policy:
   Round Robin will be used when workload state is burst. The same as random, the fuzzifier will provide candidate list of the most balanced VMs for Round Robin policy. Then Round Robin will use this list to allocate VMs in a cycle manner.

3.1.3 Fuzzifier

The main function of fuzzifier is to enhance the decision of the load balancing algorithm by providing a list of the most balanced VMs in the data center and deliver it to the load balancer to allocate one VM in this list.

The fuzzifier is consisting of Fuzzy Inference System (FIS) to simulate the way of human decision making by using fuzzy control rules and linguistic parameters. In our wok, the FIS uses two inputs which are processor speed and the load in VM, and balanced load as the output. Twelve IF-THEN rules are proposed for FIS to be used in taking the decision. Those rules are based on previous studies such as [30].The IF-THEN rules are stated in Figure 3.2 below:

```
RULE 1 : IF processor_speed IS low AND assigned_load IS least THEN balanced_load IS high;
RULE 2 : IF processor_speed IS low AND assigned_load IS medium THEN balanced_load IS medium;
RULE 3 : IF processor_speed IS low AND assigned_load IS high THEN balanced_load IS low;
RULE 4 : IF processor_speed IS medium AND assigned_load IS least THEN balanced_load IS high;
RULE 5 : IF processor_speed IS medium AND assigned_load IS medium THEN balanced_load IS medium;
RULE 6 : IF processor_speed IS medium AND assigned_load IS high THEN balanced_load IS low;
RULE 7 : IF processor_speed IS high AND assigned_load IS least THEN balanced_load IS high;
RULE 8 : IF processor_speed IS high AND assigned_load IS medium THEN balanced_load IS medium;
RULE 9 : IF processor_speed IS high AND assigned_load IS high THEN balanced_load IS low;
RULE 10 : IF processor_speed IS very_high AND assigned_load IS least THEN balanced_load IS high;
RULE 11 : IF processor_speed IS very_high AND assigned_load IS medium THEN balanced_load IS high;
RULE 12 : IF processor_speed IS very_high AND assigned_load IS high THEN balanced_load IS medium;
```

Figure 3.2 Base Rules for FIS

The member functions for the two inputs (processor speed and the load in VM) and the output is presented in Figures 3.3, 3.4, 3.5 respectively.

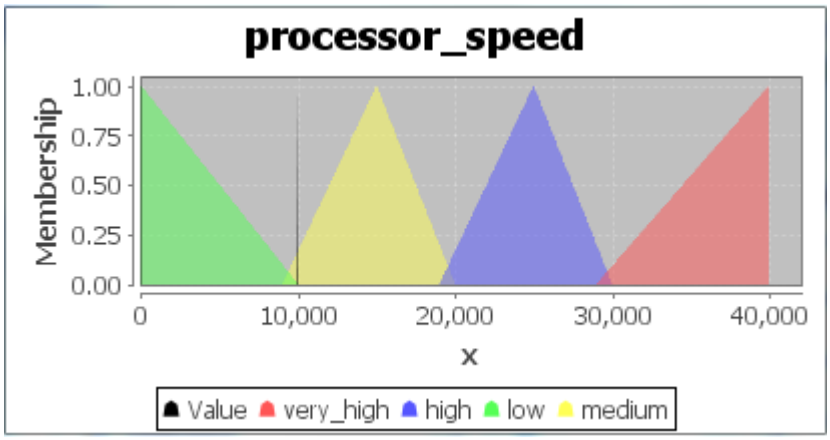


Figure 3.3 Membership input function of Processor Speed

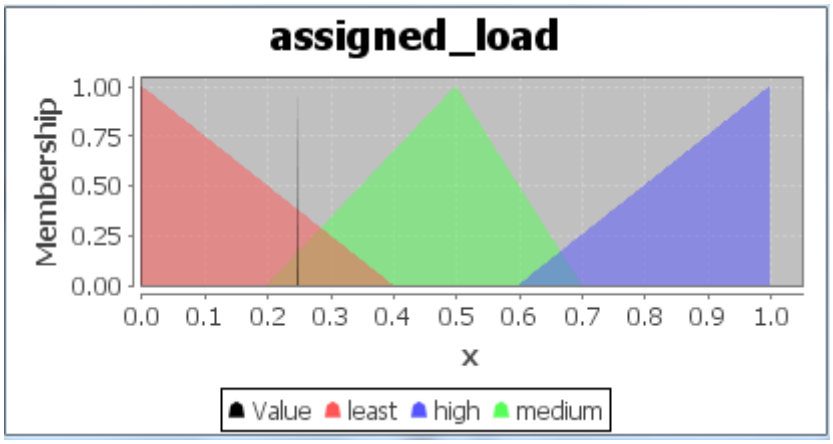


Figure 3.4 Membership input function of Assigned Load

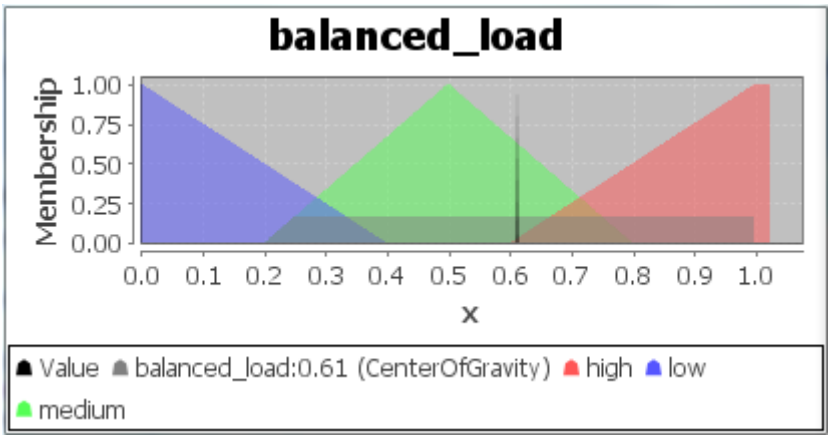


Figure 3.5 Membership output function of Balanced Load

For our FIS we use open source Java library called jFuzzyLogic [38] which offers a fully functional and complete implementation of a fuzzy inference system according to International Electrotechnical Commission (IEC) [1] standard, providing a programming interface and Eclipse plugin to easily write and test code for fuzzy control applications [39, 40].

3.2 Implementation

The implementation of the Adaptive Load Balancing Algorithm is done using Java programming language because Cloudsim - the simulator used in experiments- is built using java.

Table 3.1 present the pseudo code for the proposed algorithm.

The Adaptive algorithm starts with collecting information about the CPU speed and the current load for every VM. That information used by the fuzzifier which decides the balanced degree for every VM. Then the requests rate in last 15 minutes is checked if it is higher than a threshold, this indicates that there is a burst so the fuzzy RR is applied. Otherwise, the fuzzy Random is applied. Table 3.1 below presents the pseudo code for Adaptive Algorithm.

Table 3.1 Pseudo code for Adaptive Load Balancing Algorithm

Algorithm 3-1 ADAPTIVE LOAD BALANCING
Input: *: requestRate* Number of received requests in last 15 minutes
Output: *vmid* The ID of the selected VM

```
1.  CALL Fuzzifier()
2.  IF (requestRate > threshould) THEN vmid ← fuzzyRR()
3.  ELSE  vmid ← fuzzyRandom()
4.  RETURN vmid      //return the allocated VM ID to the Data center
```

The Fuzzy Roiund Robin has the same concept of original Round Robin. It allocates VMs to nodes in a cyclic manner. The main difference between RR and Fuzzy RR is that the original RR is static because the selection of VM does not built on information about VM while Fuzzy RR uses FIS to detect the balanced level of the VM then the RR concept is applied on a set of the most balanced VMs. The Fuzzy RR selects the next VM from the set which contains the high balanced load VMs. If the set is empty then the selection process will be done on the set with medium balanced load VMs. If it is empty this means that the system is under very high load and the selection will be on the low balanced set.

Fuzzy Random is applied when there is no burst. Instead of allocating VM randomly an improvement is done on Random algorithm to make the allocation process wiser. The same as Fuzzy RR, the FIS classify the VMs into three classes, high balanced, medium balanced, and low balanced. The Fuzzy Random

[1] The International Electrotechnical Commission (IEC) is the world's leading organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

allocates VM in the high balanced set randomly. If it is empty it will allocate one from the medium balanced. If it is empty it will select one from the low balanced VMs set.

The Fuzzifier scan the VMs periodically every 1 sec, in order to classify them under three classes high balanced, medium balanced and low balanced. The fuzzifier take the decision based on information about the VM specifically processor speed and current assigned load. Depending on predefined IF-THEN Rules the fuzzifier decide the balance level on the VM and put it in the high balanced, medium balanced or low balanced set. Table 3.2 illustrates the Fuzifier Pseudo Code.

Table 3.2 Pseudo code for Fuzzifier

Algorithm 3-2 FUZZIFIER

Input: *processorSpeed* VM processor speed,
vmid VM Id,
assignedLoad Current load for VM.

Output: *highBalancedLoad[0…k]* list of VM with high balanced load,
mediumBalancedLoad[0…m] list of VM with medium balanced load,
lowBalancedLoad[0…n] list of VM with low balanced load.

```
1.  //set input variable for fuzzy inference system
2.  setVariable("processor_speed", processorSpeed)
3.  setVariable("assigned_load", assignedLoad)
4.  // get fuzzy output result
5.  balancedLoad ← getVariable("balanced_load").getValue()
6.  IF (balancedLoad IS high) THEN
7.      highBalancedLoad[count+1]← vmid
8.  ELSE IF (balancedLoad IS medium) THEN
9.      mediumBalancedLoad[count+1]← vmid
10. ELSE IF (balancedLoad IS low) THEN
11.     highBalancedLoad[count+1]← vmid
12. ENDIF
```

3.3 Summery

In this chapter the Adaptive algorithm was presented and discussed in details. The Adaptive algorithm has three main components: burst detector, load balancing algorithms, fuzzifier. The burst detector is responsible for detecting the status of the workload if it is bursty the Fuzzy RR is applied; otherwise the Fuzzy Random is applied. Fuzzy RR and Fuzzy random allocate VM based on the information supplied by the fuzzifier which classify the VMs into three classes high balanced, medium balanced and low balanced. The Fuzzy Random and the Fuzzy Round Robin select one VM from the high balanced class (randomly or in cyclic manner) and assign the received request to it.

Chapter 4

Experiments and results

Chapter 4 Experiments and Results

The performance of load balancing algorithms varies according to the intensity of the received requests. One of the problems which cause degradation in the performance of load balancing algorithm is bursty workload. In this chapter we will present the experiments had been done on the proposed method and compare it with other load balancing algorithms. The experiments are divided into three parts. The first part presents a study for the performance of load balancing algorithms in normal and burst workload. The second part consists of experiments which helped us to choose the best algorithm for high load cases and the best algorithm for low load cases to be used in our proposed algorithm. The third part contains the experiments which test and evaluate the performance of the proposed algorithm.

For every experiment, firstly we will present in details the objective, configuration and the result for every experiment. And then at the end of each part, we shall summarize and discuss all the experiments results. All experiments are done using CloudAnalyst Simulator [41], and the simulation duration is one day.

4.1 Cloudsim Simulator

The proposed algorithm should be tested in a cloud computing environment. For that purpose we have two choices, either use a real test beds such as Amazon EC2 or use simulation tools to simulate a cloud environment. In our work we prefer to use simulator for several reasons:

1. The experiments on real cloud environment will be limited to the cloud provider configurations, so tests cannot examine different configurations.
2. The conditions prevailing in the Internet-Base environment are beyond the control of the tester and this may affect the tests results.
3. Applying tests on real cloud incurs payments.

Table 4.1 presents a comparison between real cloud and simulator [42]

Table 4.1 Real Cloud and Simulator Comparison

Real Cloud	Simulator
Experiments cannot examine different configuration.	Provide possibility to evaluate the hypothesis using different configurations.
Internet environment is out of control.	The tester can generate different Internet environments which serve his hypothesis.
Experiments incur payments	Allow repeatable and controllable experiments without costs

There are varieties of simulators for modeling cloud computing but CloudSim is one of the best. CloudSim is an open source Java-based simulation toolkit developed by the GRIDS Laboratory at University of Melbourne[43]. It is an extensible simulation toolkit that enables modelling and simulation

of Cloud computing environments [44]. The CloudSim toolkit supports modeling of infrastructures containing Cloud Data Centers, Virtual Machines (VM), users, user workloads, and pricing models [45].

4.1.1 CloudAnalyst Simulator

CloudAnalyst is a tool developed at the University of Melbourne. It is a graphical simulation tool based on Cloudsim for modeling and analysis behavior of cloud computing environment, which supports visual modeling and simulation of large-scale applications that are deployed on Cloud Infrastructures [43, 45, 46]. As depicted in Figure 4.1 CloudAnalyst is built directly on top of CloudSim toolkit, adding GUI feature which gives a high capability of configuration in quick and easy manner. In this various configuration parameters can be set like number of users, number of request generated per user per hour , number of virtual machines, number of processors, amount of storage, network bandwidth and other necessary parameters [20].

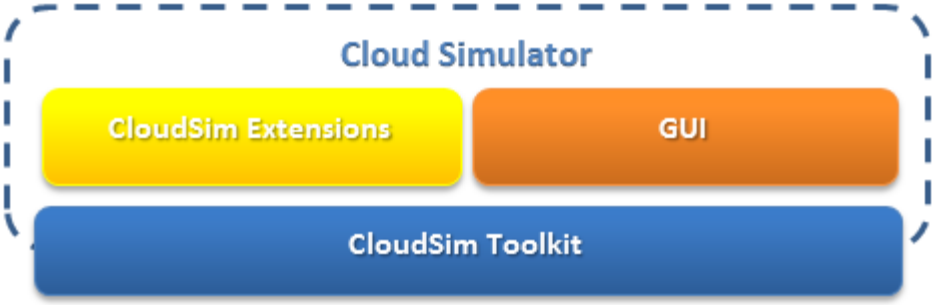


Figure 4.1 CloudAnalyst Archeticher [20]

The main features of CloudAnalyst are the following: [45]

1. Easy to use Graphical User Interface (GUI).
2. Ability to define a simulation with a high degree of configurability and flexibility.
3. Repeatability of experiments.
4. Graphical output which shows the results in the form of chart and tables which is easy to understand.

4.1.2 CloudAnalyst Components:

The main components of CloudAnalys simulator are:

1. **Region:** CloudAnalyst divide the world into 6 regions. Those regions are the 6 main continents in the world. This segmentation provides a level of realistic for the large scaled simulation being attempted in the CloudAnalyst. Cloud entities such as User Bases and Data Centers are distributed over those regions[20].

2. **UserBase:** This component models a group of users and generates traffic representing the users. A single User Base may represent thousands of users but is configured as a single unit and the traffic generated in simultaneous transmission represents the size of the user base[20].

3. **Datacenter:** The core hardware infrastructure services are modeled in the simulator by a Datacenter component. It encapsulates a set of compute hosts (servers) that can be either homogeneous or heterogeneous as regards to their resource configurations[42].

4. **Host:** This class models a physical server in Data Center. Configuration of this component includes the amount of memory and storage, a list of processing elements (to represent a multi-core machine), an allocation policy for sharing the processing power among virtual machines (space-shared, time-shared), and policies to provisioning memory and bandwidth to the virtual machines [47].

5. **VirtualMachine:** This class models an instance of a VM, whose management during its life cycle is the responsibility of the Host component. The Host can simultaneously instantiate multiple VMs and allocate cores based on predefined processor sharing policies (space-shared, time-shared). Vertual Machine is responsible for processing the received requests[44].

6. **VMProvisioner:** This abstract class represents the provisioning policy for allocating VMs to Hosts. The chief functionality of the VMProvisioner is to select available host in a data center, which meets the memory, storage, and availability requirement for a VM deployment. The default SimpleVMProvisioner implementation provided with the CloudSim package allocates VMs to the first available Host that meets the aforementioned requirements[42].

7. **VmLoadBalancer:** Data center controller uses VM load balancer to determine which VM should be assigned the next requests (Cloudlet) for processing [48].

8. **InternetCloudlet.** An InternetCloudlet is a grouping of user requests. The number of requests bundled into a single InternetCloudlet is configurable in CloudAnalyst. The InternetCloudlet carries information such as the size of a request execution command, size of input and output files, the originator and target application id used for routing by the Internet and the number of requests [46].

Figure 4.2 presents the main components of CloudAnalyst simulator

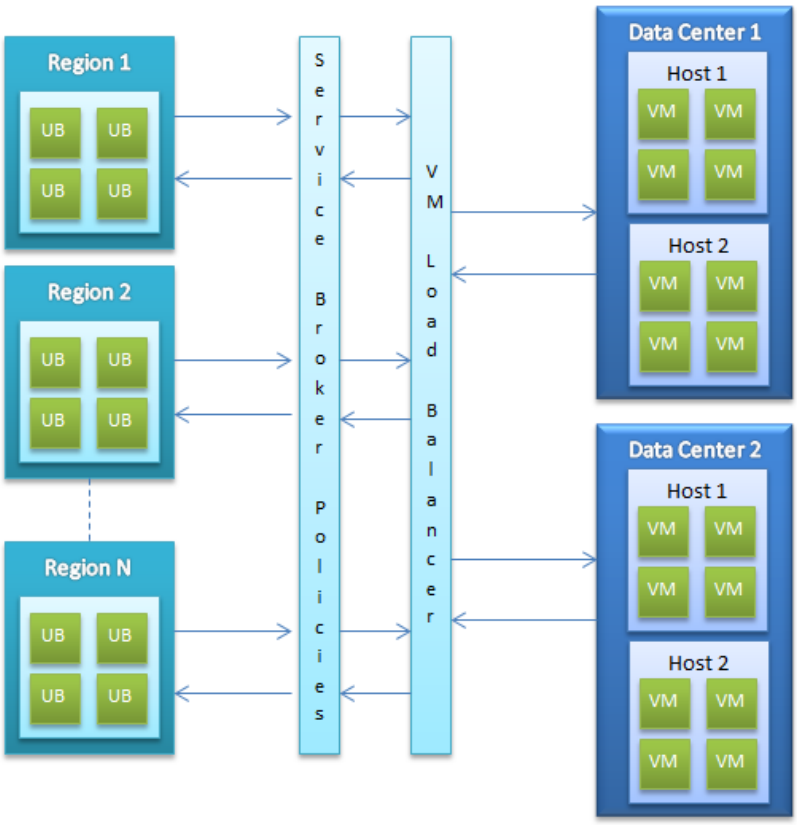


Figure 4.2 Main Components of CloudAnalyst Simulator [46]

All those components described above and some other components work together to simulate the interaction between the provider and the consumer. Let us see how those components deal with users requests and what is the sequence of processing those requests in Figure4.3 [48].

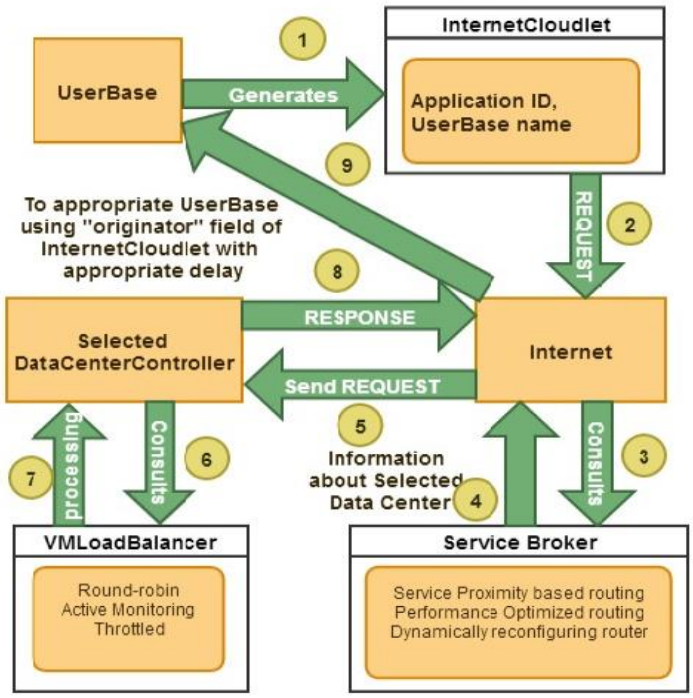


Figure 4.3 Routing of User Requests [48]

1. First of all the User Base generates an Internet Cloudlet which encapsulate the application id and the name of user base.
2. Then the REQUEST (Cloudlet) is sent to the Internet.
3. The Internet asks the Service Broker to select a data center.
4. The Service Broker sends information about the selected data center to the Internet.
5. Depending on the data center geographical location the Internet adds appropriate network delay with the REQUEST and sends to the selected data center controller.
6. Selected data center controller uses any one of the virtual machines load balancing policy.
7. VmLoadBalancer assign the request to the selected VM.
8. Selected data center sends the RESPONSE to the Internet after processing the REQUEST.
9. The Internet sends the RESPONSE to the User Base.

4.1.3 CloudAnalyst Metrics

One of the most important features of CloudAnalyst is that it gives a final report for the simulation result. This report summarizes the results of some metrics which are used for evaluation process. Those metrics are listed below [23]:

1. **Overall response time: Minimum, Maximum and Average in ms.**
   This metric presents the overall response time for the whole cloud system. The Average score present the average response time for all user bases, while the Minimum and the Maximum present the highest and the lowest response time between user bases. For evaluation process, researchers considered the average response time to compare their works with others.

