ATHABASCA UNIVERSITY

A Predictive Workload Balancing Algorithm in Cloud Services

BY

MAHDEE JODAYREE

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF
MASTER OF SCIENCE IN INFORMATION SYSTEMS

SCHOOL OF COMPUTING AND INFORMATION SYSTEMS

ATHABASCA UNIVERSITY
August, 2018

Approval of Thesis

The undersigned certify that they have read the thesis entitled

A PREDICTIVE WORKLOAD BALANCING ALGORITHM IN CLOUD SERVICES

Submitted by

Mahdee Jodayree

In partial fulfillment of the requirements for the degree of

Master of Science in Information Systems

The thesis examination committee certifies that the thesis
and the oral examination is approved

Supervisor:
Dr. Mahmoud Abaza
Athabasca University

Committee Member:
Dr. Ching Tan
Athabasca University

External Examiner:
Dr. Ebrahim Bagheri
Ryerson University

August, 2018

1 University Drive, Athabasca, AB, T9S 3A3 Canada
P: 780.675-6821 | Toll-free (CAN/U.S.) 1.800.788.9041 (ext 6821)
fgs@athabascau.ca | fgs.athabascau.ca | athabascau.ca

Abstract

In today's business world, many companies and government agencies depend on the infrastructures of cloud services to host and process their information. Load processing of many cloud services is distributed in a static manner which can overload the largest available systems. This paper is an exploratory study on the predictive approach for dynamic resource distribution of cloud services.

Today, many cloud service providers are exploring the benefit of dynamic workload-balancing for their resource management. Rather than issuing fixed resources to each customer, a dynamic hosting alternative offers a way to allocate resources dynamically and more efficiently to save computational power.

Efficient cloud resource management can be achieved by simulating cloud services based on the predictions of incoming workloads, which can be more efficient than static allocation methods (Wolke, Bichler, and Setzer, 2015). Previous researchers in this area have focused on dynamic load balancing algorithms that are based on a current workload demanded by a client. These approaches require high computational power and additional time to meet the demands of dynamic cloud services. This paper introduces a rule-based workload-balancing algorithm based on the predictions of an end-to-end system called Cicada. A simulation of cloud services can be achieved by a cloud service simulator called CloudSim and it will be used to achieve an algorithm with lower computational demand and a faster workload balancing. The final result will demonstrate the effectiveness of a predictive workload balancing approach that can achieve faster workload balancing with a lower computational power usage.

Acknowledgements

This thesis could not be possible without the help of my supervisor Dr. Abaza and Dr.Tan. My gratitude to my supervisor, Dr. Mahmoud Abaza, who read my revisions and advised me towards the completion of my thesis. I also would like to thank committee member Dr.Tan who offered support and direction for my proposal and my thesis work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Background

Cloud computing services play a major role in today's computing. Leading information technology companies like Amazon's AWS, HP, Microsoft, and Google deploy large data centers with extensive hardware network for effective service delivery to cloud clients. Cloud service providers require proper resource management and provisioning to allow clients to access cloud services from the internet (Singh, & Jangwal, 2012). In recent years, cloud service providers have shifted towards dynamic resource management to enable sharing of cloud computing resources between different users. Dynamic cloud computing technique enables resources to be assigned to different clients based on the current demand of each client turning the cloud to a limitless computational platform with limitless storage space which improves the performance of cloud services. To achieve best resource allocation in dynamic hosting frameworks, cloud service providers should provision resources intelligently to all clients. This intelligent resource balancing is known as workload balancing in a cloud service models. Cloud service environments have adapted different provisioning strategies to improve their service level.

1.2 Statement of Purpose

The main problem with load-balancing in a dynamic cloud environment, is the overload prevention problem. Today many load-balancing algorithms focus on balancing the current over-loads rather than preventing it in a first place.

To prevent any overloads or any over-provisioning in a dynamic cloud environment, one must find a reliable method of workload prediction and a reliable framework for simulating a cloud environment. This paper reviews previous literatures to presents a reliable prediction

This research explores the effectiveness of different cloud simulators and different prediction tools to predict a workload of a cloud efficiently. This paper will explore the cloud service simulation problems and resource management algorithms.

1.3 Contribution and Significance

The use of static resources for cloud services (Sheng, Qiao, Vasilakos, Szabo, Bourne, & Xu, 2014) has many drawbacks, for instance, static hosting in web-based platforms has unreliable service level, where a single outage can make the platforms unusable. It is also very costly and inefficient to assign a static amount of resources to a specific workload where static resources can remain unused for various periods of time. The dynamic hosting approach can enable vendors and providers to support efficient resource allocation and resource management mechanisms for their hosting platforms, however dynamic resource management for cloud services requires an efficient and a fast resource allocation algorithm.

The main contribution of this research is a proposed ruled-based algorithm called C-Rule Algorithm that would use a very efficient prediction tool (LaCurts, 2014) simulation framework to prevent any unbalanced in the system in a dynamic environment. This unbalance prevention will be achieved by simulating different resource allocation scenario of a predicted workload, in order to achieve an optimal resource provisioning for a specific workload with a low computation need.

1.4 Organization of Thesis

Chapter 1 introduces the main goal and the layout of the paper. Chapter II contains a review of literature of previous researches related to workload balancing of cloud services. Chapter II also provides background information and introduces the necessary tools for achieving a predictive workload balancing. Chapter III presents the C-Rule algorithm and the concept behind it. Chapter IV explains the method of modeling for this paper and it describes the procedures for importing and running data in CloudSim simulation. Chapter V implements the cloud simulation by CloudSim and explains each individual step required to achieve simulation in CloudSim. APPENDIX B provides the result for this research paper including the final output from the CloudSim. APPENDIX C provides the Java code for a cloud simulation and outlines the implementation of the newly introduced algorithm. Chapter V contains the final conclusion of this paper.

1.5 Research Contribution

The core contribution of this paper is an introduction of a new predictive rule based algorithm which can predict the incoming cloud workloads by analyzing all incoming network traffics and compare that prediction to historical load balancing results and decide whether the cloud can handle the incoming workload. The main contribution of the algorithm will be to prevent any over-provisioning or any under-provisioning. The new introduced rule-based algorithm will be called **Custom-rule algorithm or C-Rule algorithm**.

1.6 Hypothesis

The following hypotheses will be proven in this research.

H0: A reliable load balancing is achievable based on the predictions of end-to-end software toolkit called Cicada

H1: A rule-based workload-balancing algorithm based on prediction of Cicada will consume less computational power than a non-predictive workload-balancing.

In the next section the research questions of this thesis paper will be discussed.

1.7 Research Questions

With these goals in mind, the following research questions will help the researcher to explore the most efficient intelligent load balancing algorithm for dynamic internet hosting.

1. Under what conditions the prediction of Cicada cannot be reliable to predict the amount of workload in a dynamic internet hosting platforms?
2. Under what conditions CloudSim cannot generate a reliable workload simulation?

CHAPTER 2 LITERATURE REVIEW

This chapter introduces the literature review of predictive workload balancing and demonstrates the reasons, which led to a prediction-based workload balancing. An overview of different algorithms, simulation framework and previous studies on dynamic provisioning will be covered in this chapter. The main advantage and disadvantages of previous introduced algorithms and predictions will be also discussed in this chapter. The main contribution of this research will be discussed in the Research Contribution and hypothesis section of this chapter.

Today cloud computing enables companies to delivery different computing services such as storages, software, and databases to their clients over the Internet. This resource sharing technique enables organizations to focus on their main objectives rather than on computer infrastructure and maintenance.

There are two resource management models, static and dynamic. Initially, cloud computing services were introduced as static computing services where a specific amount of resources were assigned to specific organizations however over the time with the rapid growth of computing needs for many organizations and business, dynamic cloud computing was introduced. Dynamic cloud computing allowed cloud service providers to share and assign resources based on the demand for a specific workload. The dynamic resource management model enabled limitless computational platform with unlimited storage which improves the performance of cloud computing. For instance, in a static computing, any outrages can generate downtime, wherein dynamic computing, if any outages occur the computing job can be automatically shifted to another location.

2.1 Introduction to Cloud Service and Cloud Deployment Models

There are three types of Cloud computing service model:

- Software As A Service (SaaS)
- Platform As A Service (PaaS)
- Infrastructure As A Service (IaaS)

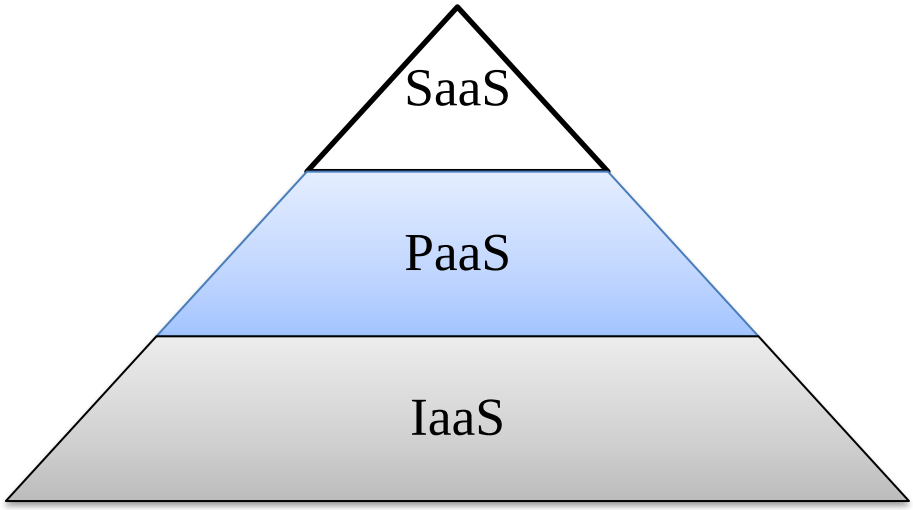

Figure 1. Generic 3-Layer model of cloud computing (Source: Mahmood, 2011)

Software As A Service is the top layer of cloud computing services where software applications mainly standard software is offered as a cloud service to the users. An outstanding example of a SaaS service is Google Docs. Google docs offer a free fully functional word processor, the spreadsheet application, and presentation creator software enabling users to collaborate with each other from different locations.

If users need to develop their own application on the cloud they must use **Platform As A Service (PaaS).** This platform provides a cloud service environment in which developers can use appropriate APIs to make an application such as Facebook, which

can be run and shared in anywhere in the world with any platforms without the risk of software pirating.

Infrastructure as a Service segment of cloud services provide developing tools with limitless storage and computing powers to developers and ordinary users. For example Google drive and Apple iCloud offer cloud storage service for all people including ordinary users and developers. Allowing them to develop, run, and store different applications in cloud environments. For example, Amazon EC2 and Windows Azure are typical IaaS (Sleit, Misk, Badwan, & Khalil, 2013).