2. **Overall processing time in the data center: Minimum, maximum and average in ms.**
   The same as response time CloudAnalyst calculate the average processing for all the data centers. Also it presents the highest and the lowest processing time scored between all the data centers.
3. **Response time per user :Minimum, maximum and average**
   For every user base, the minimum, maximum and average response time is calculated and presented in a summery table.
4. **Minimum, maximum and average time per data center.**
   For every data center, CloudAnalyst calculate the minimum, maximum and average processing time and present it in a summery table.
5. **Cost of execution in $**
   The final result report gives details about the cost of execution considering:
   - Virtual machine total cost.
   - Cost per VM of Data Center.
   - Cost of data in each data center.
   - Total cost in each data center.

4.2 Evaluation Metrics

In this research our goal is to improve the performance of cloud system. In order to evaluate the performance of our proposed algorithm we select the most important metrics which are:

1. **Average response time:** It can be measured as, the time interval between sending a request and receiving its response. It should be minimized to boost the overall performance [49].
2. **Average processing time**: It is the amount of time actually needed to process a task.

We measured these two metrics for our algorithm and then compare them with the response and processing time for three popular load balancing algorithms which are: Round Robin (RR), ESCE, and Random, in order to evaluate the improvement of our algorithm.

4.3 Experiments Part I

In this part we studded the performance of most popular load balancing algorithms: RR, ESCE, and Random, in two workload cases, normal and burst. The aim of this part is to examine the effect of bursty workload on the performance of different load balancing algorithms.

4.3.1 Experiment 1: Normal Workload Using Homogeneous Hosts

This experiment is done to test the performance of three popular load balancing algorithms in normal workload. In this experiment we will study the performance of Round Robin, Equally Spread Current Execution (ESCE), and Random Algorithm using two different configurations for normal workloads in homogeneous environment (same number of CPU in the Data Center and same CPU speed).

4.3.1.1 Configurations

For this experiment, two different configurations for user bases are used in order to test the performance of the load balancing algorithms under two different samples of normal workloads. In this experiment the number of processors is equal in all the physical hosts. The duration of the simulation is 1 Day.

4.3.1.1.1 User Base Configuration

a. **Configuration 1:**

In configuration 1 we use normal workload with two Peak Hours for every user base. The configurations used in this experiment are as followed:

Table 4.2 User Base Configurations for Experment 1 (Config. 1)

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	15	400000	40000
UB2	1	12	100	15	17	100000	10000
UB3	2	12	100	20	22	300000	30000
UB4	3	12	100	1	3	150000	15000
UB5	4	12	100	21	23	50000	5000
UB6	5	12	100	9	11	80000	8000

b. **Configuration 2:**

In configuration 2, we increased the Peak Hours for every user base and minimize the average of users in the Peak and the Off Peak periods as shown in Table 4.3.

Table 4.3 User Base Configurations for Experiment 2 (Config. 2)

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak User Base Configration for Normal Workload Users
UB1	0	12	100	6	15	20000	2000
UB2	1	12	100	15	17	10000	1000
UB3	2	12	100	17	22	14000	1400
UB4	3	12	100	1	6	15000	1500
UB5	4	12	100	21	23	5000	500
UB6	5	12	100	5	11	8000	800

4.3.1.1.2 Data Center Configuration

One Data Center is used with 50 Virtual Machines and 20 Physical HW Units all of them have the same number of processors and the same processor speed. Table 4.4 present the Data Center configurations while Table 4.5 shows the configuration of the Physical Hardware of the Data Center.

Table 4.4 Data Center Configurations for Experiment 1

Data Center	#VMs	Image Size	Memory	BW
DC1	50	10000	1024	1000

Table 4.5 Physical Hardware Configurations for DC1 for Experiment 1

Num. of Physical HW Unit	Memory (Mb)	Storage (Mb)	Available BW	Number of Processors	Processor Speed	VM Policy
20	2048	100000	10000	4	100	TIME_SHARED

4.3.1.2 Results

The configurations above have been used for each load balancing polices one by one and depending on that we get the following results:

a. **Result 1:**

4.3.1.2.1 Data Center Hourly Loading

According to configuration 1 for User Base, the curve of the requests received in Data Center1 per hour is as shown in Figure 4.4. In this figure the peaks appear in the curve present the peak hours of the user bases.

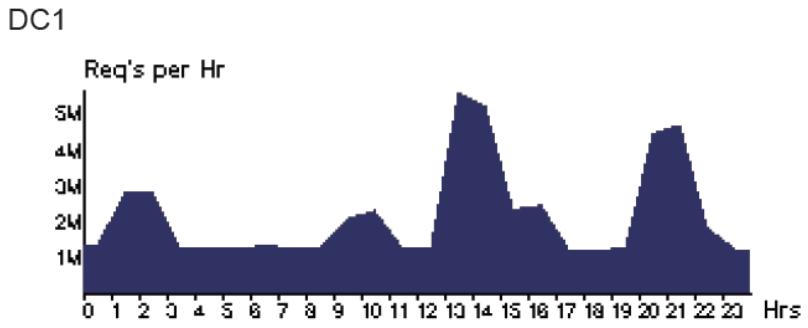


Figure 4.4 Data Center Hourly Loading for Normal Workload (Config.1)

4.3.1.2.2 Response Time

The results from applying Configuration1 shows that Random policy has the best response time while the ESCE and Round Robin came next with convergent results. The response time chart for the three load balancing algorithms is shown in Figure 4.5.

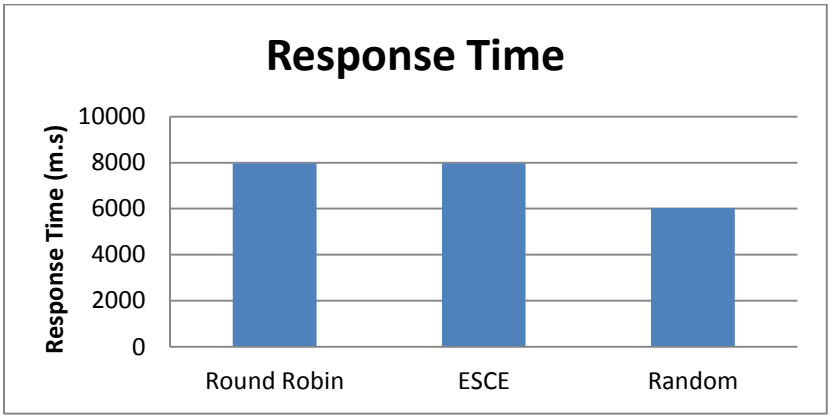


Figure 4.5 Response Time Chart for Normal Workload (Config.1)

4.3.1.2.3 Processing Time

The results show that Random algorithm has the best processing time compared with Round Robin and ESCE. Figure 4.6 explains the difference in processing time for the three algorithms.

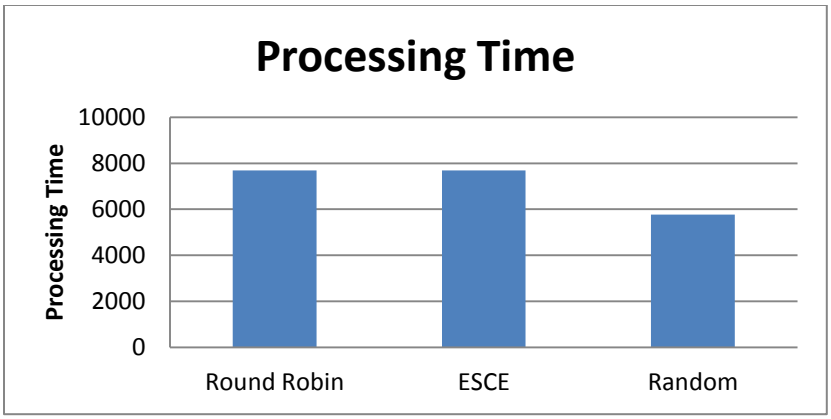


Figure 4.6 Processing Time Chart for Normal Workload (Config.1)

b. Result 2

4.3.1.2.4 Data Center Hourly Loading

When applying Configuration 2 of User Base, the Hourly Loading graph is as presented in Figure 4.7. As we can see in this figure there is not a clear peak in the curve as it was in Configuration 1. This is because the peak hours of the User Bases in Configuration 2 are longer than ones in Configuration 1, and there are overlaps between them.

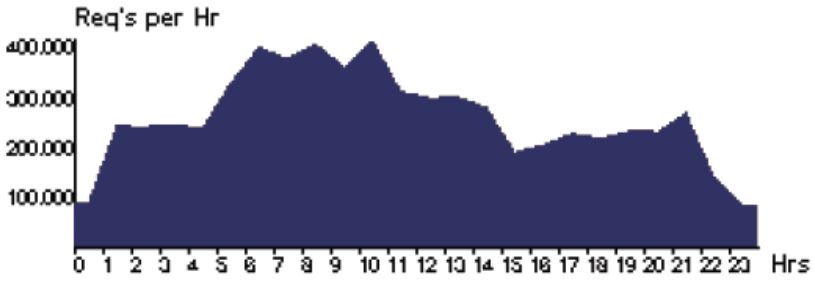


Figure 4.7 Data Center Hourly Loading for Normal Workload (Config.2)

4.3.1.2.5 Response Time

The same as Configuration 1, Random recorded the best response time while ESCE and Round Robin have almost the same response time records. Figure 4.8 presents the response time chart for the three algorithms.

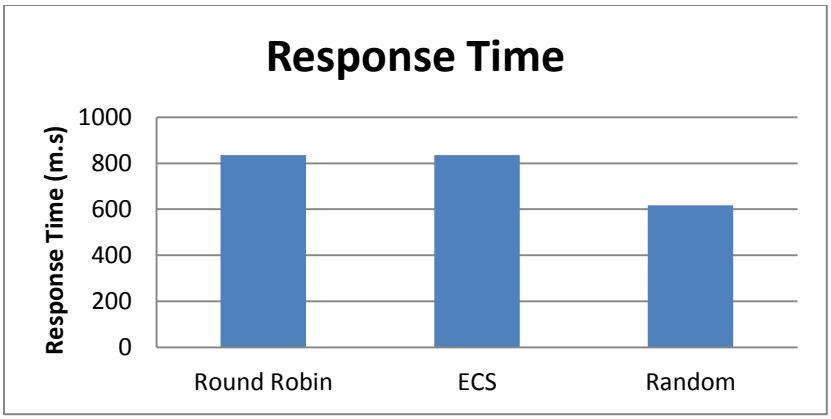


Figure 4.8 Response Time Chart for Normal Workload (Config.2)

4.3.1.2.6 Processing Time

Again as it is clear in Figure 4.9 Random algorithm has the best processing time compared with Round Robin and ESCE.

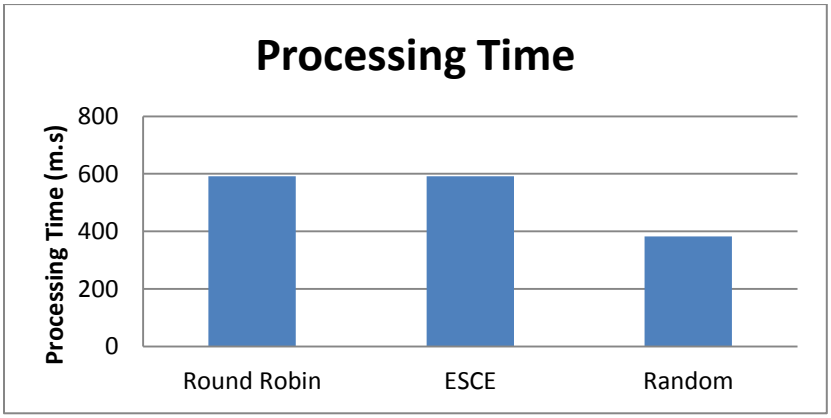


Figure 4.9 Processing Time Chart for Normal Workload (Config.2)

4.3.1.3 Discussion

In this experiment two loads are applied by changing the duration of Peak Hours and the average number of users in every User Base. All the Physical Hosts have the same number of CPU and the same CPU speed. The results show that in both configurations, Random algorithm has better response time and processing time than ECSE and Round Robin algorithms. Figure 4.10 and Figure 4.11 summarize the response time and the processing time results for both configurations.

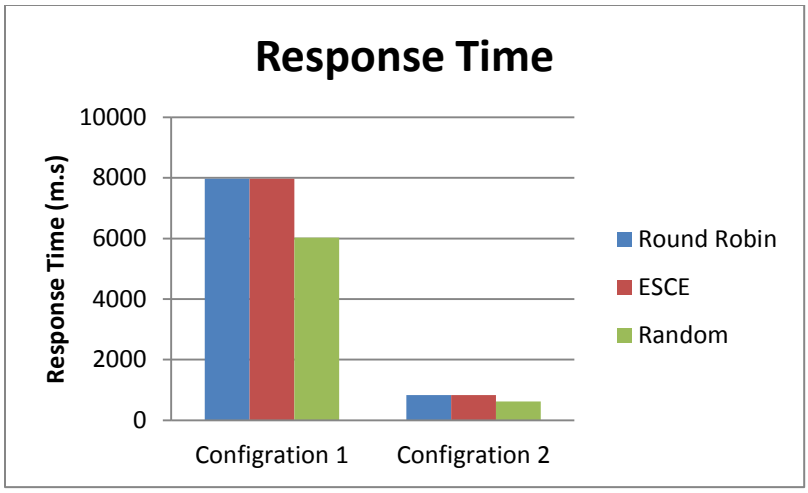


Figure 4.10 Response Time Chart for Normal Workload

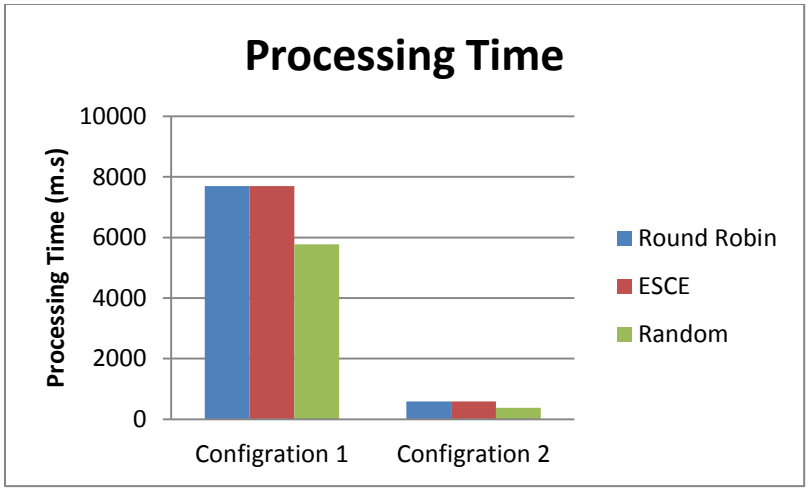


Figure 4.11 Processing Time Chart for Normal Workload

4.3.2 Experiment 2: Normal Workload Using Heterogeneous Hosts

This experiment is done to test the performance of three load balancing algorithms in normal workload. In this experiment we will study the performance of Round Robin, Equally Spread Current Execution (ESCE), and Random Algorithm in two different normal workloads and heterogeneous hosts (different CPU characteristics in the Data Center).

4.3.2.1 Configurations

In this experiment, we used the same two configurations for user bases in experiment 1 in heterogeneous environment. Different number of processors and different processor speeds in the physical hosts are used. The duration of the simulation is 1 Day.

4.3.2.1.1 User Base Configuration

a. **Configuration 1:**

In configuration 1 we use normal workload with two Peak Hours for every user base. The configurations used in this experiment are as followed:

Table 4.6 User Base Configuration for Experment 2

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	15	400000	40000
UB2	1	12	100	15	17	100000	10000
UB3	2	12	100	20	22	300000	30000
UB4	3	12	100	1	3	150000	15000
UB5	4	12	100	21	23	50000	5000
UB6	5	12	100	9	11	80000	8000

b. Configuration 2:

In configuration 2, the Peak Hours for every user base was increased and the average of users in the Peak and the Off Peak periods was minimized as shown in Table 4.7.

Table 4.7 User Base Configurations for Experiment 2 (Config. 2)

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	6	15	20000	2000
UB2	1	12	100	15	17	10000	1000
UB3	2	12	100	17	22	14000	1400
UB4	3	12	100	1	6	15000	1500
UB5	4	12	100	21	23	5000	500
UB6	5	12	100	5	11	8000	800

4.3.2.1.2 Data Center Configuration

One Data Center is used with 50 Virtual Machines and 5 Physical HW Units. The five hosts have different number of processors and different processors speed. Table 4.8 present the Data Center configurations while Table 4.9 shows the configuration of every Physical Hardware in the Data Center.

Table 4.8 Data Center Configurations for Experiment 2

Data Center	#VMs	Image Size	Memory	BW
DC1	50	10000	1024	1000

Table 4.9 Physical Hardware Configurations for DC1 for Experiment 2

ID	Memory (Mb)	Storage (Mb)	Available BW	Number of Processors	Processor Speed	VM Policy
0	2048	100000	10000	4	2000	TIME_SHARED
1	2048	100000	10000	5	5000	TIME_SHARED
2	2048	100000	10000	2	9000	TIME_SHARED
3	2048	100000	10000	2	1000	TIME_SHARED
4	2048	100000	10000	2	15000	TIME_SHARED

4.3.2.2 Results

The configurations above had been used for each load balancing polices one by one and depending on that we get the following results:

a. **Result 1:**

4.3.2.2.1 Data Center Hourly Loading

According to configuration 1 for User Base, the curve of the requests received in Data Center1 per hour is as shown in Figure 4.12. In this figure the peaks appear in the curve present the peak hours of the user bases.

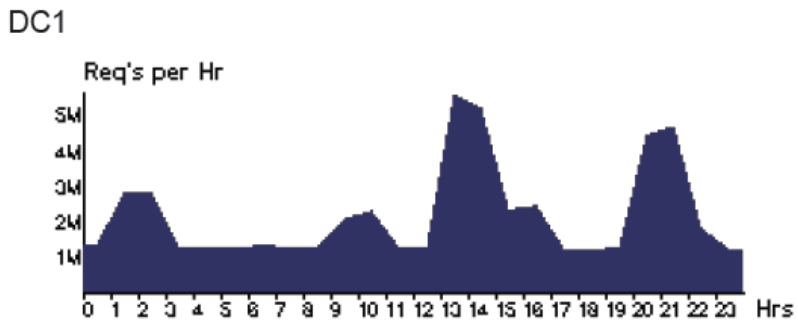


Figure 4.12 Data Center Hourly Loading for Normal Workload (Config.1)

4.3.2.2.2 Response Time

The results from applying Configuration1 shows that Round Robin policy has the best response time while the ESCE has the worst one. The response time chart for the three load balancing algorithms is shown in Figure 4.13.

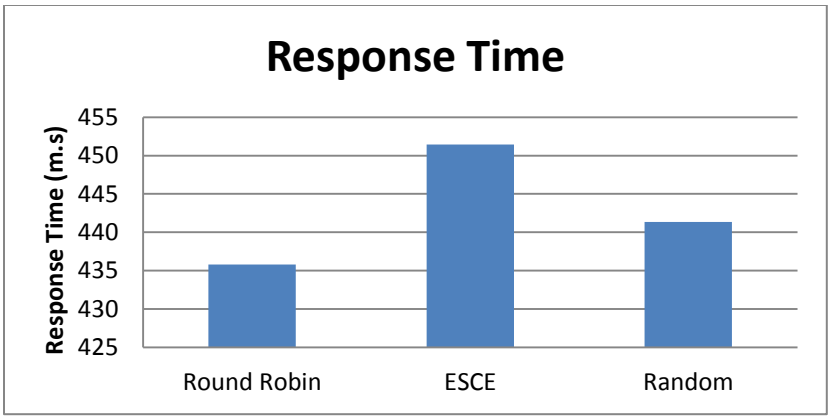


Figure 4.13 Response Time Chart for Normal Workload (Config.1)

4.3.2.2.3 Processing Time

Comparing the three algorithms, Round Robin recorded the best processing time with 198.08 ms then Random with 204.58 ms, and the worst one was ESCE with 214.06 ms. The chart in Figure 4.14 clarify the differences in processing time for the three algorithms.

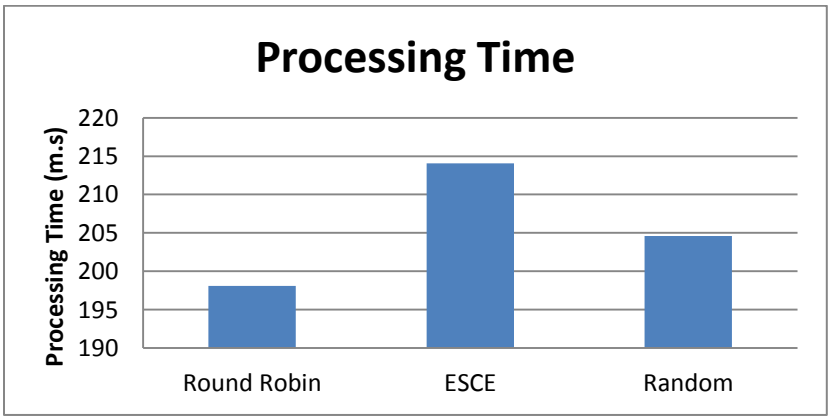


Figure 4.14 Processing Time Chart for Normal Workload (Config.1)

b. Result 2:

4.3.2.2.4 Data Center Hourly Loading

When applying Configuration 2 of User Base, the Hourly Loading graph is as presented in Figure 4.13. As we can see in this figure there is not a clear peak in the curve as it was in Configuration 1. This is because the peak hours of the User Bases in Configuration 2 are longer than ones in Configuration 1, and there are overlaps between them.