Cloud deployment model can be categorized into 5 types:

1. Private clouds
2. Public clouds
3. Community
4. Hybrid
5. Hybrid with Cloud bursting application

A cloud-computing environment is called a private cloud when the provider and consumer are associated with each other, however, in public clouds, there are no associations between the provider and the customer. The customer rents machines from the provider either by the hour or by a different function of time. Hybrid computing is a mixture of public and private computing models and a community cloud is computing infrastructure shared between different organizations.

In public cloud computing, workload balancing is needed for both provider and consumer. In public cloud computing, providers must utilize their resources so that their consumers can have the assurance of receiving sufficient amount of resources. In

addition, there is a monetary exchange between individual consumers and their cloud providers.

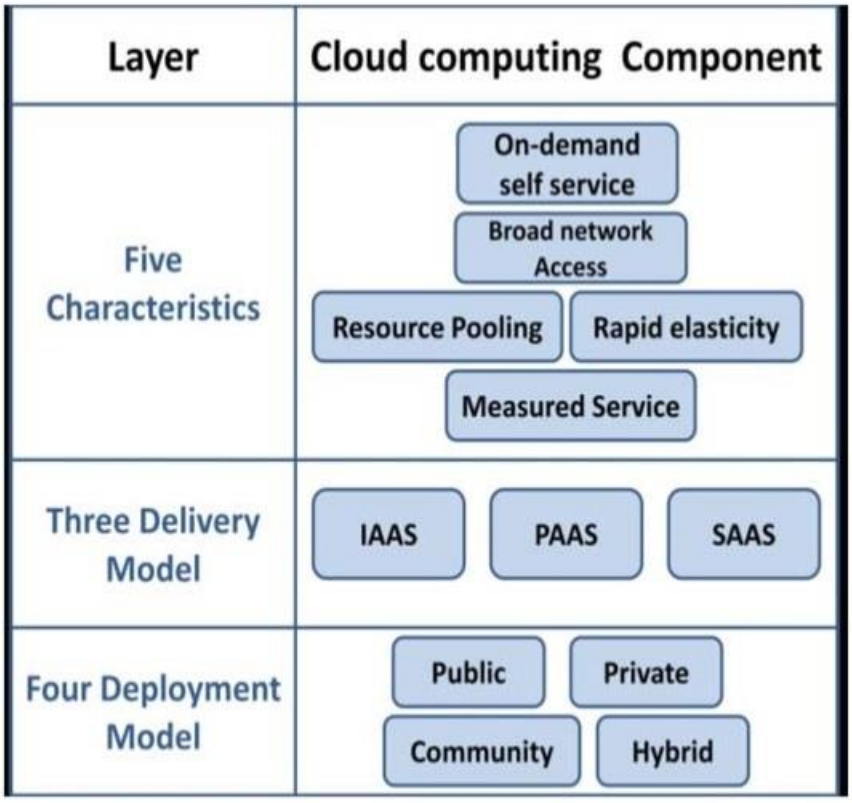

Figure 2 (Model of Cloud Computing) (Kaur & Luthra, 2014).

In a cloud service model, a cloud bursting model is an application deployment model in which the application is configured to run in a private cloud or data center computer and when there is a demand for extra computing capacity, the application burst into a public cloud for extra computing only when it is needed.

2.2 Difference between Dynamic and Static Resource Allocation of Cloud Services

In Cloud computing, the goal of resource allocation is to maximize the possible number of requests that can be processed to reduce application completion time. Dynamic environments can easily create a scarcity of resources in the system, creating a need to find efficient methods of resource allocation. This research will examine the solutions to these problems.

To achieve a reliable workload balancing, there is a need to ensure that hosted applications can handle an unpredictable spike in workload (Al-Qudah, Alzoubi, Allman, Rabinovich, and Liberatore, 2009). Dynamic resources must operate at an optimal level even when experiencing significantly higher request rates. This means that it needs to be able to shift resources to where they are needed when they are needed.

Next section is a literature review on dynamic Cloud provisioning.

2.1 Dynamic Cloud Provisioning

The massive demand for resources in dynamic computing also introduced a new problem which was a need for intelligent resource management. Dynamic cloud computing needed a reliable workload balancing to prevent over provisioning and under provisioning of resources to a client.

Over the years many different technologies have been introduced for resource management. The most popular and key technology which has been introduced for resource management of cloud is the utilization of virtual machines (VMs) for resource scheduling.

VM machines allow emulation of a different computer system based on the specification and computer architectures of a physical computer.VM machines and virtual process machines first were introduced in the 1960s by IBM. Initially, virtual machines were created to run multiple operating systems, by allowing time-sharing between multiple sing-tasking operating systems. There are different kinds of virtual machines, each designed with different functions.

- System Virtual Machines (full virtualizations VMS), is designed to provide a substitute for a real machine allowing them to execute entire operating systems.
- Process virtual machines can execute computer programs in a platform independent environment.

Today majority of cloud service providers use the full virtualizations VMs to provide cloud services such as web hosting services. There are also different kind of VM software, the most popular ones are VirtualBox, Parallels, and VMware. One example of the dynamic cloud provider is GoDaddy which is a web hosting service provider which also uses dynamic provisioning for their cloud services.

Building and operating dynamic cloud services require a deep study of cloud resource management and understand its fundamentals such as Virtual Machines (VMs) and different job scheduling policies. Next section introduces the fundamentals of cloud service and cloud deployment models.

Dynamic Virtual Machines (VMs) offers great potential and benefits in terms of supporting efficient communication mechanisms between applications. The main

benefit is that Dynamic Virtual Machines (VMs) do not require extensive server maintenance given the inherent capacity to respond to additional workload. A growing body of research examines the development of a dynamic VM resource allocation cloud services. In a previous study, Oluwatolani, Babajide, and Philip (2012) presented a scalable architectural model for Web-based applications to ensure availability and reliability even during sudden load increases (See Figure 2). The overarching idea in the proposed architecture is to allow personalization and distributed updating of data through dynamic web applications.