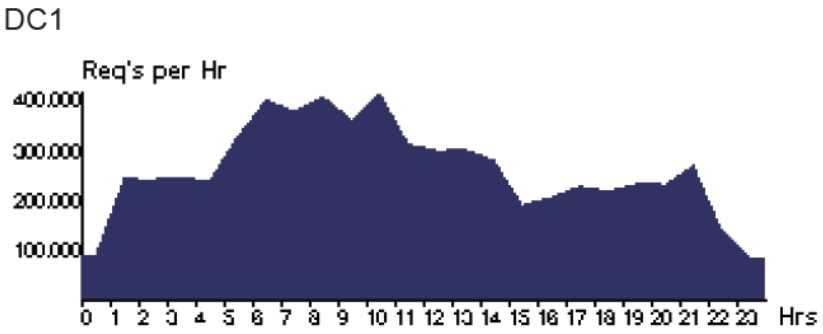


Figure 4.15 Data Center Hourly Loading for Normal Workload (Config.2)

4.3.2.2.5 Response Time

The same as Configuration 1 results, Round Robin has the best response time while ESCE has the worst one. But as we can see in Figure 4.16, the results are close pretty much.

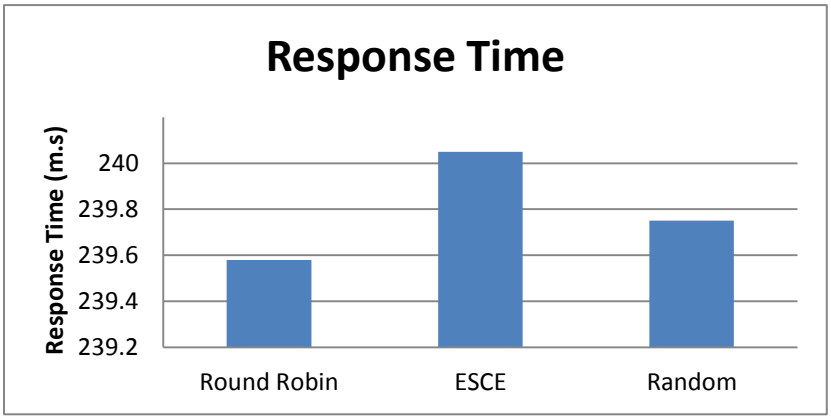


Figure 4.16 Response Time Chart for Normal Workload (Config.2)

4.3.2.2.6 Processing Time

As it is shown in the chart below, although Round Robin has the best processing time but the difference between the processing times for the three algorithms is very small. Figure 4.17 explains the processing time results for this experiment.

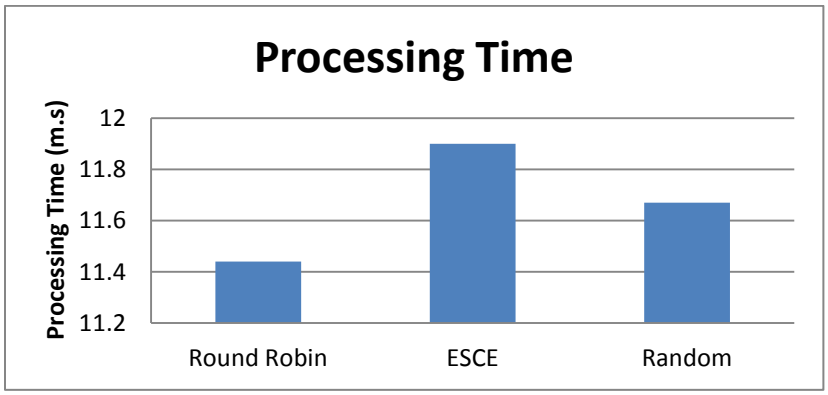


Figure 4.17 Processing Time Chart for Normal Workload (Config.2)

4.3.2.3 Discussion

In this experiment the two workloads used in Experiment 1 have been used here but in heterogeneous environment. When the results of experiment 1 and experiment 2 are compared, we can notice that using different configurations for the Physical Hosts in the Data center affect the performance of the load balancing algorithm. The results show that Round Robin has the best performance in heterogeneous hosts while on the other hand Random is the best in homogenous hosts.

Figure 4.18 and Figure 4.19 summarize the response time and the processing time results for both configurations in the first and second experiment.

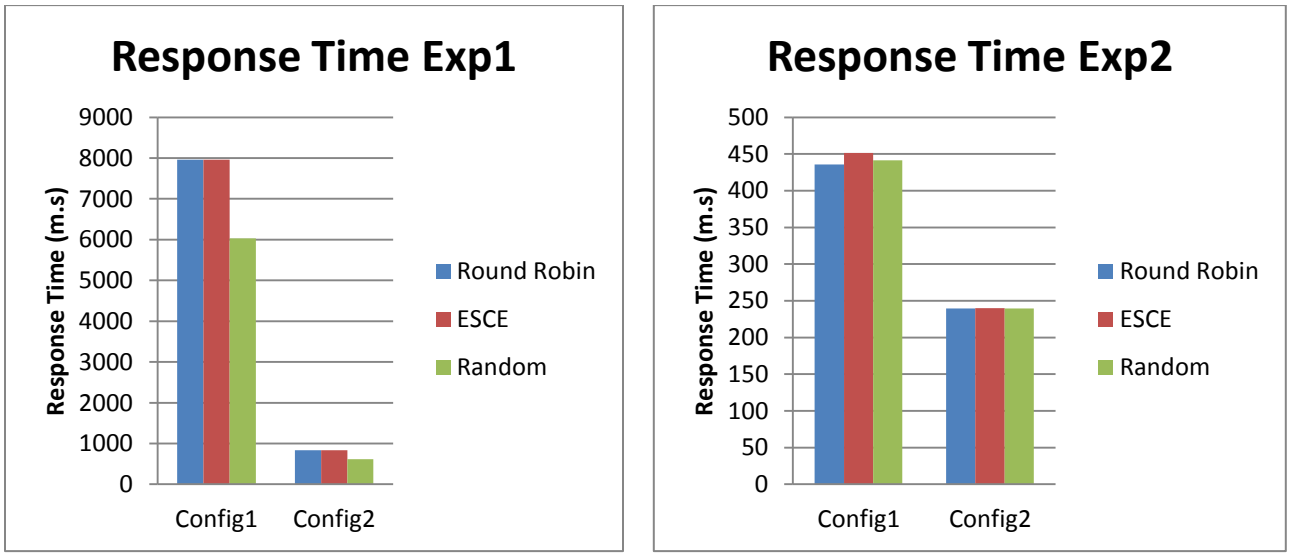


Figure 4.18 Response Time Chart for Normal Workload

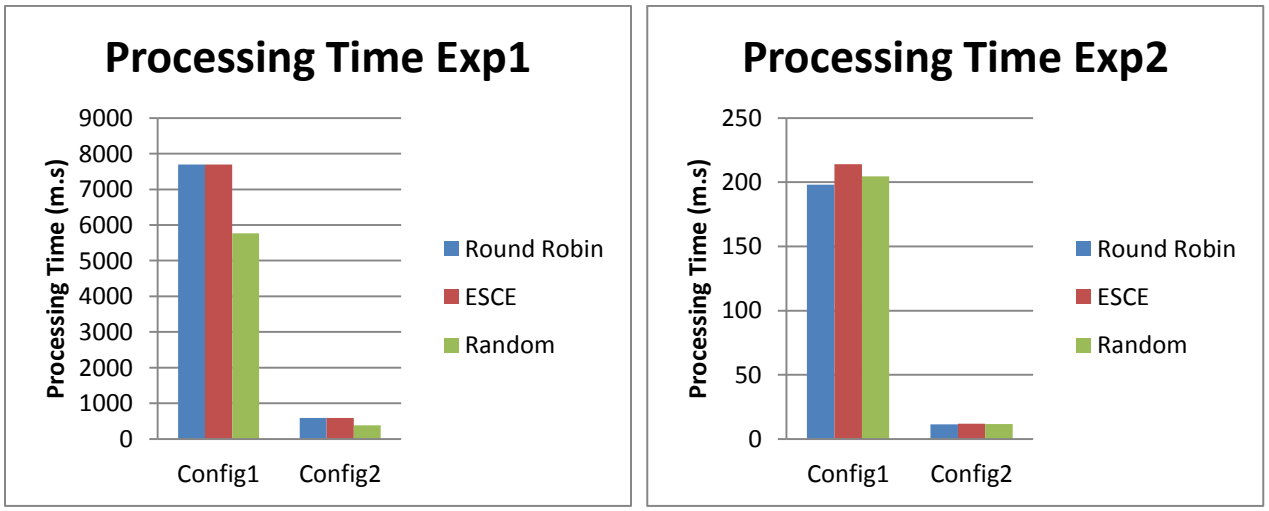


Figure 4.19 Processing Time Chart for Normal Workload

4.3.3 Experiment 3: Burst Workload

The main goal of this experiment is to study the impact of the bursty workload on the performance of Round Robin, Equally Spread Current Execution (ESCE), and Random Algorithm.

4.3.3.1 Configurations

In this experiment we are going to examine two level of burst, high and medium. For this goal two different configurations will be applied.

4.3.3.1.1 User Base Configuration

a. Configuration 1 High Burst:

In this configuration we use 6 User Bases all of them in the same region to ignore the transmission delay between regions. Every user base has one hour of peek and the average number of users in this hour is 800000 users. Table 4.10 list in details User Base configuration

Table 4.10 User Base Configurations for Experiment 3 (Config.1)

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	14	800000	40000
UB2	0	12	100	16	17	800000	10000
UB3	0	12	100	20	21	800000	30000
UB4	0	12	100	1	2	800000	15000
UB5	0	12	100	6	7	800000	50000
UB6	0	12	100	9	10	800000	80000

b. Configuration 2 Medium Burst:

Configuration 2 presents a medium level of bursty workload. The same as Configuration 1 all User Bases are in the same region and every one of these user bases has one peak hour. In contrast, in this configuration the average number of users in peak hour = the average number of users off peak * 10. Table 4.11 illustrates the User Base characteristics.

Table 4.11 User Base Configration for Experiment 3 (Config. 2)

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	14	400000	40000
UB2	0	12	100	9	10	100000	10000
UB3	0	12	100	20	21	300000	30000
UB4	0	12	100	1	2	150000	15000
UB5	0	12	100	5	6	500000	50000
UB6	0	12	100	17	18	800000	80000

4.3.3.1.2 Data Center Configuration

One Data Center is used with 50 Virtual Machines and 5 Physical HW Units. The five hosts are heterogeneous (have different number of processors and different processors speed). Table 4.12 present the Data Center configurations while Table 4.13 shows the configuration of every Physical Hardware in the Data Center.

Table 4.12 Data Center Configurations for Experiment 3

Data Center	#VMs	Image Size	Memory	BW
DC1	50	10000	1024	1000

Table 4.13 Physical Hardware Configurations for DC1 for Experiment 3

ID	Memory (Mb)	Storage (Mb)	Available BW	Number of Processors	Processor Speed	VM Policy
0	204800	100000000	1000000	4	200	TIME_SHARED
1	204800	100000000	1000000	5	500	TIME_SHARED
2	204800	100000000	1000000	2	2000	TIME_SHARED
3	204800	100000000	1000000	2	5000	TIME_SHARED
4	204800	100000000	1000000	2	9000	TIME_SHARED

4.3.3.2 Results

The results of the experiments are as follows:

a. Result 1 High Burst:

4.3.3.2.1 Data Center Hourly Loading

As shown in Figure 4.20 there are six peaks on the curve. Every peak takes one hour. During this hour an intensive number of requests are received by the Data Center. At the end of the peak hour, the workload returns to normal.

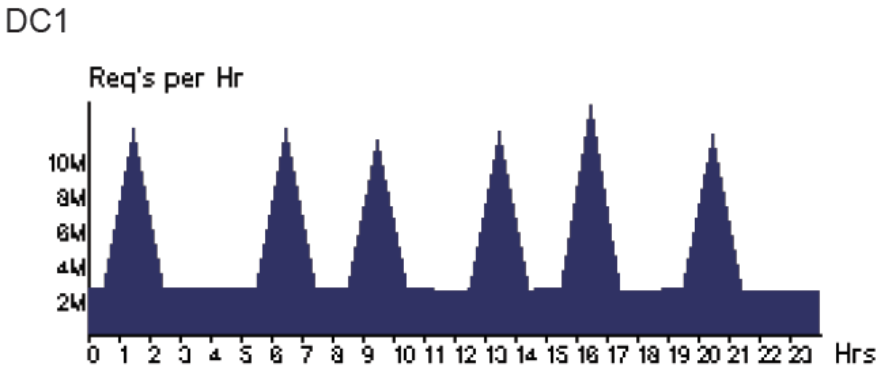


Figure 4.20 Data Center Hourly Loading for High Burst Workload (Config.1)

4.3.3.2.2 Response Time

The results obtained from applying Configuration1 are clarified in Figure 4.21. From this chart we can see that Random algorithm recorded the worst response time while Round Robin and ESCE has better records.

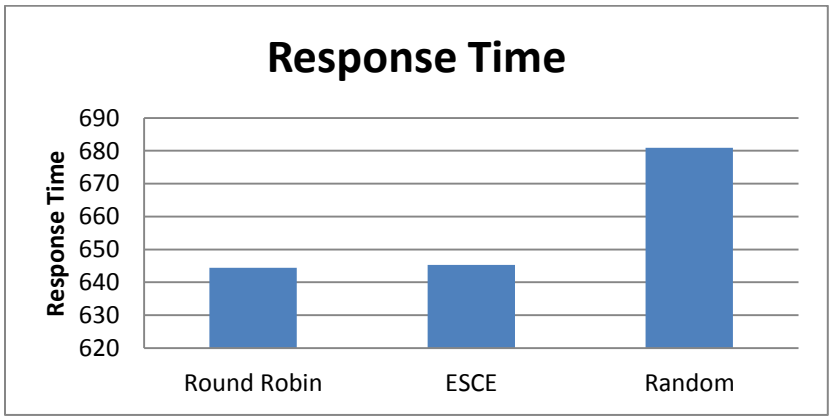


Figure 4.21 Response Time Chart for Burst Workload (Config.1)

4.3.3.2.3 Processing Time

In this experiment, Round Robin has the best processing time while Random has the worst one. Also we can notice that ESCE processing time is very close to the response time of the Round Robin algorithm. Figure 4.22 presents the processing time results for this experiment

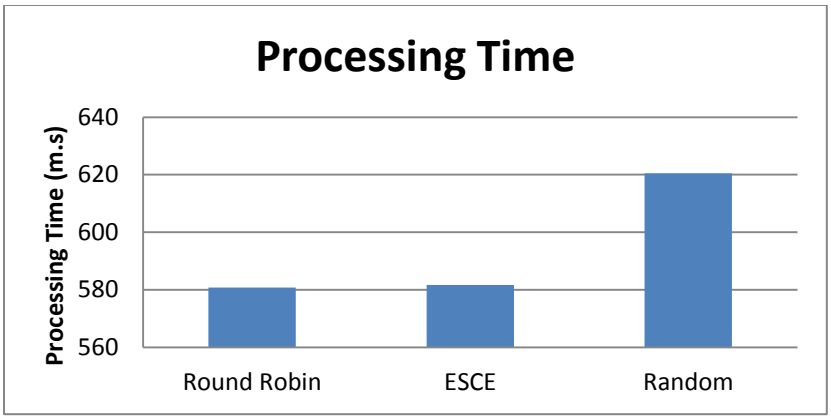


Figure 4.22 Processing Time Chart for Burst Workload (Config.1)

b. Result 2:

4.3.3.2.4 Data Center Hourly Loading

From the graph below we can see that there are six peaks on the curve with different requests rate. Every peak takes one hour as shown in Figure 4.23.

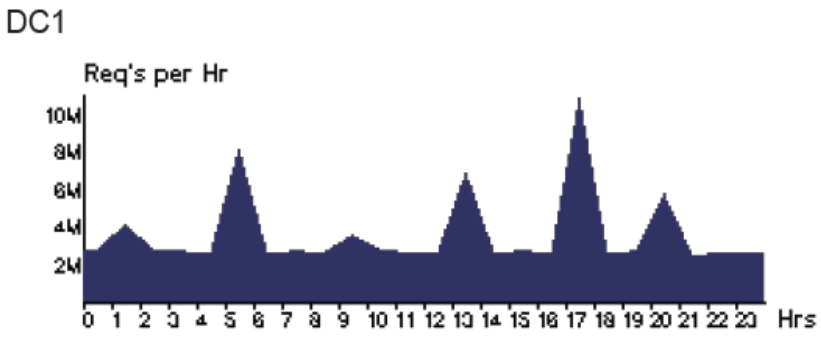


Figure 4.23 Data Center Hourly Loading for High Burst Workload (Config.2)

4.3.3.2.5 Response Time

It is apparent from the chart in Figure 4.24 that the three algorithms have very close response time results. The difference between them does not exceed one hundred parts.

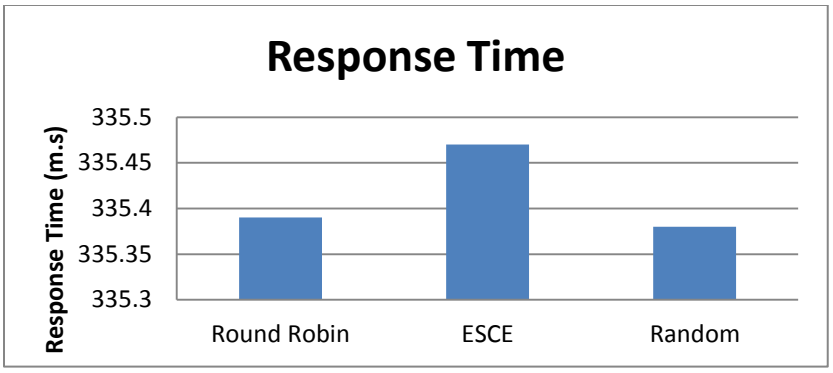


Figure 4.24 Response Time Chart for Burst Workload (Config.2)

4.3.3.2.6 Processing Time

The same as response time results, the processing time for the three algorithms is very close and Random has the best record. Figure 4.25 presents the processing time results.

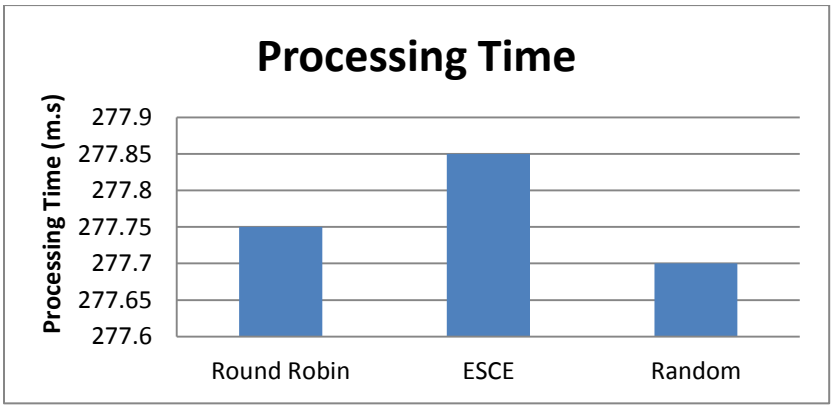


Figure 4.25 Processing Time Chart for Burst Workload (Config.2)

4.3.3.3 Discussion

High and medium burst have been studied in this experiment. The results show that Round Robin has the best Response Time and processing time under high burst. However under medium burst all the three algorithms have almost equal response time and processing time. Figure 4.26 and Figure 4.27 summarize the response time and processing time results for both configurations.