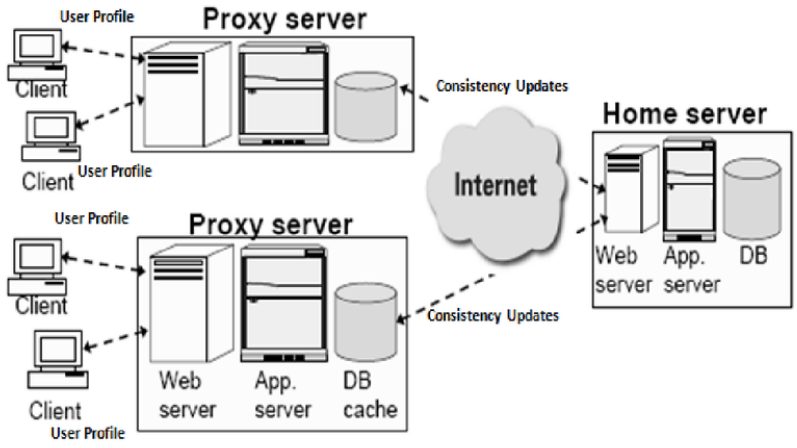

Figure 3: Scalable distributed architecture for Web applications (Oluwatolani, 2012)

In order for a hosting platform to achieve its goals in terms of handling workload demands, it should have the capability to distribute its resources among hosted applications. The main idea is that an experimental platform operating at the system level (IaaS) could feasibly manage cloud resources based on the following envisaged mechanisms:

Requirement Inference: The mechanism to predict resource requirements accurately based on workload needs of applications. The requirement predictions should rely on either analytical models of application or empirical observations

Appropriate resource sharing mechanisms: They should have the mechanism to support components of hosted applications on the constituent nodes

Workload prediction: they should have the capacity to predict system workloads (Chase, Anderson, Thakar, Vahdat & Doyle, 2001).

Dynamic capacity provisioning: They should use appropriate mechanisms for resource allocation to the hosted platforms. These attributes will be used to evaluate hosting allocation strategies to find the most efficient load-balancing alternative for dynamic hosting platforms.

2.2 Predictive Load Balancing Algorithms for Cloud

Many research has been conducted to explore the predictive load balancing for the cloud while introducing many different algorithms. One load balancing method introduced in (Umadevi, Pranav, 2017) is to use Predictive Load Balancing Algorithm in both burst and non-burst periods to maintain service quality and minimize energy consumption of cloud network. (Umadevi, Pranav, 2017) also, suggest the use of Right Scale Algorithm (RSA) for consolidating Virtual Machines (VM) into physical machines. Both algorithms use mathematical equations for load balancing and management of cloud resource, however, the research paper does not provide any simulation or real scenario to prove the efficiency of the algorithms. The prediction simply predicts the burst time and

it caps the cloud resources in burst time for better management and the algorithm uses the QoS parameters to add or remove virtual machines in order to meet the QoS goal.

Another predictive load balancing research based on ensemble forecasting (Matthias Sommer, Michael Klink, 2016) uses a reactive overload detection method to predict any overloads. Reactive overload detection uses different CPU parameters such as static threshold (ST) value and when CPU utilization exceeds the static threshold value by 80% or 90% then it will detect it as an overload. This approach also uses various computation intensive statistical calculations to compare CPU utilization values to historical data. After detecting an overload, the research paper suggests the use of CloudSim for forecasting CPU utilization in a theoretical level. The proposed concept in this paper only detects an overload when it already has happened and it proposed prediction method is very computationally intensive. The CloudSim simulation proposed in this paper is only in theoretical level and this paper does not provide any clear simulation results to prove its method and suggest further study in order to improve the load balancing results.

Another load prediction study for energy-aware scheduling (Alexandre, Joanna, Johanne, 2017), suggests training predictors for predicting a load without mentioning any accurate tool for prediction.

The literature review of predictive load balancing algorithms for cloud indicates that all previous literature has used statistical calculations for the prediction that predicts overloads that have already begun to happen. All the previous methods need high computational and centralized approaches that need to be configured and trained for

overloads. For load balancing all previous literature have used mathematical equations which needs high computation power for load balancing and management of cloud resource without offering any simulation or real scenario to prove the efficiency of the introduced algorithms. This literature tends to introduce a new accurate and reliable and less computational approach for cloud load prediction which can accurately predict cloud load. This literature will also investigate all cloud simulation frameworks, in order to find the most accurate simulation platforms.

Next section reviews the different literature on cloud simulations frameworks in order to choose the most accurate cloud simulation framework to generate accurate simulation results.

2.3 Different Cloud Simulation Frameworks

The main step in analyzing a cloud provisioning is to simulate a Cloud computing model, where simulation enables provisioning of a Cloud computing model. To evaluate the performance of a workload model, the simulation software must be able to simulate application models, resources, and policies (Calheiros et al., 2011).

2.3.1 CloudSim Simulation Framework

CloudSim is one of the most popular and well know open-source cloud simulator. CloudSim can simulate large-scale data centers by virtualizing server hosts. CloudSim is capable of provisioning host resources to virtual machines. CloudSim can also model and simulate energy-aware computational resources and dynamic provisioning of simulation elements. In CloudSim simulation can be stopped or resumed at any time. CloudSim can

simulate a cloud computing workload efficiently with a set of applications. CloudSim offers support for system modeling of Cloud systems but also it enables users to simulate system component behavior for resource provisioning such as simulation of virtual machines (VMs). CloudSim can support a single cloud as well as inter-networked clouds, which consists of integrated clouds (Calheiros et al., 2011). CloudSim allows researchers to investigate Cloud resource provisioning and power consumption of data centers.