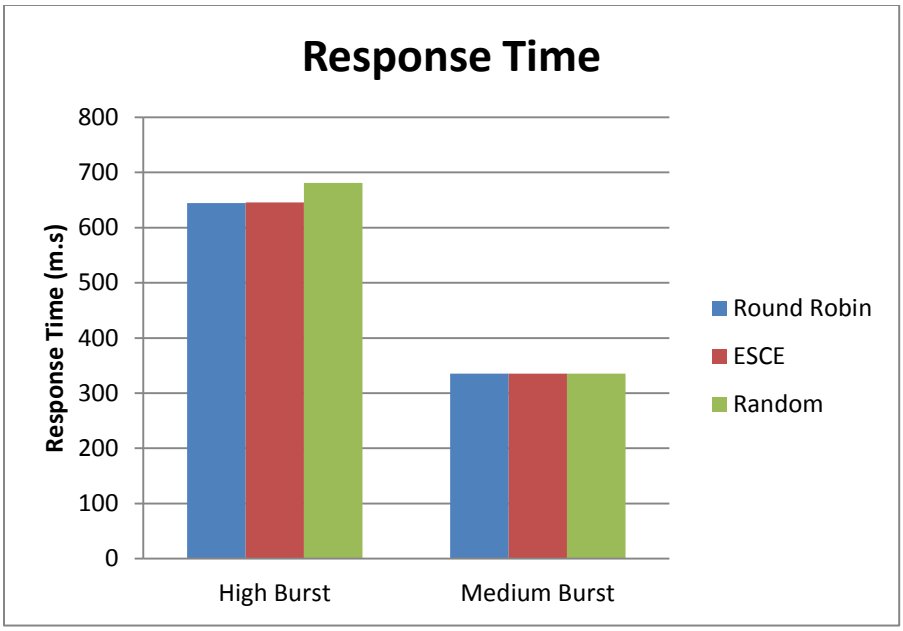


Figure 4.26 Response Time Chart for Burst Workload

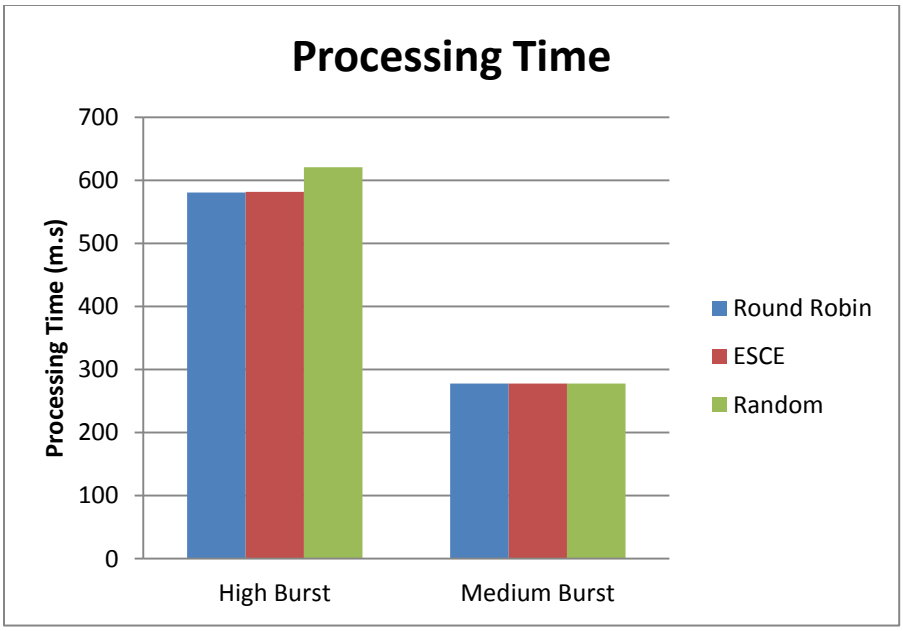


Figure 4.27 Processing Time Chart for Burst Workload

4.3.4 Results Discussion Part I

Comparing the results of experiments 1, 2 and 3, it can be noticed that the burst affect the performance of the load balancing algorithms. In normal workload Random came in second place and its performance results were close to the RR results which came in the first place. On the other hand, under high burst the performance of Random algorithm degraded badly and made it came in the third place with large difference between its performance and the better algorithms' performance. In contrast, ESCE algorithm has the worst response time in normal workload but its performance become better in burst.

4.4 Experiments Part II

This part of experiments is done in order to decide the two algorithms which will be used in our proposed algorithm for burst and non-burst cases. In burst cases the workload is high otherwise the workload is low. So we need to make experiments to find which one of the three algorithms (RR, ESCE, and Random) has the best performance when the workload is high and which one has the best performance when the workload is low.

The data center is configured with heterogeneous hosts. One Data Center is used with 50 Virtual Machines and 5 Physical HW Units. Table 4.14 and Table 4.15 illustrate in details the configuration of the Data Center and its Physical Hardware Hosts.

Table 4.14 Data Center Configurations for Experiment 4

Data Center	#VMs	Image Size	Memory	BW
DC1	50	10000	1024	1000

Table 4.15 Physical Hardware Configurations for DC1 for Experiment 4

ID	Memory (Mb)	Storage (Mb)	Available BW	Number of Processors	Processor Speed	VM Policy
0	204800	100000000	1000000	4	2000	TIME_SHARED
1	204800	100000000	1000000	5	5000	TIME_SHARED
2	204800	100000000	1000000	2	9000	TIME_SHARED
3	204800	100000000	1000000	2	1000	TIME_SHARED
4	204800	100000000	1000000	2	15000	TIME_SHARED

4.4.1 Experiment 4: High Workload

The aim of this experiment is to find which of the following algorithms: Round Robin, Equally Spread Current Execution (ESCE), and Random Algorithm have the best response time in high workload. The best algorithm will be used in our proposed algorithm in burst case.

4.4.1.1 Configurations

4.4.1.1.1 User Base Configuration

The User Base in this experiment should be configured to generate high load. To achieve this goal, six User Bases are used located in the same region. Every User Base has two peak hours, and average of users in peak period is equal to off peak period. Table 4.16 presents User Base configuration.

Table 4.16 User Base Configurations for Experiment 4

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	15	400000	400000
UB2	0	12	100	15	17	100000	100000
UB3	0	12	100	20	22	300000	300000
UB4	0	12	100	1	3	150000	150000
UB5	0	12	100	21	23	500000	500000
UB6	0	12	100	9	11	800000	800000

4.4.1.2 Results

4.4.1.2.1 Data Center Hourly Loading

From the graph in Figure 4.28, it can be seen that requests exceeded 20,000,000 requests per hour. This put the data center under high workload.

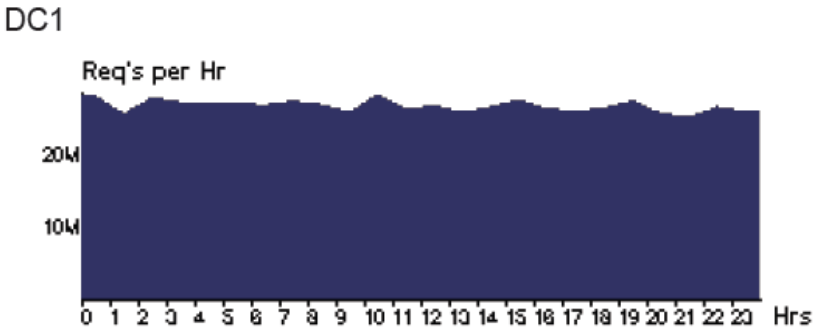


Figure 4.28 Data Center Hourly Loading for High Workload

4.4.1.2.2 Response Time

Simulation results shows that Round Robin has the best response time under high workload while the Random has the worst one. Figure 4.29 clarify the different in response time for Round Robin, ESCE, and Random algorithm.

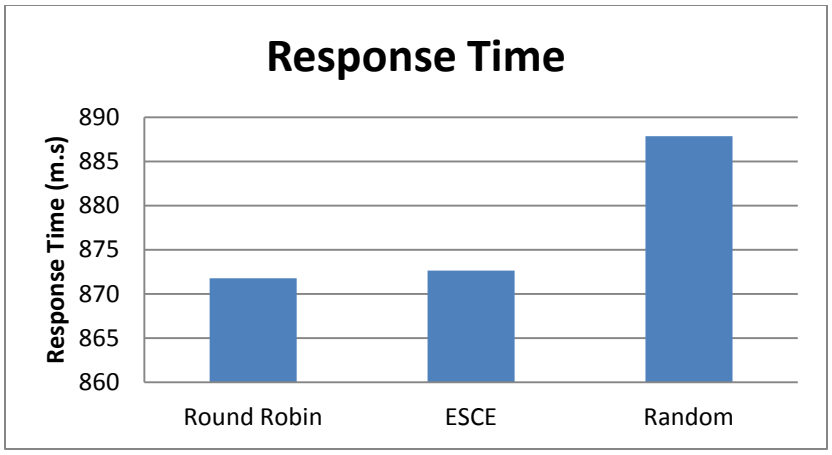


Figure 4.29 Response Time Chart for High Workload

4.4.1.2.3 Processing Time

The processing time results were as shown in the chart in Figure 4.30. The Random algorithm has the worst processing time while the Round Robin has the best one.

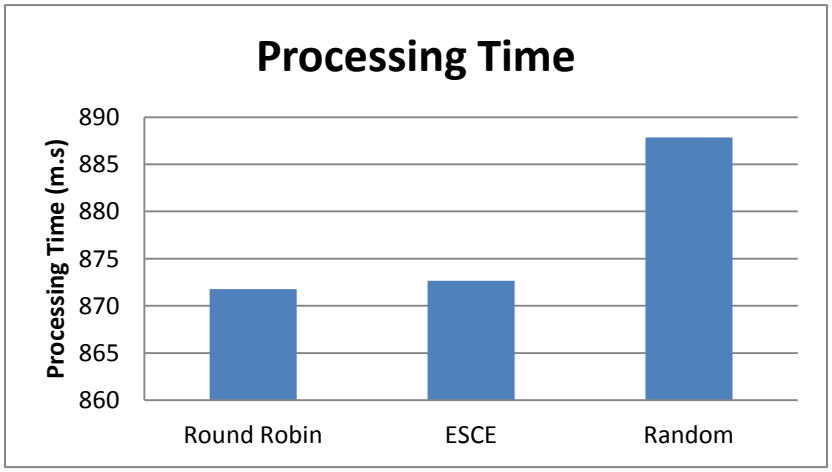


Figure 4.30 Processing Time Chart for High Workload

4.4.1.3 Discussion

In this experiment we try to put the Data Center under high load to study the performance of Load Balancing algorithms in such case. It is apparent from the results that the response time and the processing time of the Round Robin algorithm is the best one under high workload. According to these results the Round Robin will be used in our proposed algorithm.

4.4.2 Experiment 5: Low Workload

The main objective of this experiment is to find which of the three load balancing algorithms: Round Robin, ESCE, and Random algorithm have the best performance in low workload. The best one will be used in our proposed algorithm in non-burst case.

4.4.2.1 Configurations

4.4.2.1.1 User Base Configuration

In this experiment we need to put the Data center under low workload. For that purpose we use the same user base configuration in the high workload experiment but the average users is less than the average users in the high load experiment by about 1/10. User base configuration is shown in Table 4.17.

Table 4.17 User Base Configurations for Experiment 5

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	15	40000	40000
UB2	0	12	100	15	17	10000	10000
UB3	0	12	100	20	22	30000	30000
UB4	0	12	100	1	3	15000	15000
UB5	0	12	100	21	23	50000	50000
UB6	0	12	100	9	11	80000	80000

4.4.2.2 Results

4.4.2.2.1 Data Center Hourly Loading

Figure 4.31 shows the hourly loading on the data center. The number of requests is about 2,000,000 requests per hour

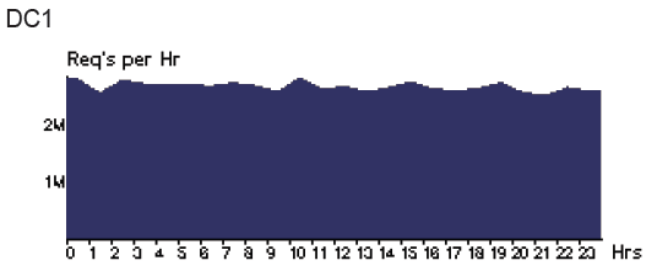


Figure 4.31 Data Center Hourly Loading for Low Workload

4.4.2.2.2 Response Time

Simulation results shows that Random has the best response time under low workload compared with RR an ESCE. The Figure 4.32 presents the response time for the three algorithms.

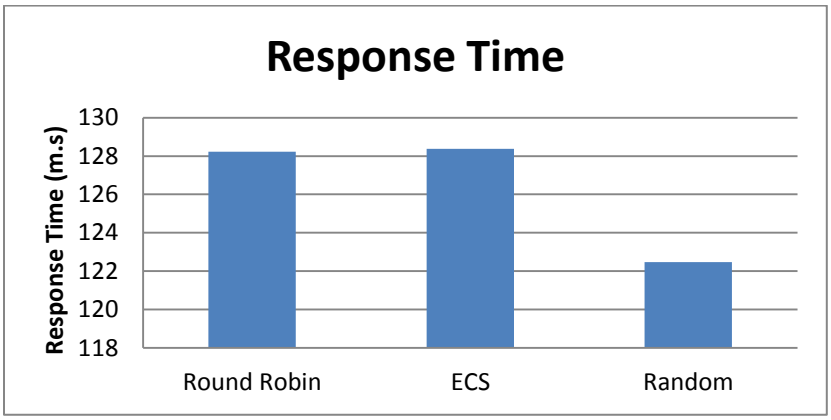


Figure 4.32 Response Time Chart for Low Workload

4.4.2.2.3 Processing Time

Under low workload, Random has the best processing time. Round Robin and ESCE has very close processing times but ESCE is the worst. Figure 4.33 explains the processing time results for this experiment.

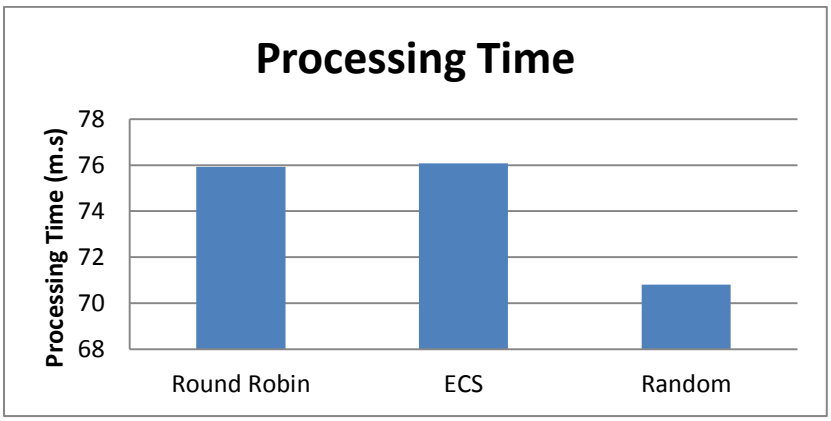


Figure 4.33 Processing Time Chart for Low Workload

4.4.2.3 Discussion

In this experiment we aimed to find the best performance algorithm under low workload. The simulation results show that Random algorithm works efficiently in low workload more than the other two algorithms. This leads as to select it to be used in our proposed algorithm.

4.4.3 Results Discussion Part II

Based on the results of this part, we decided to use Round Robin for burst cases as it has the best performance when the load is high and Random Algorithm for non-burst cases because it recorded the best performance when the load is low.

4.5 Experiments Part III

In this part the Adaptive Algorithm is going to be tested and evaluated by comparing its performance with other load balancing algorithms and under different levels of burstness.

For this part of experiments, one Data Center is used with 50 Virtual Machines and 5 Physical HW Units. The five hosts are heterogeneous. Table 4.18 and Table 2.19 illustrate in details the configuration of the Data Center and its Physical Hardware Hosts

Table 4.18 Data Center Configuration for Experiment 6

Data Center	#VMs	Image Size	Memory	BW
DC1	50	10000	1024	1000

Table 4.19 Physical Hardware Configurations for Experiment 6

ID	Memory (Mb)	Storage (Mb)	Available BW	Number of Processors	Processor Speed	VM Policy
0	204800	100000000	1000000	4	2000	TIME_SHARED
1	204800	100000000	1000000	5	5000	TIME_SHARED
2	204800	100000000	1000000	2	9000	TIME_SHARED
3	204800	100000000	1000000	2	1000	TIME_SHARED
4	204800	100000000	1000000	2	15000	TIME_SHARED

4.5.1 Experiment 6: Adaptive Algorithm without Fuzzifier

In this experiment we will test the performance of the proposed algorithm before using the fuzzifier under bursty workload. The response time and the processing time of the adaptive algorithm will be compared with response time and the processing time of the Round Robin, ESCE, and Random Algorithms.

4.5.1.1 Configurations

4.5.1.1.1 User Base Configuration

The User Base in this experiment should be configured to generate bursty workload. We use the same configuration in Experiment 3. As presented in table below there will be 6 User Bases and every user base has one peak hour so intensive requests will be sent to the data center during this hour, making a burst. The average number of users in peak is equal to the average number of users in off peak * 10. Table 4.20 shows User Base configuration.

Table 4.20 User Base Configuration for Experiment 6

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	14	400000	40000
UB2	0	12	100	9	10	100000	10000
UB3	0	12	100	20	21	300000	30000
UB4	0	12	100	1	2	150000	15000
UB5	0	12	100	5	6	500000	50000
UB6	0	12	100	17	18	800000	80000

4.5.1.2 Results

4.5.1.2.1 Data Center Hourly Loading

Figure 4.34 shows the hourly loading on the data center in this experiment.

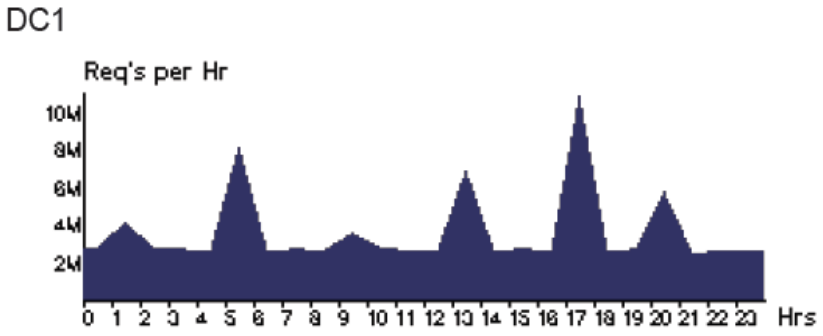


Figure 4.34 Data Center Hourly Loading for Adaptive Algorithm Experment Befor Using Fuzzifier

4.5.1.2.2 Response Time

The results shows that Adaptive algorithm improve the performance of the cloud system as it recorded the best response in this experiment. The chart in Figure 4.35 shows the improvement of adaptive algorithm on response time over the other algorithms

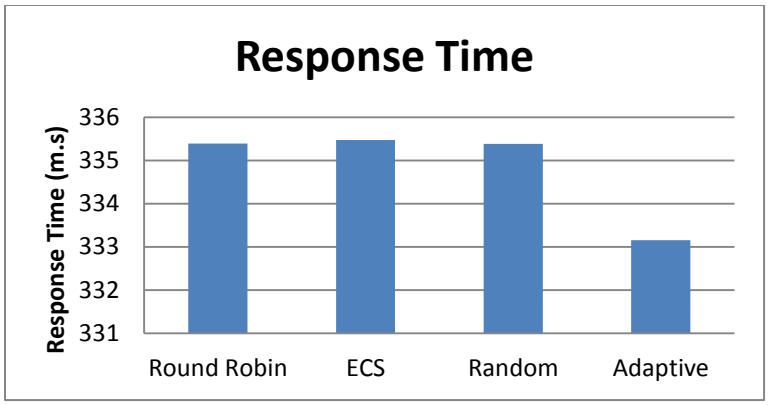


Figure 4.35 Response Time Chart for Adaptive Algorithm Experment Befor Using Fuzzifier

4.5.1.2.3 Processing Time

In addition to improving the response time, Adaptive algorithm also improves the processing time. The processing time results are presented in Figure 4.36.

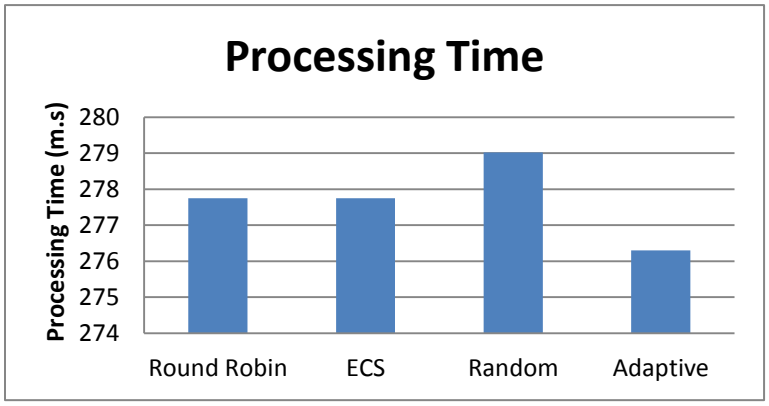


Figure 4.36 Processing Time Chart for Adaptive Algorithm Experment Befor Using Fuzzifier

4.5.1.3 Discussion

In this experiment our proposed algorithm is tested under bursty workload and compared with other algorithms. The simulation results showed that our algorithm recorded the best response time and processing time compared with other algorithms.

4.5.2 Experiment 7: Adaptive Algorithm using Fuzzifier

In this experiment we will test the performance of the proposed algorithm using the Fuzzifier. The response time and the processing time of the adaptive algorithm will be compared with response time and the processing time of the RR and Random Algorithms.

4.5.2.1 Configurations

4.5.2.1.1 User Base Configuration

We use the same configuration in Experiment 6. Table 4.21 shows User Base configuration.