2.3.2 ICanCloud Simulation Platform

ICanCloud is another cloud simulation platform, which is capable of modeling and simulating many cloud computing systems. The main functionality of iCanCloud is to analyze and predict the trade-offs between performance and cost of different applications. iCanCloud is capable of simulating multiple applications in different hardware while considering information about cost. iCanCloud can model and simulate many different computing architectures with different cloud brokering policies such as customized VMs with different uni-core and multi-core systems.

2.3.3 GreenCloud Simulation Platform

GreenCloud (Jiang Z 2013) simulator is another cloud simulator which focuses on energy power consumption and cost of the physical components of a cloud computing network. With GreenCloud simulator, the workload of cloud computing scenarios and of all its infrastructural elements of a data center can be simulated in order to calculate the total cost of energy consumption.

2.3.4 CloudSched Simulation Platform

CloudSched is also another simulation platform which can model and simulate large Cloud computing environments such as VMs, data centers, and physical machines. CloudSched can also use different resource scheduling policies and algorithms to simulate a network infrastructure (Jiang, 2017).

An extensive study (Wenhong, 2015) on the most popular open-source cloud simulators such as ICanCloud, GreenCloud, CloudSched, and CloudSim has proven that the most efficient cloud simulator for computationally intensive tasks, data interchanges between data centers and internal network communications is CloudSim.

Next section discusses the literature review of workload balancing algorithms.

2.4 Workload Balancing Algorithms

The main goal of load balancing is to achieve the minimum process execution wait time with minimum amount of computational resources. In a perfect load balancing which has a zero execution wait time, all processes are handled simultaneously and there are no wait-times for processing information. Many algorithms were introduced to address workload prediction and workload balancing, the most popular algorithm for workload balancing are Round Robin, Random Algorithm and least loaded algorithm. The following literature below explains the most concept behind the popular workload balancing algorithms.

2.4.1 Round Robin Algorithm

Round-Robin (RR) is scheduling technique that achieves load balancing by assigning equal time quanta to cyclic tasks and processes (Pasha, Agaarwal, & Rastogi, 2014). In RR, the algorithm divides time quanta is into equal slices and assigns with the specific time interval. The time scheduling principle describes the scheduling of the time slides when using the algorithm such that all the nodes are assigned with a quantum and with an operation. All resources are treated as time slices. While RR provides an efficient mechanism for load balancing in terms of meeting peak user demands and providing high quality services, this approach presents significant challenges in bursty workloads (Issawi, Halees, & Radi, 2015).

Bursty workload refers to uneven pattern of data transmission, a common problem in large systems such as web-based applications. The problem with bursty workload is that it can degrade system performance and lead to system unavailability. Burstiness is a major problem in the context of cloud computing given the increasing number of cloud users. Static algorithms such as RR have inherent limitations given that they depend on prior knowledge without considering current state of a node. This means that the algorithm can degrade system performance. The limitations of RR algorithms in environments characterized by bursty workloads indicate the need for enhanced algorithms. The Round Robin (RR) algorithm has two major advantages. Firstly, the algorithm is easy to implement. Secondly, it requires a simple scheduler. Thirdly, the RR algorithm is useful for a small and static system. However, the RR algorithm has its limitations in the context cloud environments. For example, the RR model does not take into consideration the current load on the VMs such as the processing capacity and size of tasks being scheduled (Rajeshkannan & Aramudhan,

2016). Moreover, the static and centralized nature of RR algorithm makes it unsuitable for cloud environments.

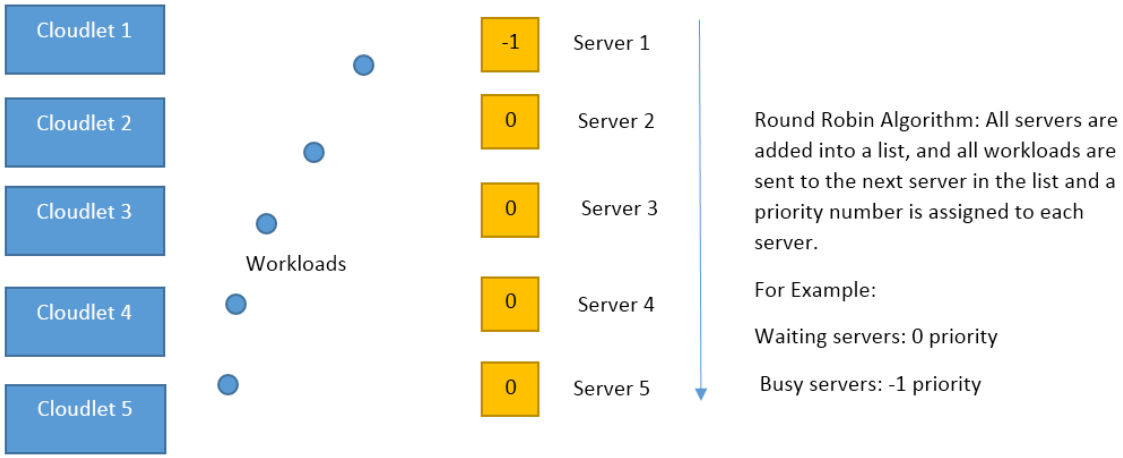

Figure 4. Round Robin Algorithm

Random Algorithm: Random Algorithm connects cloudlets and servers randomly by assigning random numbers to each servers. Unlike Round Robin algorithm, Random algorithm can handle large number of requests and evenly distribute the workload to each node. Similar to RoundRobin algorithm, another advantage of Random algorithm is that it is sufficient for machines with similar Ram and CPU specs. Random algorithm is the most efficient algorithm for peak time traffic and when Cicada cannot detect a reliable prediction, random algorithm can distribute the workload evenly between different VMs.