Table 4.21 User Base Configuration for Experiment 7

Name	Region	Requests per user per Hr.	Data Size per req.(Bytes)	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB1	0	12	100	13	14	400000	40000
UB2	0	12	100	9	10	100000	10000
UB3	0	12	100	20	21	300000	30000
UB4	0	12	100	1	2	150000	15000
UB5	0	12	100	5	6	500000	50000
UB6	0	12	100	17	18	800000	80000

4.5.2.2 Results

4.5.2.2.1 Data Center Hourly Loading

Figure 4.37 shows the hourly loading on the data center in this experiment.

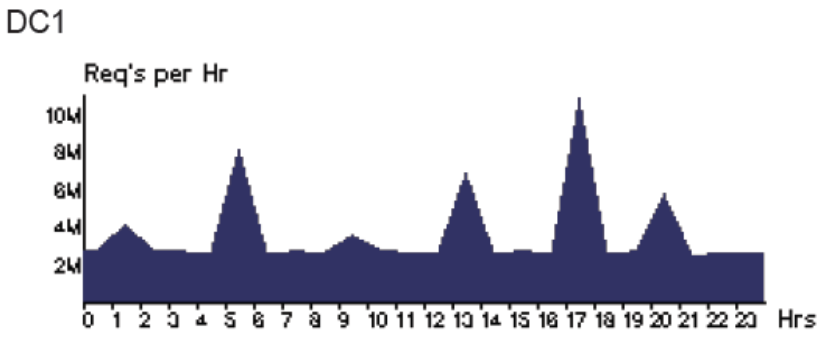


Figure 4.37 Data Center Hourly Loading for Adaptive Algorithm Experment Using Fuzzifier

4.5.2.2.2 Response Time

The results show that using fuzzifier improves the performance of Adaptive algorithm as the response time become better. The chart in Figure 4.38 shows the improvement of adaptive algorithm using fuzzifier on response time over the other algorithms

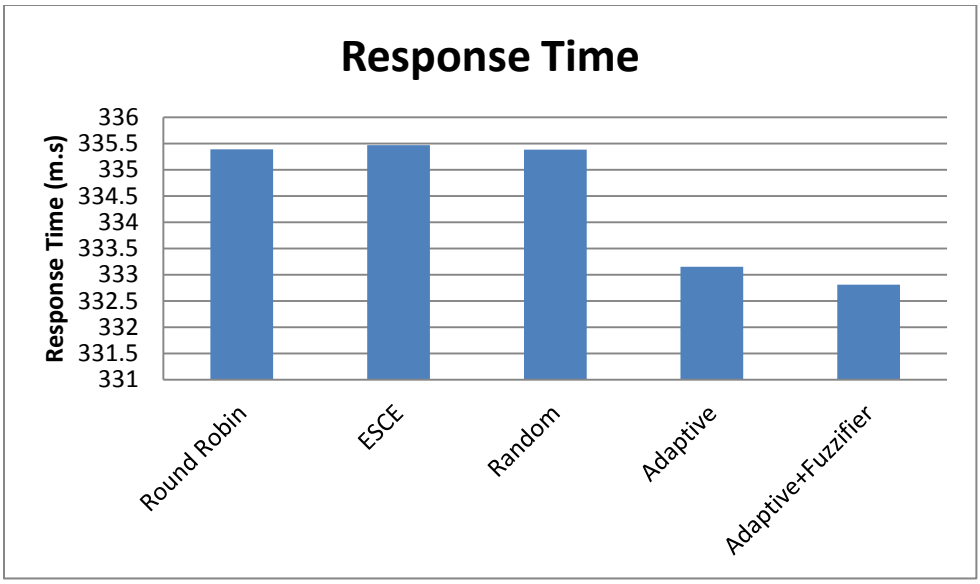


Figure 4.38 Response Time Chart for Adaptive Algorithm Experiment Using Fuzzifier

4.5.2.2.3 Processing Time

Using fuzzifier improves the processing time of the Adaptive algorithm. As we can see in Figure 4.39 the Adaptive algorithm using fuzzifier recorded the best response time.

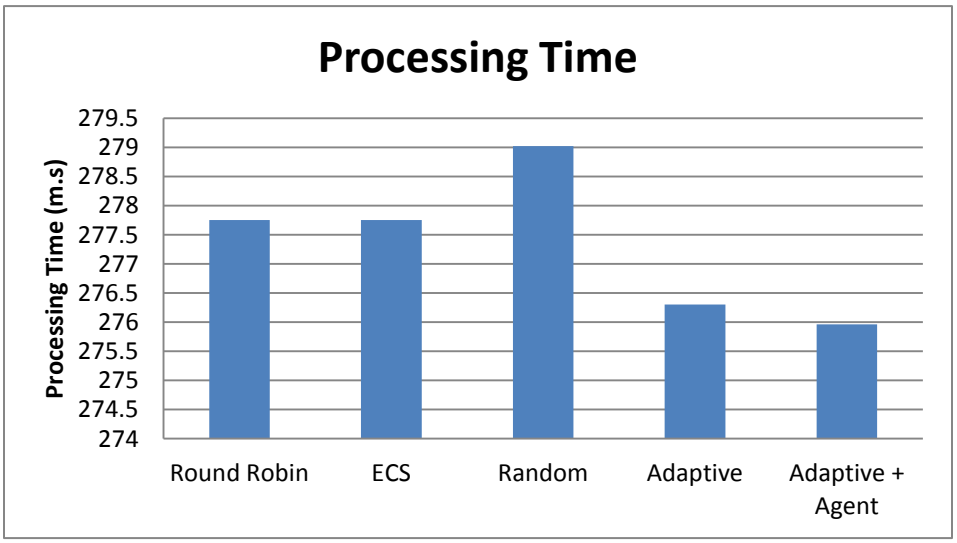


Figure 4.39 Processing Time Chart for Adaptive Algorithm Experiment Using Fuzzifier

4.5.2.3 Discussion

In this experiment we add a fuzzifier in our proposed algorithm. Again we tested the adaptive algorithm under burst workload and compared it with other algorithms. The simulation results showed that using fuzzifier improves the performance of Adaptive algorithm.

4.5.3 Experiment 8:

In this experiment we will test the performance of the Adaptive algorithm in different burst levels. In addition we will study the effect of the instruction length on the performance of Adaptive algorithm by comparing the response time and the processing time with RR and Random Algorithms.

4.5.3.1 Configurations

4.5.3.1.1 User Base Configuration

In order to test the performance of Adaptive algorithm in different burst cases, ten different workloads should be configured and tested separately. This can be achieved by changing: num. of UBs, Peak Hours, Avg. Peak Users, and Avg. Off Peak Users. Tables form 4.22 to 4.26 show configurations for ten different Bursty Workloads.

Table 4.22 Bursty Workload for Experiment 1 and 2

	Experiment 1				Experiment 2			
	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB 1	13	14	400000	40000	1	2	400000	40000
UB 2	9	10	200000	20000	4	5	200000	20000
UB 3	20	21	300000	30000	6	7	300000	30000
UB 4	1	2	200000	20000	8	9	200000	20000
UB 5	5	6	500000	50000	11	12	500000	50000
UB 6	17	18	800000	80000	17	18	800000	80000

Table 4.23 Bursty Workload for Experiment 3 and 4

	Experiment 3				Experiment 4			
	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB 1	1	2	100000	10000	3	4	100000	10000
UB 2	4	5	200000	20000	6	7	200000	20000
UB 3	6	7	700000	70000	9	10	700000	70000
UB 4	9	10	400000	40000	15	16	400000	40000
UB 5	13	14	500000	50000	18	19	500000	50000
UB 6	20	21	800000	80000	21	22	800000	80000

Table 4.24 Bursty Workload for Experiment 5 and 6

	Experiment 5				Experiment 6			
	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB 1	1	2	100000	10000	1	2	300000	30000
UB 2	4	5	200000	20000	4	5	200000	20000
UB 3	6	7	150000	15000	6	7	500000	50000
UB 4	9	10	250000	25000	9	10	400000	40000
UB 5	13	14	100000	10000	13	14	700000	70000
UB 6	20	21	200000	20000	20	21	500000	50000

Table 4.25 Bursty Workload for Experiment 7 and 8

	Experiment 7				Experiment 8			
	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB 1	13	14	400000	4000	13	14	800000	40000
UB 2	6	7	200000	2000	9	10	800000	10000
UB 3	20	21	300000	3000	20	21	800000	30000
UB 4	1	2	150000	1500	1	2	800000	15000
UB 5	15	16	500000	5000	5	6	800000	50000
UB 6	20	21	800000	8000	17	18	800000	80000

Table 4.26 Bursty Workload for Experiment 9 and 10

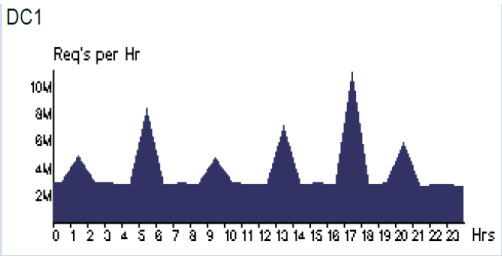
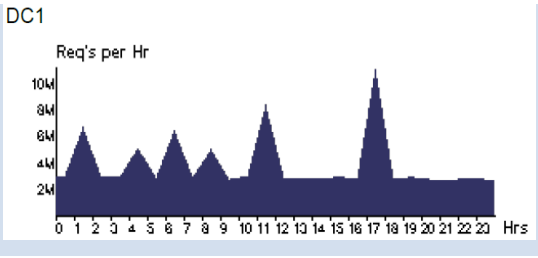
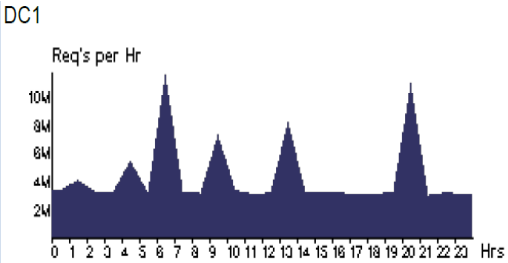
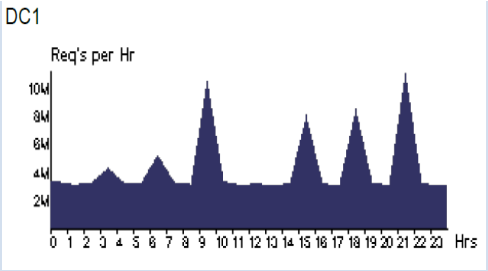
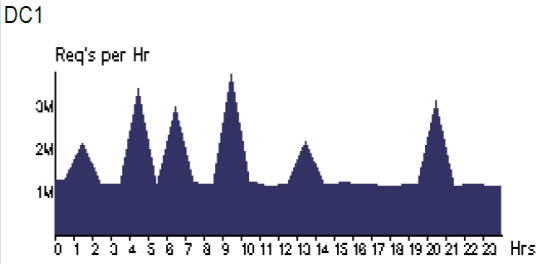
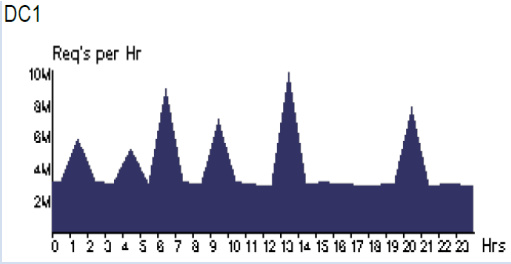
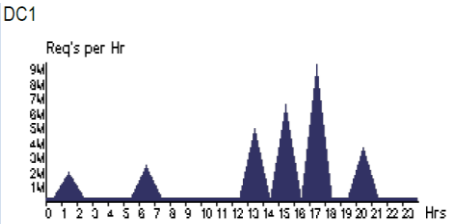
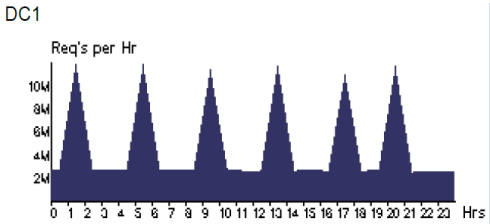
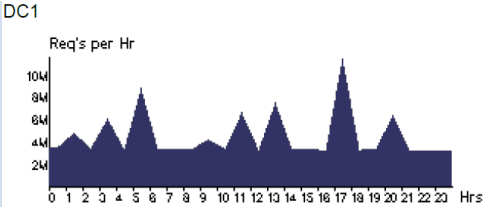
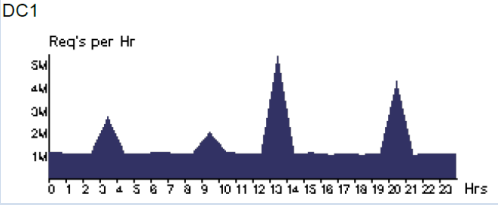
	Experiment 9				Experiment 10			
	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users	Peak Hours Start(GMT)	Peak Hours End(GMT)	Avg. Peak Users	Avg. Off-peak Users
UB 1	13	14	400000	40000	13	14	400000	40000
UB 2	9	10	100000	20000	9	10	100000	10000
UB 3	20	21	300000	30000	20	21	300000	30000
UB 4	1	2	150000	15000	3	4	150000	15000
UB 5	5	6	500000	50000	-	-	-	-
UB 6	17	18	800000	80000	-	-	-	-
UB 7	11	12	300000	30000	-	-	-	-
UB 8	3	4	250000	25000	-	-	-	-

4.5.3.2 Results

4.5.3.2.1 Data Center Hourly Loading

Table 4.27 shows the hourly loading on the data center for the ten experiments.

Table 4.27 Data Center Hourly Loading for the Ten Experiments

Experiment 1	Experiment 2
	
Experiment 3	**Experiment 4**
	
Experiment 5	**Experiment 6**
	
Experiment 7	**Experiment 8**
	
Experiment 9	**Experiment 10**
	

4.5.3.2.2 Response Time

The results show that Adaptive algorithm has the best response time in almost all the ten burst workload except Experiment 7 and 8. Tables from 4.28 to 4.32 illustrate a comparison between RR, ESCE, Random, and Adaptive algorithm for the ten workloads when the instruction length is 250,500 and 100 Byte.

Table 4.28 Response Time Results for Experiment 1 and 2

	Experiment 1			Experiment 2		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	328.88	616.43	1192.74	327.9	614.59	1189.18
ESCE	328.98	615.13	1193.31	328.01	615.1	1191.16
Random	329.12	616.91	1192.21	327.82	614.19	1190.74
Adaptive1800000	326.69	621.05	1183.51	325.33	608.36	1182.36
Adaptive1900000	326.36	621.56	1182.05	325.26	608.6	1182.75
Adaptive2000000	326.84	621.05	1181.68	325.72	608.56	1183.37

Table 4.29 Response Time Results for Experiment 3 and 4

	Experiment 3			Experiment 4		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	374.64	708.78	1375.91	367.52	694.38	1348.31
ESCE	374.84	709.04	1378.87	367.62	694.86	1351.02
Random	376.01	712.22	1382.93	368.28	699.3	1356.25
Adaptive1800000	372.41	710.66	1369.2	365.55	689.88	1346.98
Adaptive1900000	373.03	706.19	1368.61	364.76	689.33	1346.48
Adaptive2000000	373.29	707.79	1369.85	365.52	689.91	1347

Table 4.30 Response Time Results for Experiment 5 and 6

	Experiment 5			Experiment 6		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	149.28	254.32	463.51	316.69	592.01	1142.51
ESCE	149.6	254.81	464.46	316.8	592.5	1144
Random	146.08	247.12	447.85	315.91	589.71	1146.48
Adaptive1800000	146.21	247.88	447.08	317.95	588.84	1137.3
Adaptive1900000	146.3	247.48	250.63	314.46	593.55	1131.89
Adaptive2000000	146.2	246.49	448.83	314.67	593.97	1134.09

Table 4.31 Response Time Results for Experiment 7 and 8

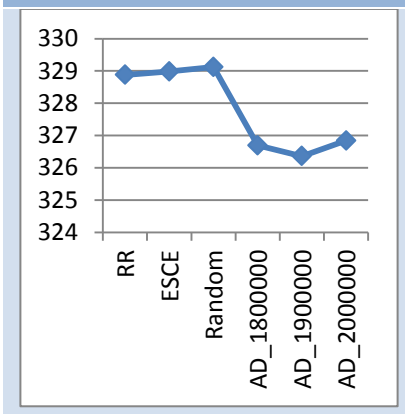
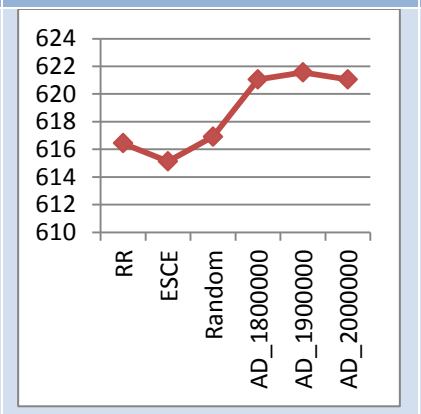
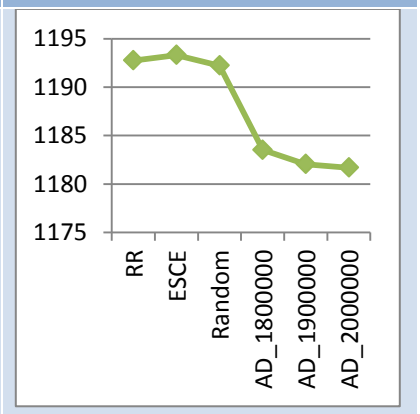
	Experiment 7			Experiment 8		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	652.04	1264.15	2489.25	660.92	1284.56	2535.39
ESCE	652.17	1264.57	2489.94	661.02	1285.39	2536.09
Random	663.23	1284.32	2539.46	673.52	1312.76	2597.73
Adaptive1800000	655.08	1270.26	2518.17	661.89	1288.09	2567.23
Adaptive1900000	653.54	1269.24	2513.49	661.87	1288.68	2569.41
Adaptive2000000	654.52	1267.28	2514.3	663.55	1289.43	2568.16

Table 4.32 Response Time Results for Experiment 9 and 10

	Experiment 9			Experiment 10		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	311.26	580.86	1120.76	214.38	385.8	727.28
ESCE	311.36	581.31	1121.23	214.5	385.98	727.59
Random	311.53	579.84	1120.6	210.35	378.94	715.53
Adaptive1800000	312.14	583.27	1127.34	210.7	377.99	711.22
Adaptive1900000	309.13	576.65	1117.1	211.7	379.23	712.81
Adaptive2000000	309.63	584.77	1116.57	211.23	377.12	714.4

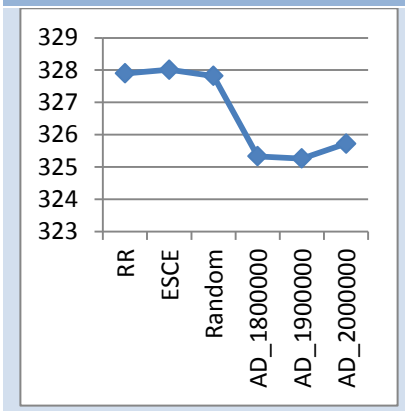
The following charts in Table 4.33 clarify the variations of the performance for the Adaptive algorithm and other algorithms.