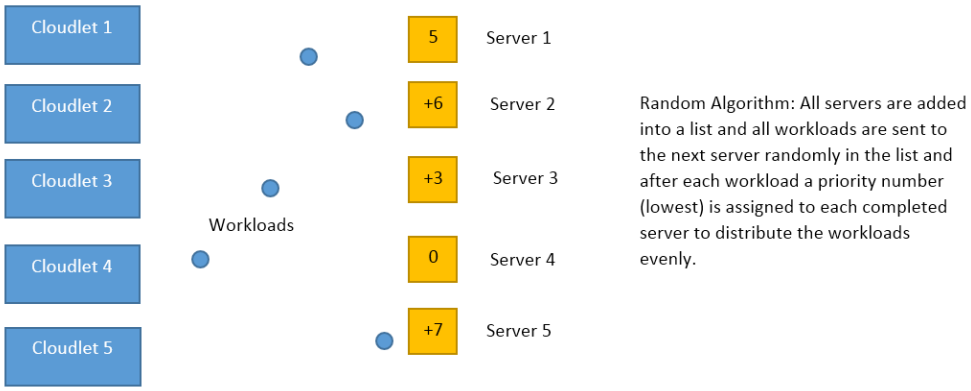

Figure 5. Random Algorithm

In a previous study, Issawi et al. (2015) proposed a novel load-balancing algorithm, Adaptive Algorithm, which can adapt to variations in the request by combining RR algorithm and Random algorithm. The strategic objective of the proposed algorithm is to use RR policy in high workload and deploy the Random policy in low workload. The system comprises a burst detector, which detects workload state. The Random policy activates when the system detects normal burst with a fuzzier supplying candidate list of balanced virtual machines in the datacenter. If the workload state is burst, the fuzzier uses the supplied list of VMs to allocate workload. Simulation experiments using CloudAnalyst showed that the new algorithm decreases the response and processing time (Issawi et al., 2015). These findings suggest the feasibility of using Adaptive Algorithm to achieve improved performance in cloud systems characterized by bursty workloads.

Another problem in cloud computing environments which needs to be addressed is the scheduling of non-preemptive tasks. According to Devi and Uthariaraj (2016), load balancing of non-preemptive tasks on VMs is a vital task-scheduling feature in cloud environments. The objective is to ensure that share load among the

VMs for optimal resource utilization and lower the task completion time. Devi and Uthariaraj (2016) proposed an improved weighted RR algorithm that takes into account the capabilities of all the VMs. To achieve this, the proposed system integrates a static scheduler algorithm that focuses on the initial placement of tasks and a dynamic scheduler that focuses on the load in the configured VMs. The load balancer in the proposed algorithm distributes the load evenly across the VMs. Further experiments to evaluate the performance of the algorithm demonstrated its suitability in both homogeneous and heterogeneous tasks, but with improved performance compared to other RR algorithms.

2.4.2 Throttled Load Balancer (TLB) load balancing algorithm

Throttled Load Balancer (TLB) is another load balancing algorithm which allocates a pre-defined number of cloudlets to a single VM for a specific time (Nema & Edwin, 2016). If the number of requests is larger than the available VM's processing power, the algorithms allocate all incoming requests in a queue and wait for the next available VM. Patel and Rajawat (2015) presented a Throttled-scheduling system that maintains load balancing while enabling efficient task scheduling and resource allocation (See Figure 4). The role of the TLB's load balancer is to maintain a table of the entire candidate VMs and denote their status, whether busy or available. The client or server makes a request to the data center to determine the availability of a suitable VM to perform a recommended task (Patel and Rajawat (2015). The load balancer scans the table of VMs to find a suitable VM to load the data.

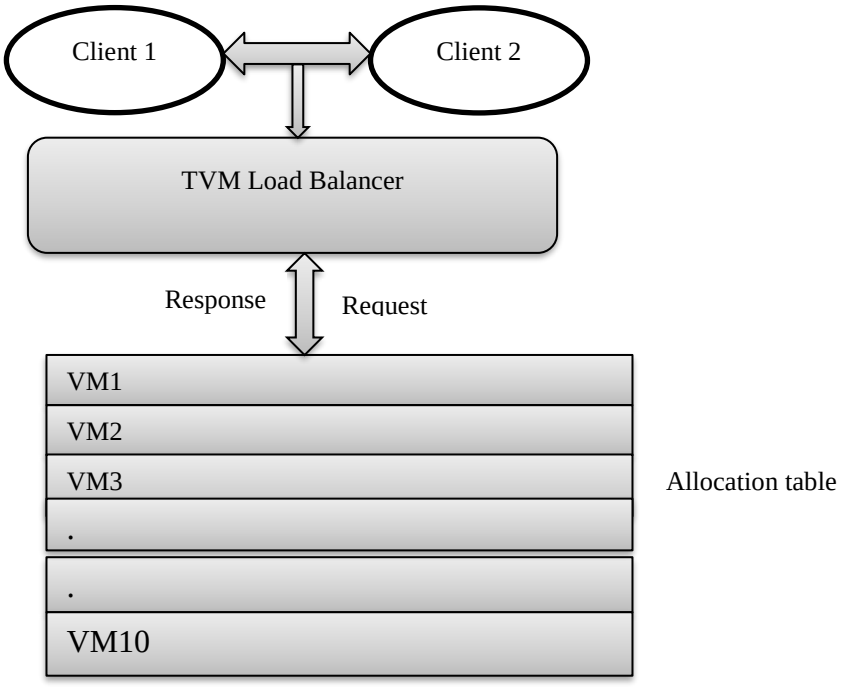

Figure 6. Throttled Algorithm (Source: Patel & Rajawat, 2015)