Table 4.33 Responce Time Charts for the Ten Experments

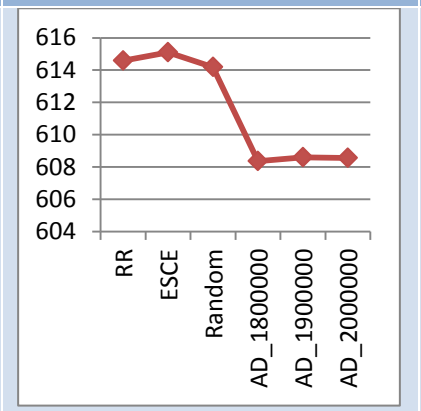
Experiment 1		
Inst. 250	Inst. 500	Inst. 1000
		

Experiment 2

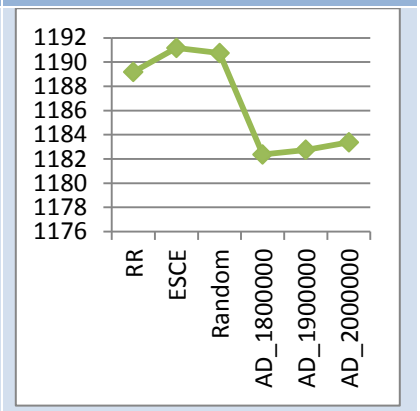
Inst. 250



Inst. 500

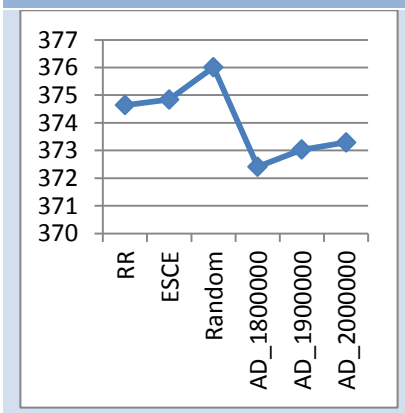


Inst. 1000

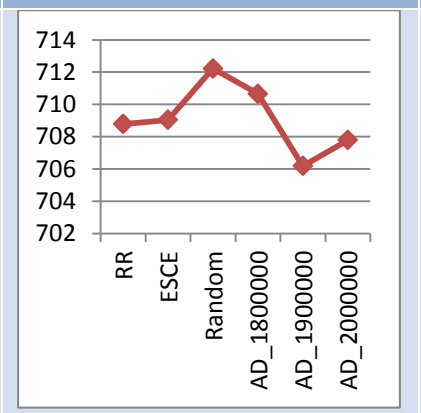


Experiment 3

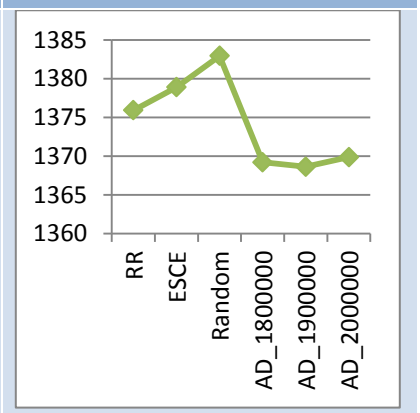
Inst. 250



Inst. 500

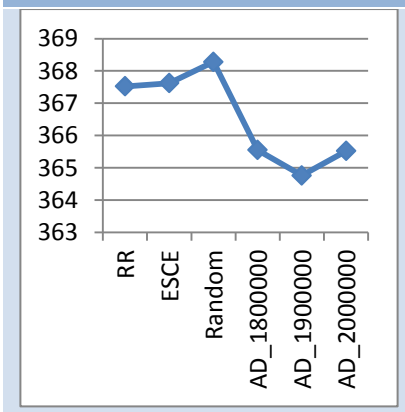


Inst. 1000

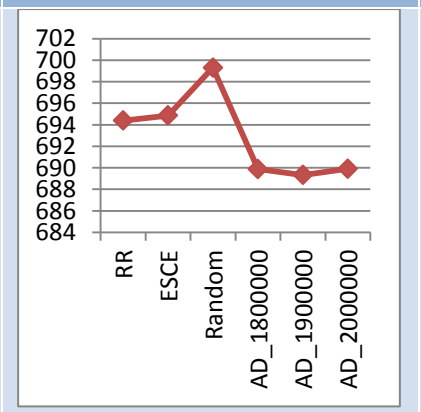


Experiment 4

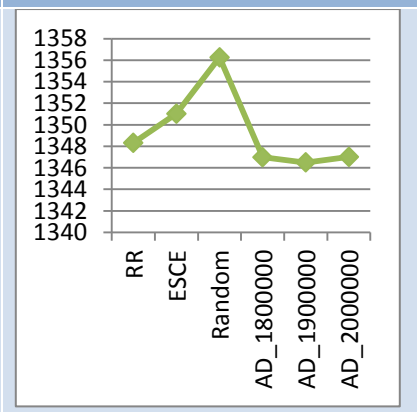
Inst. 250



Inst. 500

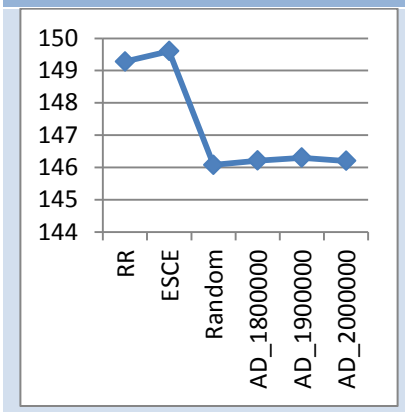


Inst. 1000

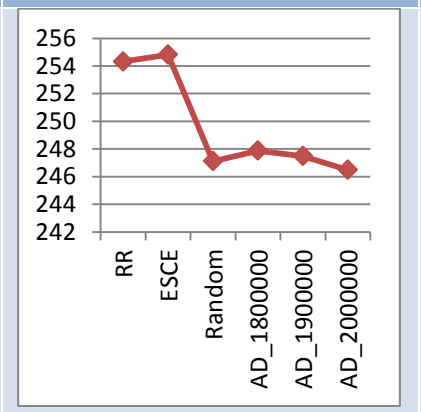


Experiment 5

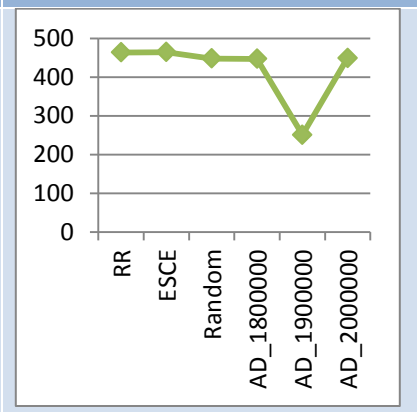
Inst. 250



Inst. 500

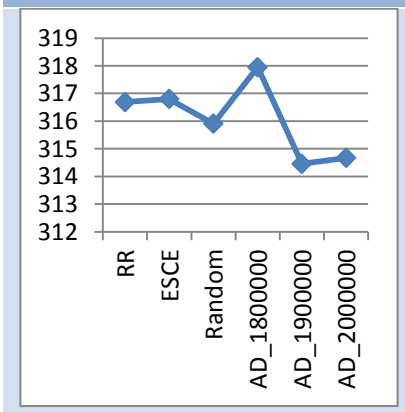


Inst. 1000

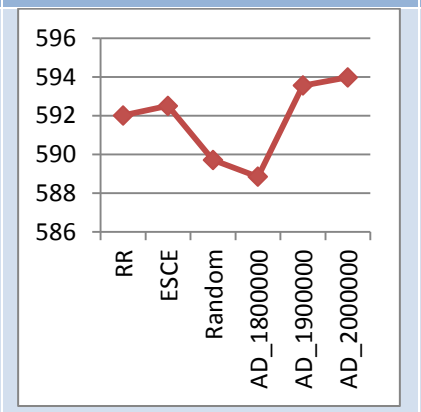


Experiment 6

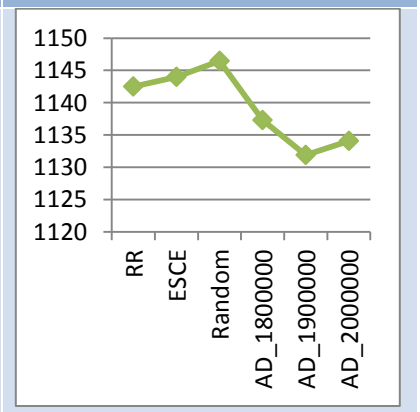
Inst. 250



Inst. 500

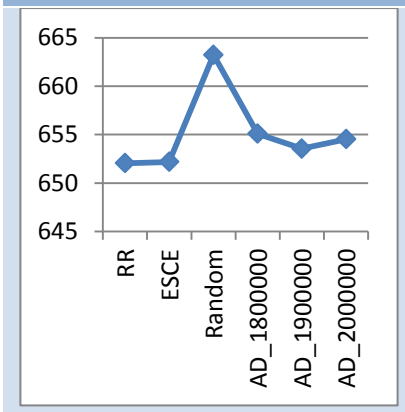


Inst. 1000

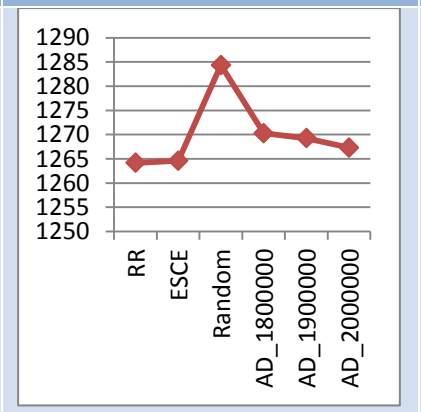


Experiment 7

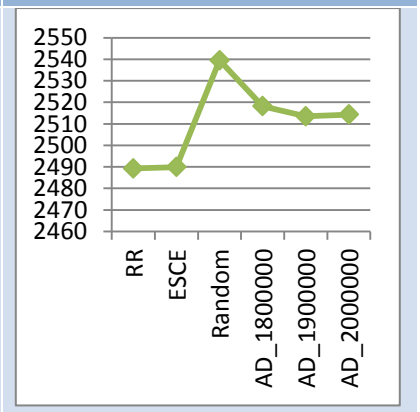
Inst. 250



Inst. 500

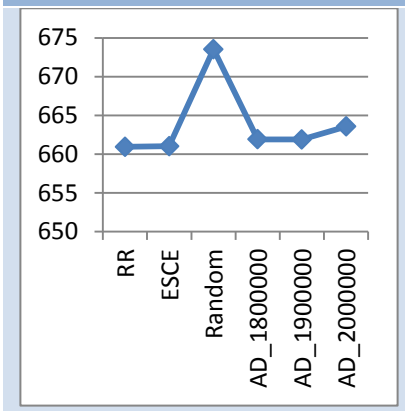


Inst. 1000

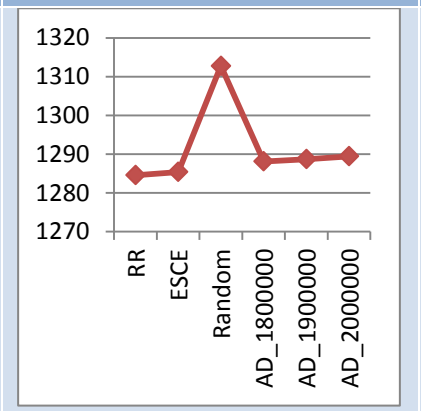


Experiment 8

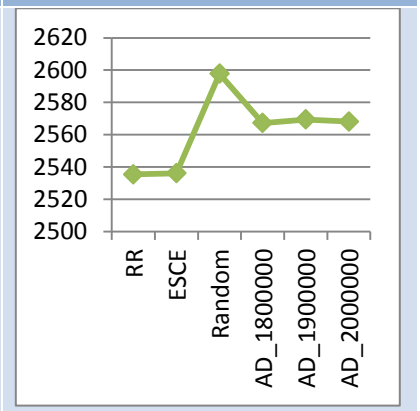
Inst. 250



Inst. 500

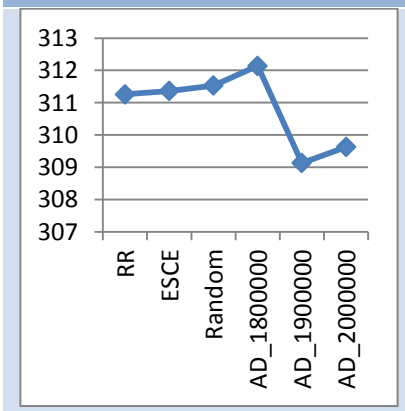


Inst. 1000

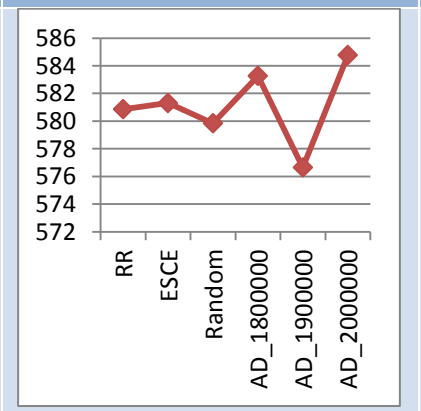


Experiment 9

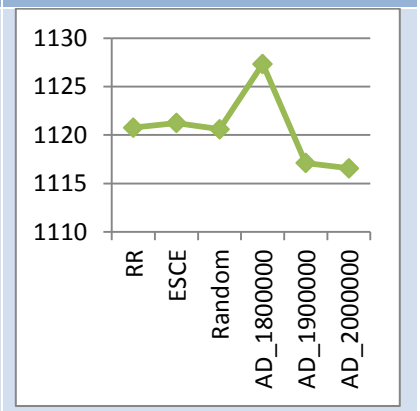
Inst. 250



Inst. 500

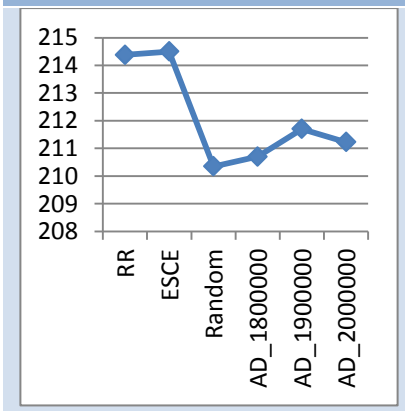


Inst. 1000

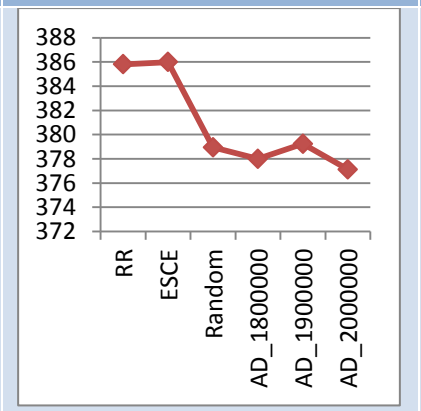


Experiment 10

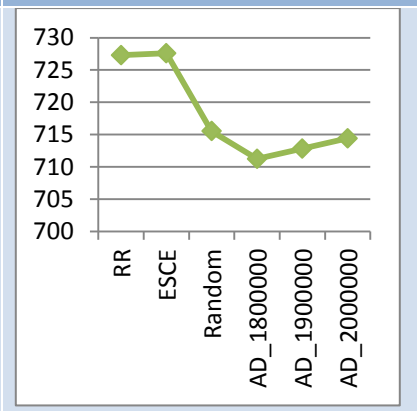
Inst. 250



Inst. 500



Inst. 1000



4.5.3.2.3 Processing Time

As we can see in Tables from 4.34 to 4.38 the Adaptive algorithm improved the processing time under bursty workload.

Table 4.34 Processing Time Results for Experiment 1 and 2

	Experiment 1			Experiment 2		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	271.67	558.88	1134.73	270.69	557.04	1131.17
ESCE	271.78	559.64	1135.29	270.81	557.56	1133.16
Random	273.85	559.36	1135.89	272.60	558.71	1134.48
Adaptive1800000	270.19	564.39	1126.10	268.81	551.6	1125.00
Adaptive1900000	269.86	564.88	1124.64	268.75	551.83	1125.37
Adaptive2000000	270.33	564.58	1124.29	269.19	551.79	1125.99

Table 4.35 Processing Time Results for Experiment 3 and 4

	Experiment 3			Experiment 4		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	316.41	650.16	1316.68	309.42	635.89	1289.13
ESCE	316.62	650.42	1319.63	309.53	636.38	1291.83
Random	319.91	655.84	1325.62	312.31	643.09	1299.15
Adaptive1800000	314.91	652.94	1310.65	308.28	632.26	1288.55
Adaptive1900000	315.67	648.55	1310.16	307.49	631.73	1288.06
Adaptive2000000	315.94	650.15	1311.42	308.23	632.33	1288.58

Table 4.36 Processing Time Results for Experiment 5 and 6

	Experiment 5			Experiment 6		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	96.51	201.42	410.55	259.80	535.09	1085.52
ESCE	96.84	201.92	411.51	259.93	535.58	1087.01
Random	94.15	195.19	395.62	261.04	534.84	1091.23
Adaptive1800000	94.03	195.71	394.68	261.96	532.75	1081.03
Adaptive1900000	94.32	195.49	398.34	258.34	537.53	1075.60
Adaptive2000000	94.24	194.56	396.59	258.56	537.96	1077.79

Table 4.37 Processing Time Results for Experiment 7 and 8

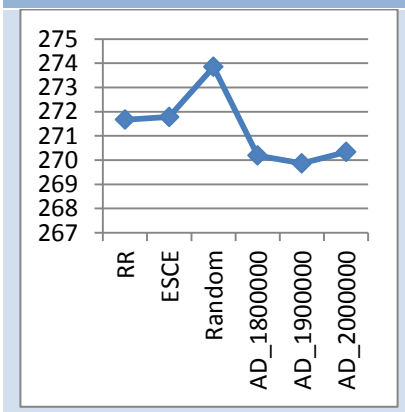
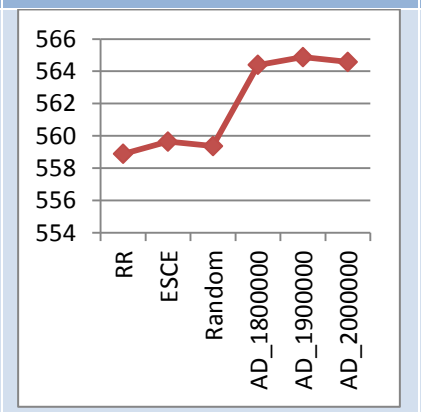
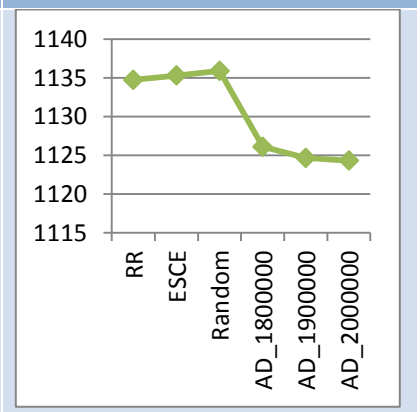
	Experiment 7			Experiment 8		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	584.43	1193.23	2411.96	597.82	1222.01	2473.83
ESCE	584.58	1193.68	2412.67	597.93	1222.85	2474.54
Random	599.64	1217.32	2465.47	613.63	1253.49	2539.39
Adaptive1800000	588.61	1200.44	2441.65	599.76	1226.55	2506.68
Adaptive1900000	587.30	1199.64	2437.18	599.73	1227.68	2508.89
Adaptive2000000	588.36	1197.74	2438.04	601.41	1227.86	2507.62

Table 4.38 Processing Time Results for Experiment 9 and 10

	Experiment 9			Experiment 10		
	Inst. 250	Inst. 500	Inst. 1000	Inst. 250	Inst. 500	Inst. 1000
Round Robin	254.23	523.29	1062.33	159.63	330.68	671.59
ESCE	254.33	523.74	1062.80	159.77	330.87	671.90
Random	256.36	524.25	1063.77	156.99	325.28	660.93
Adaptive1800000	255.73	526.38	1069.44	156.53	323.55	656.02
Adaptive1900000	252.73	519.80	1059.22	157.51	324.76	657.62
Adaptive2000000	253.25	528.01	1058.70	157.03	322.67	659.19

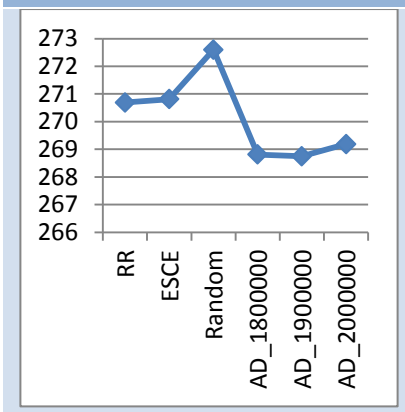
The following Table 4.39 presents processing time graphs for the ten burst workload.