The proposed model of cloud load balancing combines RR, Throttled, and ESCE. The purpose of the throttled algorithm is to maintain the map table capturing all the VMs (Patel & Rajawat, 2015). Simulation experiments showed the feasibility of the proposed model based on metrics like response time, cost, and request processing time.

2.4.3 Active Monitoring Load Balancer (AMLB) algorithm

Active Monitoring Load Balancer (AMLB) algorithm stores all information related to each VM such as the number of requests and their specific location. When a VM is activated, it is assigned with a VM id and the data controller maintains ids of all VMs and sends the new location of each VM to AMLB. This algorithm comprises four main components: clients, Data Center Controller, the AMLB, and the VMs (Jena & Ahmad, 2013). In order to allocate new VMs, the controller should receive new

requests from the clients. The AMLB parses the index table of candidate VMs to find the least loaded and returns the VM ID to the controller.

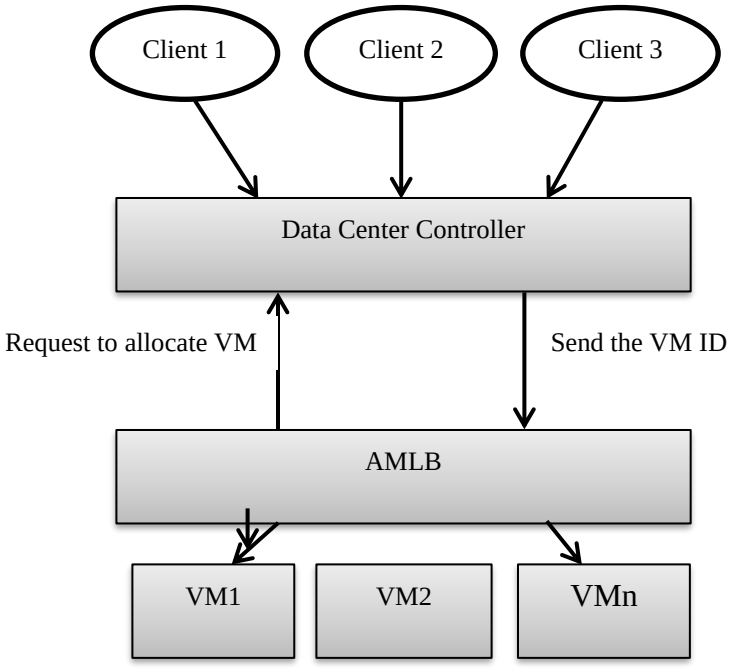

Figure 7. Active Monitoring Load Balancing (Source: Jena & Ahmad, 2013)

2.4.4 The Central Load Balancing Decision Model (CLBDM) algorithm

The Central Load Balancing Decision Model (CLBDM) is an algorithm, which combines the Round Robin Algorithm and session switching of the application layer. CLBDM is the improved version of Round Robin Algorithm and threshold time is added to the algorithm. In this new approach the difference between the client and the node in the cloud is calculated and if this round is greater than the threshold time then the connection between the client and node will be disconnected and that specific task will be moved. Round Robin will be used to determine the new node for this task. That is, CLBDM uses RR but it relies on the measurement of the execution time of tasks in a cloud resource calculated as the duration of connections between server and client (Lee & Jeng, 2011).

2.4.5 Min-Min Load Balancing algorithm

The existing load balancing algorithm for cloud computing differ in a number of ways. The Min-Min Load Balancing algorithm provides remarkable performance regarding task scheduling as it assigns tasks to resources starting with the tasks that require the minimum execution time (Rajeshkannan & Aramudhan, 2016). The Max-Max Load Balancing algorithm is similar to the Min-Min algorithm in that it calculates the execution completion time of tasks. It performs well in a static environment. According to Mathur, Larji, and Goyal (2017), Min-Min algorithms are efficient when the resources require less execution time but the Max-Min algorithm works better when handling tasks with higher time requirements. RR is a static algorithm that does not use task prioritization. The algorithm is unaware of the running time of processes. The Genetic Load Balancing algorithm provides better performance compared to RR as it has vast search space. The Game Theory algorithm works best in public clouds, but it lacks the capacity to predict the arrival of tasks (Rajeshkannan & Aramudhan, 2016).

Algorithm	Category	Parameters	Processing Power	Response Time	Advantages	Disadvantages
Round Robin	Static	Waiting time	Optimal power allocation	381.05 ms (average) No task prioritization	Reduced response time	Poor resource utilization
Max-Min	Static	Waiting time	Uses Minimum Execution Time (MET)	Uses Minimum Execution Time (MET)	Executes tasks with MCT	Starvation
Min-min	Static	Waiting time	Uses Minimum Execution Time (MET)	Uses Minimum Execution Time (MET)	Good performance for multiple small tasks	Starvation

Genetic Algorithm	Dynamic	Process utilization	-	-	Finds optimal solutions	Assumes same priority for all tasks
Stochastic Hill Climbing	Dynamic	Scalability			Better than RR	Improvements required
C-Rule (Custome rule)	Dynamic	predictably	Less computational Power is needed	Uses less VM (less resources)	Historical data can help. Faster load balancing.	Requires reliable prediction

Table 1. Comparison of load balancing algorithms (Rajeshkannan, 2016).