Table 4.39 Processing Time Charts for the Ten Experiments

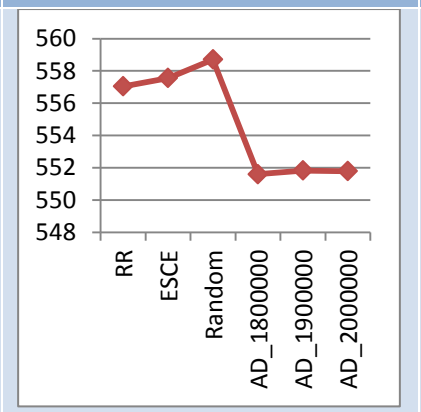
Experiment 1		
Inst. 250	Inst. 500	Inst. 1000
		

Experiment 2

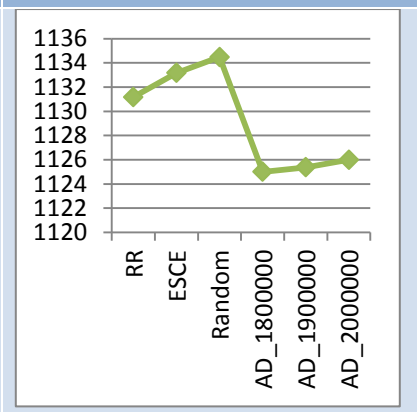
Inst. 250



Inst. 500

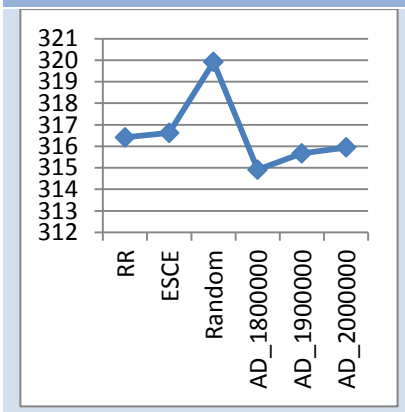


Inst. 1000

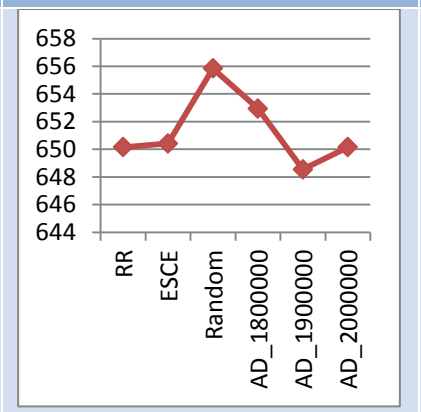


Experiment 3

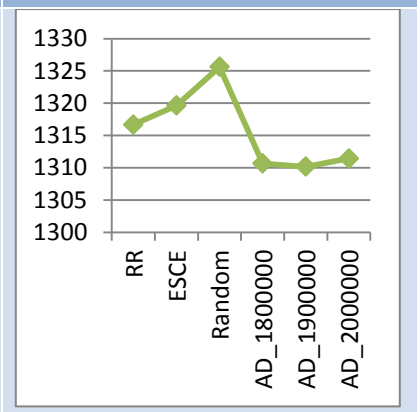
Inst. 250



Inst. 500

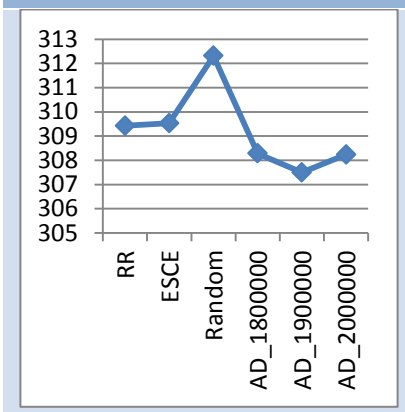


Inst. 1000

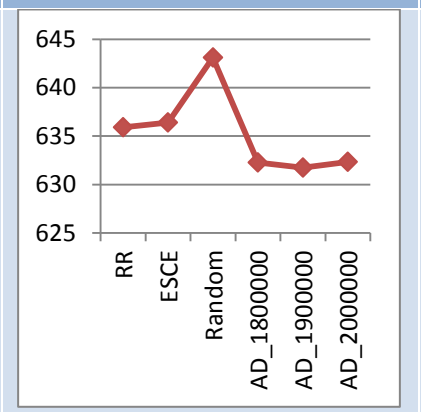


Experiment 4

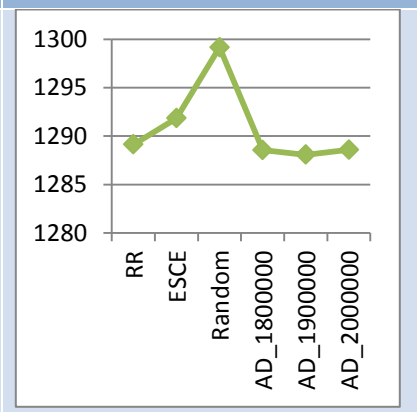
Inst. 250



Inst. 500

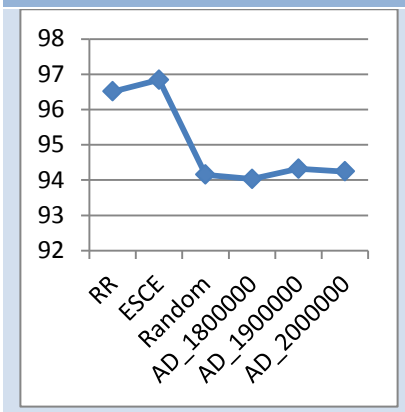


Inst. 1000

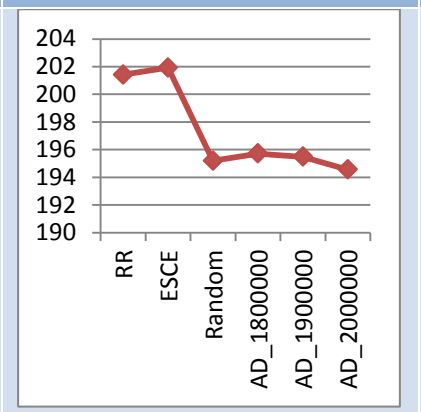


Experiment 5

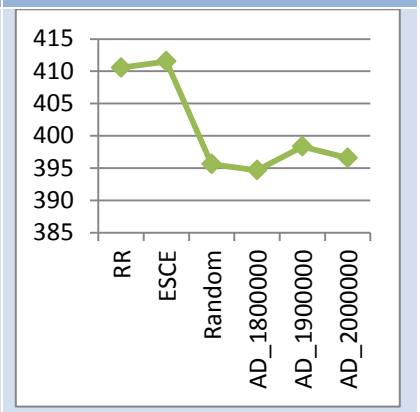
Inst. 250



Inst. 500

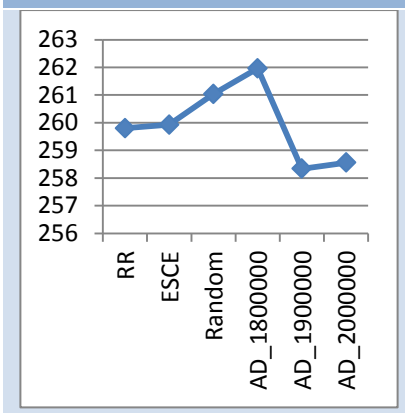


Inst. 1000

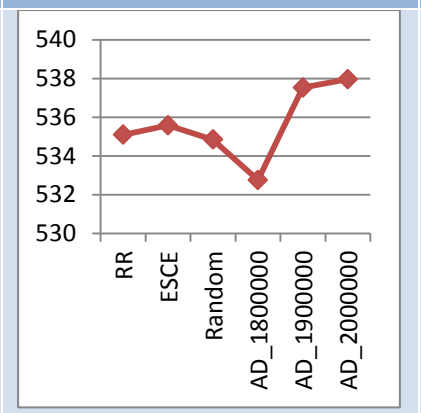


Experiment 6

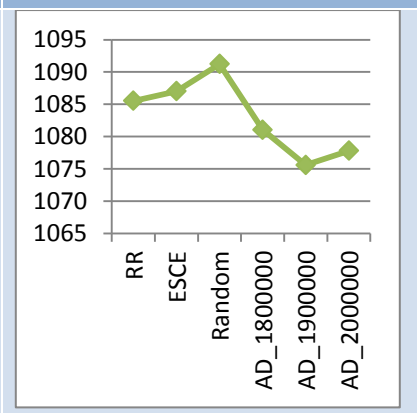
Inst. 250



Inst. 500

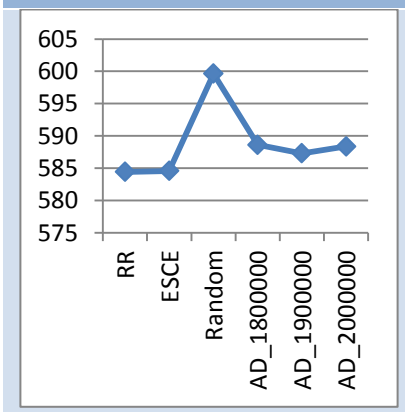


Inst. 1000

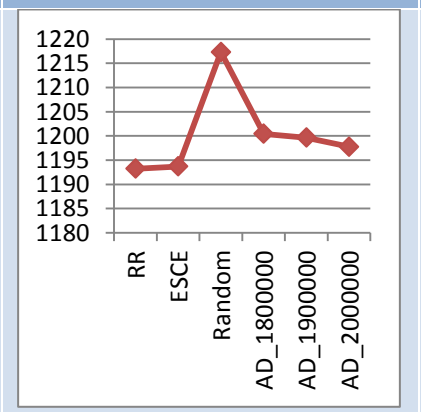


Experiment 7

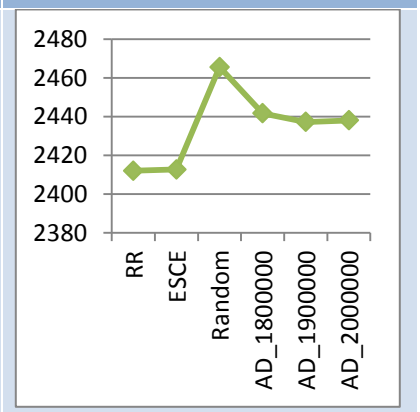
Inst. 250



Inst. 500

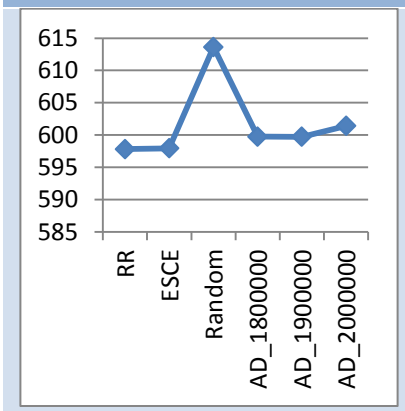


Inst. 1000

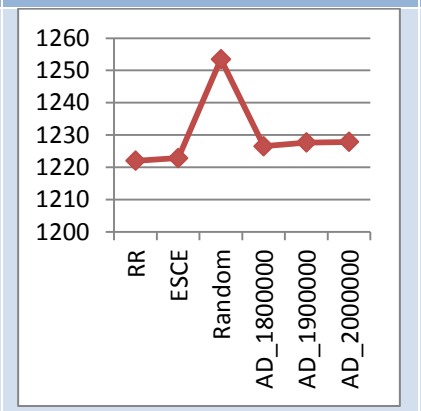


Experiment 8

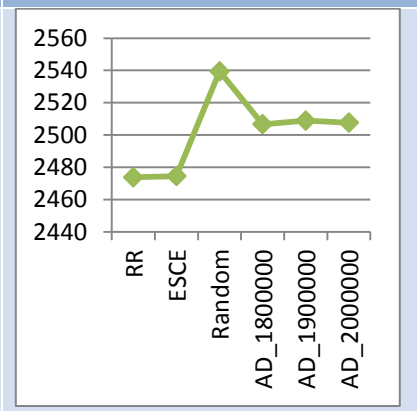
Inst. 250



Inst. 500

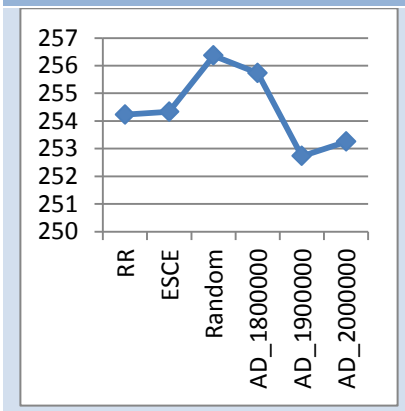


Inst. 1000

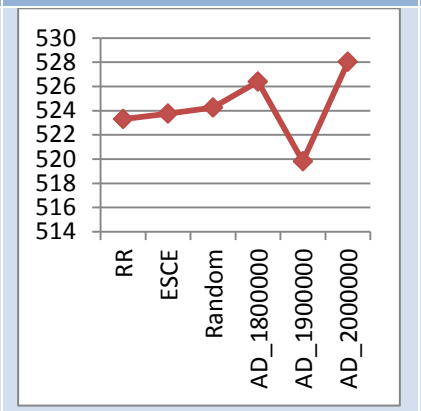


Experiment 9

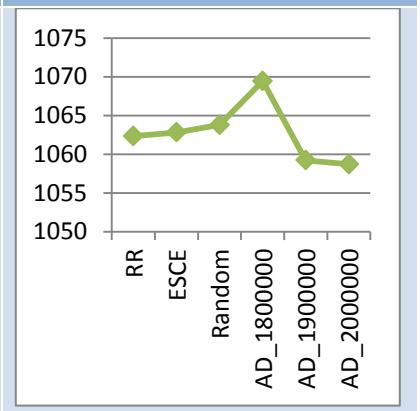
Inst. 250



Inst. 500

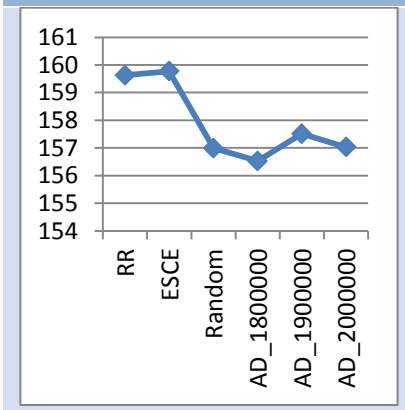


Inst. 1000

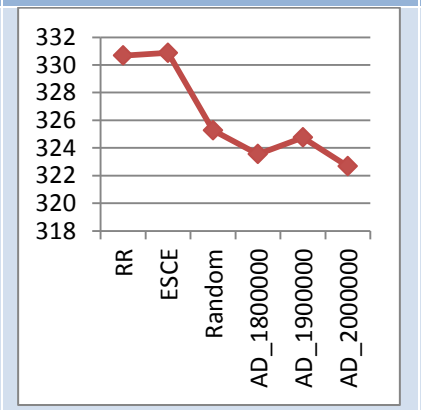


Experiment 10

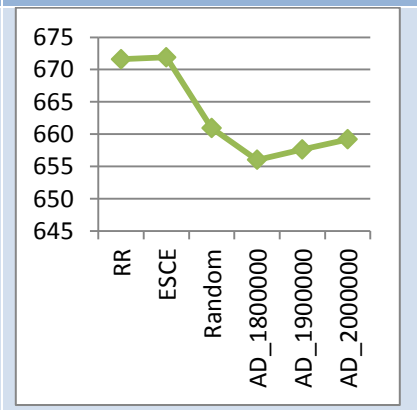
Inst. 250



Inst. 500



Inst. 1000



4.5.3.3 Discussion

In this experiment Adaptive algorithm has been tested under different levels of bursty workload. The results showed that Adaptive algorithm improved the response and processing time for the cloud system under bursty workload except experiment 7 and 8. Our explanation for this exception is that the load most the time is very low (Experiment 7) or very high (Experiment 8) so the threshold used in the experiments is not suitable for those cases. In addition in these experiments we repeated the experiment for every workload using three lengths of executed instruction (250, 500, and 1000). The results showed that the differences in performance increased between adaptive and other algorithms. This is because the instruction takes much more time to be processed by the VM so the VM will be busy in this time and the waiting queue will become longer so the response time and the processing time will be increased for RR, ESCE and Random. On the other hand using fuzzifier in Adaptive algorithm improved the performance because it helps the Fuzzy RR and the Fuzzy Random to select VM which has low load and high speed so it can finish the task in shorter time.

4.5.4 Results Discussion Part III

In this part adaptive algorithm had been tested under bursty workload. Firstly we test the Adaptive algorithm without using fuzzy. The results showed that adaptive algorithm improved the response and the processing time. Secondly the experiment was repeated by using fuzzy with Adaptive algorithm. The results showed that the fuzzifier had improved the response time of Adaptive algorithm by 1ms. This small improvement is because the length of the instruction is small so the waiting queue for VM would not be dramatically apparent. To prove our hypothesis, Adaptive algorithm had been tested under ten different bursty workloads and using three different instruction length 250, 500, 1000 Byte. The results showed that when the instruction length is 250 the response time is decreased by 2 ms. When the instruction length is increased to be 1000 Byte, the response time decreased by about 10 ms. As it is observed from the experiments results the Adaptive algorithm improved the performance of the cloud system by decreasing the response time and the processing time and this improvement become obviously when the instruction length increased.

4.6 Summary

In this chapter the experiments were presents and discussed in details. The experiments were done using CloudAnalyst simulator. The simulation duration was configured to be 1 day. The results ware compared with three of the most popular load balancing algorithms which are: RR, ESCE, and Random.

The experiments were done on three parts. Firstly, we tried to understand the variations in the performance and the differences between RR, ESCE, and Random algorithms in normal and bursty workload. The results showed that the bursty workload degrade the performance of load balancing algorithms.

In the second part RR, ESCE, and Random algorithms were tested under high and low workload. The main objective from these experiments was to find the best algorithm in high workload and the best one in low workload to be used in the Adaptive algorithm. The results guide us to use RR in bursty cases and Random in normal cases.

Finally, the third part of experiments was performed to test and evaluate the performance of the Adaptive algorithm in different levels of bursty workload. In every experiments RR, ESCE, Random, and Adaptive algorithm where tested by using three different instruction length (250 Byte, 500 Byte, 1000 Byte). Overall, results indicated that the Adaptive algorithm improved the performance of the cloud system and this improvement became much more obvious when the instruction length increased. The improvement in the response time ranged from 2ms (when the instruction length is 250) to 10 ms (when the instruction length is 1000) while in the processing time it ranges from 2ms to 5 ms.

Chapter 5

Conclusion and Future Works

Chapter 5 Conclusion and Future Works

5.1 Conclusion

In this thesis, a load balancing algorithm which handles the problem of bursty workload has been introduced. The proposed algorithm called Adaptive Load Balancing Algorithm and is based mainly on swapping between two different algorithms (RR and Random) according to the workload status. Adaptive Algorithm consists of 3 main parts: burst detector, load balancing algorithms, fuzzifier. When the data center receives a request, the burst detector decides the state of the workload (Normal or Burst). Depending on the detector decision, if the workload state is burst so the RR will be applied, otherwise the Fuzzy Random will be applied. The main role of the fuzzifier is to prepare a candidate set of the most balanced VMs based on CPU speed and current load on the VM. This set is used by Fuzzy RR and Fuzzy Random to select the VM to handle the received request.

To test our algorithm CloudAnalyst simulator had been used. Several experiments had been done on different workload patterns. For evaluation, two metrics had been chosen: processing time, and response time. The results had been compared with three popular load balancing algorithms: RR, ESCE, and Random.

The experiments in this work divided into three parts. In the first part experiments were done in order to study the performance of the RR, ESCE, and Random algorithms in bursty and normal workloads. The results showed that the burstness affect the performance of the load balancing algorithms. The second part of experiments had been done to decide which the two algorithms that will be used are in Adaptive algorithm. Based on the results of this part the RR is used when the workload is bursty and the Random is used when the load is normal. In the third part the Adaptive algorithm was tested under different bursty levels. The results showed that Adaptive algorithm decreased the response and the processing time. The decreasing in response time and processing time is about 2 ms when the instruction length was 250 Byte and the decreasing became more obviously with 10 ms for response time and 5 ms for processing time when the instruction length was 1000 Byte.

5.2 Future Works

For future works several suggestions may be done in order to enhance Adaptive Algorithm:

1. A prediction algorithm may be used to predict the workload and decide the proper threshold.
2. Measure the resource utilization such as CPU, and Memory.
3. Power consumption is a very important issue, so we need to test this metric and improve the algorithm to minimize the consumption of the power.
4. Test the proposed algorithm on a real cloud.

References

1. Zhang, Q., Cheng, L., and Boutaba, R., *Cloud computing: state-of-the-art and research challenges.* Journal of Internet Services and Applications, **1**(1): pp. 7-18,(2010).
2. Peter Mell, T.G., *The NIST Definition of Cloud Computing*, (2009).
3. Ratan, M. and Anant, J., *Ant colony Optimization: A Solution of Load Balancing in Cloud.* International Journal of Web & Semantic Technology (IJWesT), **3**(2): pp. 33-50,(2012).
4. Dong, J., *Network Dictionary*. Javvin Technologies Inc, (2007).
5. Sajid, M. and Raza, Z. *Cloud Computing: Issues & Challenges*. in *International Conference on Cloud*. pp. 35-41, (2013).
6. Uma, J., Ramasamy, V., and Kaleeswaran, A., *Load Balancing Algorithms in Cloud Computing Environment - A Methodical Comparison.* International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), **3**(2): pp. 272-275,(2014).
7. Sarna, D.E.Y., *Implementing and Developing Cloud Computing Applications*. CRC Press, (2011).
8. Nandgaonkar, S.V. and Raut, A.B., *A Comprehensive Study on Cloud Computing.* International Journal of Computer Science and Mobile Computing, **3**(4): pp. 733 – 738,(2014).
9. Devasena, L.C., *Impact Study Of Cloud Computing On Business Development.* Operations Research and Applications: An International Journal (ORAJ), **1**(1): pp. 1-7,(2014).
10. Velte, A.T., Velte, T.J., and Elsenpeter, R., *Cloud Computing A Practical Approach*. TATA McGRAW-HILL, (2010).
11. Marisol, G.V., Cucinotta, T., and Lu, C., *Challenges in real-time virtualization and predictable cloud computing.* Journal of Systems Architecture (ELSEVIER): pp. 1-15,(2014).
12. Kaleeswari and Juliet, N., *Dynamic Resource Allocation by Using Elastic Compute Cloud Service.* International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET), **3**(5): pp. 12375-12379,(2014).
13. Sareen, P., *Cloud Computing: Types, Architecture, Applications, Concerns, Virtualization and Role of IT Governance in Cloud.* International Journal of Advanced Research in Computer Science and Software Engineering, **3**(3): pp. 533-538,(2013).
14. Alakeel, A.M., *A Guide to Dynamic Load Balancing in Distributed Computer Systems.* International Journal of Computer Science and Network Security (IJCSNS), **10**(6): pp. 153-160,(2010).
15. Jena, S. and Ahmad, Z., *Response Time Minimization of Different Load Balancing Algorithms in Cloud Computing.* International Journal of Computer Applications, **69**(17): pp. 22-27,(2013).
16. Wu, K., Chen, L., Ye, S., and Li, Y., *A Load Balancing Algorithm based on the Variation Trend of Entropy in Homogeneous Cluster.* International Journal of Grid and Distributed Computing, **7**(2): pp. 11-20,(2014).
17. Chaczko, Z., Mahadevan, V., Aslanzadeh, S., and Mcdermid, C. *Availability and Load Balancing in Cloud Computing*. Singapore: International Conference on Computer and Software Modeling, (2011).
18. Mehta, R., Yask, P., and Harshal, T., *Architecture For Distributing Load Dynamically In Cloud Using Server Performance Analysis Under Bursty Workloads.* **1**(9),(2012).
19. Subramanian S, N.K.G., Kiran Kumar M, Sreesh P, and G R Karpagam, *An Adaptive Algorithm For Dynamic Priority Based Virtual Machine Scheduling In Cloud.* International Journal of Computer Science Issues, **9**(6),(2012).
20. Mohapatra, S., Rekha, K.S., and Mohanty, S., *A Comparison of Four Popular Heuristics for Load Balancing of Virtual Machines in Cloud Computing.* International Journal of Computer Applications, **68**,(2013).

21. Sharma, T. and Banga, V., *Efficient and Enhanced Algorithm in Cloud Computing.* **3**(1),(2013).
22. Mohialdeen, I.A., *Comparative Study Of Scheduling Algorithms In Cloud Computing Environment.* Journal of Computer Science, **2**(9): pp. 252-263,(2013).
23. Khanghahi, N. and Ravanmehr, R., *Cloud Computing Performance Evaluation: Issues And Challenges.* International Journal on Cloud Computing: Services and Architecture (IJCCSA), **3**(5): pp. 29-41,(2013).
24. Chin, M.L., Tan, C., and Bandan, M.I., *Efficient DNS based Load Balancing for Bursty Web Application Traffic.* **1**,(2012).
25. Li, H. and Muskulus, M., *Analysis and Modeling of Job Arrivals in a Production Grid.* **34**(4): pp. 59-70,(2007).
26. Mia, N., Zhangb, Q., Riskac, A., Riskac, E., and Riedel, E., *Performance impacts of autocorrelated flows in multi-tiered systems.*
27. Riska, A. and Riedel, E. *Long-Range Dependence at the Disk Drive Level*. in *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*. IEEE, (2006).
28. McNeill, F.M. and Thro, E., *Fuzzy Logic A Practical Approach*. United Kingdom: Academic Press Limited. 279, (1994).
29. Helmy, T., Al-Jamimi, H., Ahmed, B., and Loqman, H., *Fuzzy Logic–Based Scheme for Load Balancing in Grid Services.* A Journal of Software Engineering and Applications: pp. 149-156,(2012).
30. Sethi, S., Anupama, S., and Jena, K., S, *Efficient load Balancing in Cloud Computing using Fuzzy Logic.* IOSR Journal of Engineering (IOSRJEN), **2**(7): pp. PP 65-71,(2012).
31. Dave, S. and Maheta, P., *Utilizing Round Robin Concept for Load Balancing Algorithm at Virtual Machine Level in Cloud Environment.* International Journal of Computer Applications, **49**(4),(2014).
32. Singhal, U. and Jain, S., *A New Fuzzy Logic and GSO based Load balancing Mechanism for Public Cloud.* International Journal of Grid Distribution Computing, **7**(5): pp. 97-110,(2014).
33. Tai, J., Zhang, J., Li, J., Meleis, W., and Mi, N. *ARA: Adaptive Resource Allocation for Cloud Computing Environments under Bursty Workloads*. in *IEEE International Performance Computing and Communications Conference*. IEEE International Performance Computing and Communications Conference, (2011).
34. Naik, N. and Patel, A., *Load Balancing Under Bursty Environment For Cloud Computing.* International Journal of Engineering Research & Technology (IJERT), **2**(6),(2013).
35. Ghorbani, M., Wang, Y., Xue, Y., Pedram, M., and Bogdan, P. *Prediction and Control of Bursty Cloud Workloads: A Fractal Framework*. in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2014 International Conference on* New Delhi IEEE pp. 1 - 9 (2014).
36. Zhang, J., Mi, N., Tai, J., and Meleis, W. *Decentralized Scheduling of Bursty Workload on Computing Grids*. IEEE, (2011).
37. Zhang, J. and Meleis, W. *Adaptive Grid Computing For Mpi Applications*. in *Parallel and Distributed Computing and Systems*. (2009).
38. *jFuzzyLogic*. [cited 2015; Available from: http://jfuzzylogic.sourceforge.net/html/index.html.
39. Cingolani, P. and Jesús, A.-F., *jFuzzyLogic: a Java Library to Design Fuzzy Logic Controllers According to the Standard for Fuzzy Control Programming.* International Journal of Computational Intelligence Systems, **6**(1): pp. 61–75,(2013).
40. Cingolani, P. and Jesus, A.-F. *jFuzzyLogic: a robust and flexible Fuzzy-Logic inference system language implementation*. in *Fuzzy Systems (FUZZ-IEEE)*. IEEE International Conference on. IEEE, (2012).
41. *cloudsim*. [cited 2014; Available from: http://www.cloudbus.org/cloudsim/.

42. Calheiros, R.N., Ranjan, R., De Rose, C.A.F., and Buyya, R., *CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services*. pp. 1-9, (2009).
43. Pakize, S.R., Khademi, S.M., and Gandomi, A., *Comparison Of CloudSim, CloudAnalyst And CloudReports Simulator in Cloud Computing.* International Journal of Computer Science And Network Solutions, **2**: pp. 19-27,(2014).
44. Buyya, R., Ranjan, R., and Calheiros, R.N. *Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities*. in *International Conference on High Performance Computing and Simulation*. Proceedings of the 2009 International Conference on High Performance Computing and Simulation, HPCS 2009, (2009).
45. Wickremasinghe Bhathiya, N., C.R., and Rajkumar, B. *CloudAnalyst: A CloudSim-based Visual Modeller for Analysing Cloud Computing Environments and Applications.* . in *International Conference on Advanced Information Networking and Applications (AINA)*. IEEE Computer Society, pp. 446-452, (2010).
46. Wickremasinghe, B., *CloudAnalyst: A CloudSim-based Tool for Modelling and Analysis of Large Scale Cloud Computing Environments*, (2009).
47. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., and Buyya, R., *CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms.* Software: Practice and Experience, **41**(1): pp. 23-50,(2011).
48. Mishra, R.K. and Bhukya, S.N., *Service Broker Algorithm for Cloud-Analyst.* International Journal of Computer Science and Information Technologies, **5 (3)**: pp. 3957-3962,(2014).
49. Raghava, S. and Singh, D., *Comparative Study on Load Balancing Techniques in Cloud Computing.* Open Journal Of Mobile Computing And Cloud Computing, **1**(1): pp. 18-25,(2014).

Appendix A

CloudAnalyst Simulator Screens

Appendix A CloudAnalyst Simulator Screens

A.1. CloudAnalyst Main Screen

When CloudAnalyst starts, the main screen appears with a map of the world on the center of the screen. As can be seen in Figure A.1 CloudAnalyst divides the world in to 6 regions located in the 6 main continents. Those regions are used in distributing DCs and UBs during configuration process.

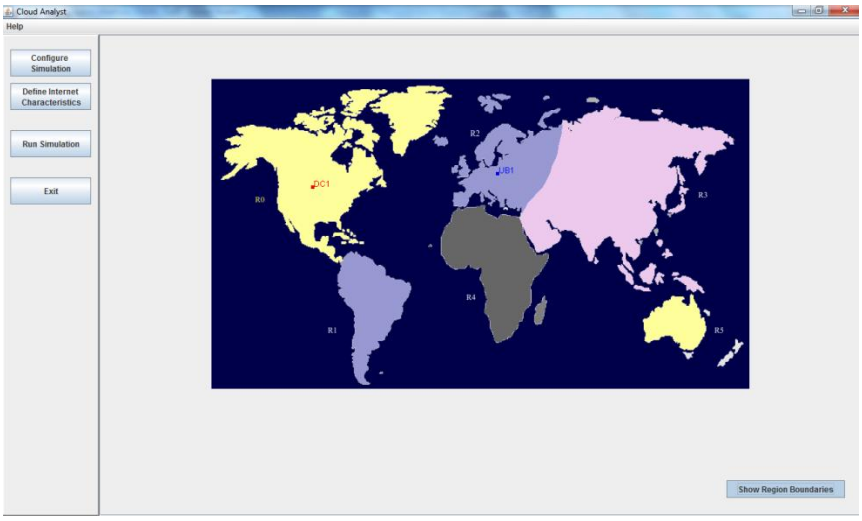


Figure A.1 CloudAnalyst Main Screen

On the left side of the screen there is a Control Panel with the following 4 options:

1. **Configure Simulation:** Opens the Configure Simulation Screen
2. **Define Internet Characteristics:** Opens Internet Characteristics Screen
3. **Run Simulation:** Starts the simulation
4. **Exit**.

A.2. Configure Simulation Screen

Configure Simulation screen has three tabs:

1. **Main Configuration Tab**

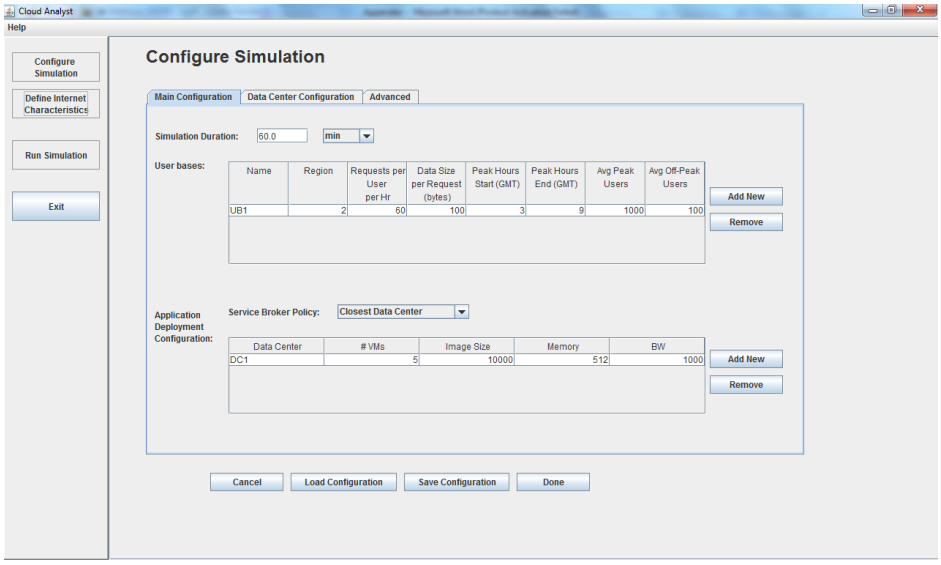


Figure A.2 Main Configuration Tab

The configuration options on the main tab as shown in Figure A.2 are:

1. Simulation time: the duration of the simulation which can be given in minutes, hours or days
2. User Bases Table: This is a table listing out all the user bases in the simulation. Each user base has following configurable fields, represented by a single row in the table.
   a. Name
   b. Region
   c. Requests per user per hour
   d. Data size per request
   e. Peak hours
   f. Average users during peak hours
   g. Average users during off-peak hours

The Add and Remove buttons next to the table can be used to add or remove user bases from the configuration.

3. Application Deployment Configuration: This table lists how many virtual machines are allocated for the application in each data center from the Data Centers tab, along with the details of a virtual machine. The fields are:
   a. Data Center: This is a drop down listing the names of data centers created in the Data Center tab.
   b. Number of VMs: How many VMs to be allocated to the application from the selected data center
   c. Image Size: a single VM image size in bytes
   d. Memory: amount of memory available to a single VM
   e. BW: amount of bandwidth available to a single VM

4. Service Broker Policy: This drop down allows you to select the brokerage policy between data centers that decide which data center should receive traffic from which user base. The available policies are:
    a. Closest data center: The data center with the least network latency (disregarding network bandwidth) from a particular user base is sent all the requests from that user base.
    b. Optimize response time: This policy attempts to balance the load between data centers when one data center gets over loaded.

The "Save Configuration" button allows you to save the configuration created as a file. Simulation files are saved with a (.sim) extension. Similarly using the "Load Configuration" button you can load a previously saved simulation configuration.

2. Data Center Tab

The data center tab allows you to define the configuration of a data center as shown in Figure A.3. The table at the top lists the data centers and using the Add/Remove buttons you can add or remove data centers to the configuration. The parameter fields are:

1. Name
2. Region
3. Architecture – Architecture of the servers used in the data center. e.g. X86
4. Operating System – e.g. Linux
5. Virtual Machine Monitor (VMM)
6. Cost per VM Hour
7. Cost per 1Mb Memory Hour
8. Storage cost per Gb
9. Data Transfer cost per Gb (both in and out)
10. Number of servers

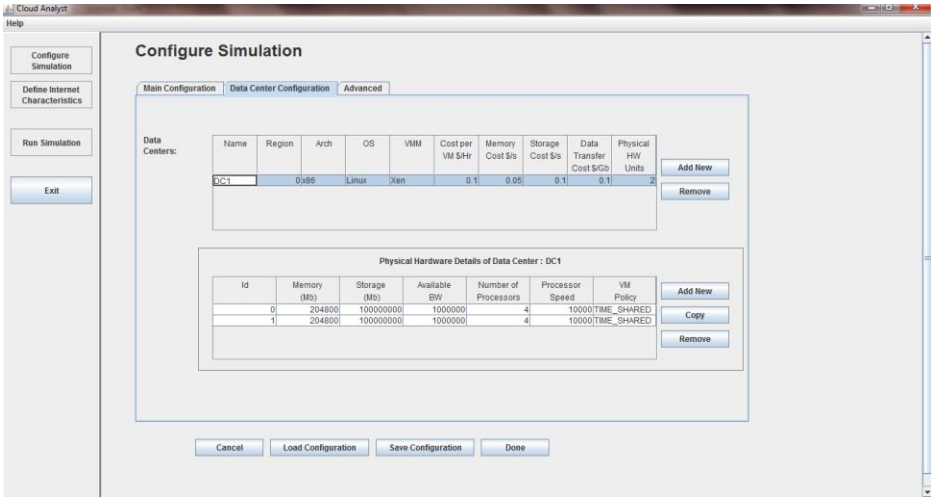


Figure 0.3 Data Center Configurations

When you select a data center from this table a second table will appear below it with the details of the server machines in the data center. The parameters for each machine can be given according to the available fields.

1. Machine Id
2. Memory
3. Storage
4. Available network bandwidth
5. Number of processors
6. Processor speed (MIPS)
7. VM allocation policy (time shared/space shared)

3. Advanced Tab

The advanced tab contains some important parameters that apply to the entire simulation as shown in Figure A.4.

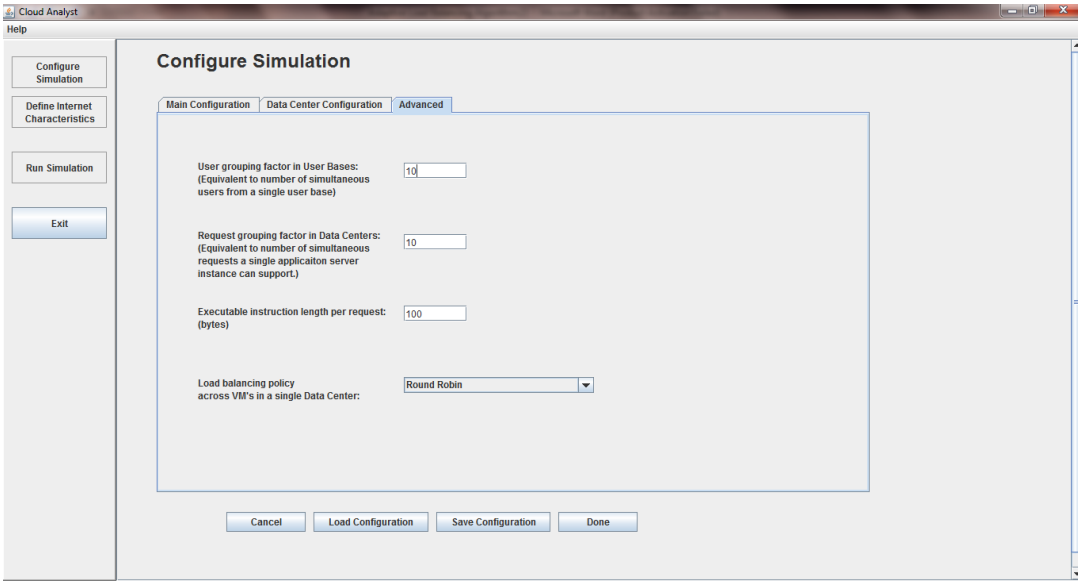


Figure A.4 Advanced Tab Configurations

1. User Grouping Factor (in User Bases) – This parameter tells the simulator how many users should be treated as a single bundle for traffic generation. The number given here will be used as the number of requests represented by a single InternetCloudlet. In the ideal scenario this parameter should be 1, with each individual user represented independently. But that will increase the simulation time unrealistically.
2. Request Grouping Factor (in Data Centers) – This parameter tells the simulator how many requests should be treated as a single unit for processing. i.e. this many requests are bundled together and assigned to a single VM as a unit. Again, ideally this should be equal to 1. But it could also be viewed as the number of simultaneous threads a single application instance (VM) can handle.

3. Executable instruction length (in bytes) – This is the main parameter that affects the execution length of a request. This is the same GridletLength parameter used in GridSim.
4. Load balancing policy – the load balancing policy used by all data centers in allocating requests to virtual machines. Available policies are:
   a. Round-robin
   b. Equally Spread Current Execution Load – The load balancer keeps track of how many Cloudlets are currently being processed by each VM and tries to even out the active load.
   c. Throttled – The load balancer throttles the number of requests assigned to a single VM.

A.3. Internet Characteristics Screen

The Internet Characteristics screen can be used to set the Internet latency and bandwidth parameters. It presents two matrices for these two categories as seen in Figure A.5.

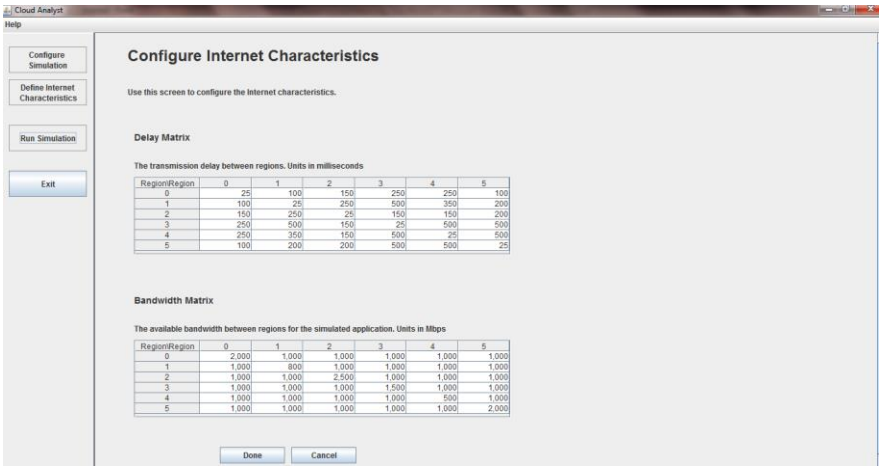


Figure A.5 Internet Characteristics Configuration

A.4. Results Screen

Once the simulation is completed the main response times will be displayed on the simulation panel next to each user base. The detailed results can be viewed by clicking the "View Detailed Results" button that appears at the right hand bottom corner of the screen after the simulation has completed.
The results screen as shown Figure A.6 will list out the data collected from the simulation. This includes:

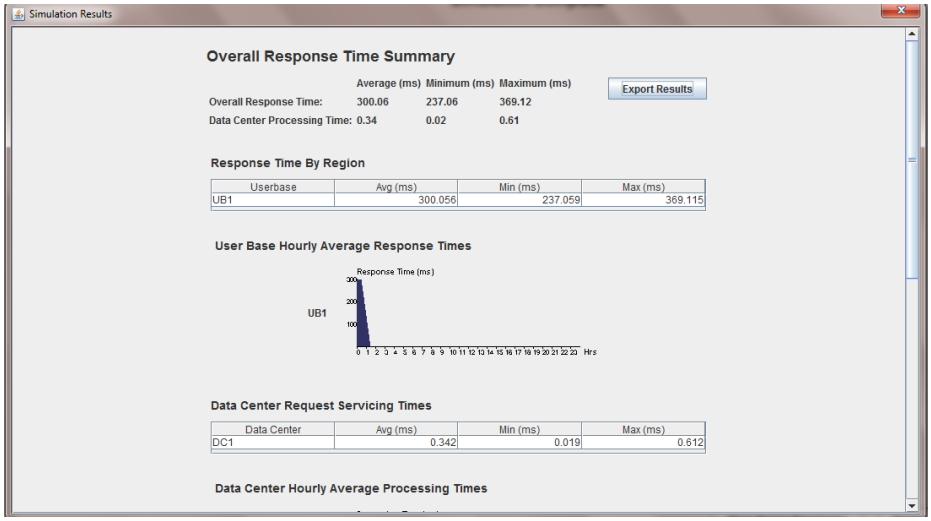


Figure A.6 Results Screen

1. Overall response time summary (for all the user bases)
2. Response time by user base in tabular format
3. Response time by user base in graphical format broken down into the 24 hours of the day
4. Request servicing time by each data center in tabular format
5. Request servicing time by data center in graphical format broken down into 24 hours of the day
6. Data center loading (number of requests serviced) in graphical format broken down in to 24 hours of the day
7. Cost details