2.4.6 Load Balancing Algorithms in Large-Scale Cloud Computing Service providers

The conventional load balancing algorithms feature severe limitations and drawbacks in cloud environments. In order to address these challenges, researchers have proposed prediction algorithms using genetic algorithms and genetic programming (Wang et al., 2014: Zhou et al., 2016). These algorithms aim to simplify task scheduling in cloud platforms characterized by a large volume of users. In particular, Wang et al. (2014) presented a novel adaptive algorithm to improve on the original adaptive algorithm (AGA). The proposed scheme meets the requirements for inter-nodes load balancing. Simulations to compare the performance of the proposed scheme and the AGA demonstrated the effectiveness and validity of the proposed method in cloud computing. The GA method has advantages related to limited parameter setting and ability to initialize from possible solutions. However, the application of GA comes with drawbacks such as the paucity of fast convergence towards optimal values given that crossover and mutation exist as random events (Wahab, Mexiani, & Atyabi, 2015).

Zhou et al (2016) proposed a method for predicting cloud storage based on a technique called analytic hierarchy process (AHPGD) and hybrid hierarchical genetic algorithm (HHGA). The AHPGD evaluates the load state of server nodes while the role of the HHGA is to train the algorithm to optimize a radial basis function neural network (RBFNN). The centralized load-balancing algorithm consists of three steps: centering nodes to predict the load of service nodes per periodic time (T), calculation of polling weight value for back-end service nodes, and central node allocation using the polling weight value after receiving request tasks. While GA provides capabilities for dynamic load balancing, the main limitation is that they are centralized.

The use of SI algorithms is expected to ameliorate some of the challenges associated with the GA. Hashem, Nashaat, Rizk (2017) proposed a load balancing algorithm based on the Honey Been Behavior. The proposed method is based on the natural foraging behavior of honey bees. That is, in hives, foraging bees give information to other bees about the location of food sources they visit. The allocated tasks update other tasks about the status of VM in the same way bees find food sources. The main goal of the proposed scheme is to distribute workload in a manner to optimize the utilization of cloud resources. The researchers evaluated the performance of the proposed method by simulating on CloudSim (Hashem et al., 2017). In addition, the authors compared the performance of the novel HB technique with the performance of two conventional algorithms: the RR algorithm and the Modified Throttle algorithm. The simulation results showed that the HB method achieves up to 50% increase in the response time compared to other algorithms, with

an average response time of about 60 seconds when executing 1000 tasks. The superior performance is associated with the ability of the HB method to take into account least load and VM availability when assigning tasks. While the Artificial Bee Colony algorithms are typically simple and easy to implement, algorithms using this approach have two inherent disadvantages. Firstly, the need for new fitness tests when adding additional parameters to improve performance makes unsuitable in certain cloud environments (Wahab, Mexiani, & Atyabi, 2015). Secondly, methods that exploit this approach tend to be slow when applied for serial processing.

Nema and Sharma (2016) proposed a similar load balancing technique that uses the honeybee method in cloud computing. The researchers modified the typical HB method as a strategy to achieve balanced load across VMs to maximize throughput. Instead of using tasks, the proposed methodology relies on dynamic loading of instructions to define load distribution. The overarching idea in this method is to recognize idle machines and resources by calculating the load earlier. The cloud partition envisaged in the proposed method can be separated into three steps. The first step is the idle mode in which the system changes to idle status if the inactive nodes exceed. The second step is the normal mode, in which the system changes to normal load operations of the usual nodes exceed. The third step is the overload status in which the system changed to overloaded operation when the overloaded nodes exceed. The main advantage of the proposed method is the performance in terms of execution time.

Previous studies focus on algorithms that load balance a cloud-based on current load, which demands time and high computational power to calculate and load

balance a cloud. These algorithms cannot prevent any over-loads, they can simply react to a current overload scenario. In this paper, the most reliable workload prediction and cloud simulation method will be used to introduce a rule-based algorithm for a predictive workload balancing. This research explores the resource management in the IaaS level and will work based on the prediction of incoming workload rather than load-balancing of current overloads.

2.5 Load Prediction Based On Incoming Network Traffic

There is a relationship between network traffic and processing load of a cloud (Blaszczyszyn, Javonavic, & Karray, 2014). Cloud computing computers receive and forward packets via physical interfaces, typically Layer 2 technologies like the Ethernet. These technologies, or so-called network links, have their characteristics defined in terms of parameters such as bandwidth. Therefore, the amount of network traffic determines the required capacity of the network links due to the nexus between bandwidth and packet forwarding rate. The relationship between the network traffic and processing workload in any region of a network is often expressed using Little's Law, which is derived from queuing systems theory (Blaszczyszyn et al., 2014). The Little's Law states that the average number of items in a queue system is a product of the average rate at which the items arrive and the average time that an item spends in the system. The Little's Law expresses the ratio of the mean traffic demand to the mean number of users in a network segment (Hwang, 2017).

2.6 Impact of the Problem

Finding the most efficient resource allocation strategy depends on resolving the major challenges in dynamic server provisioning. The resolution of these problems will have an impact on the ability of businesses to allocate their resources effectively and to provide an efficient load-balancing alternative for the more cost effective delivery of hosting services.

Three challenges hinder the deployment of dynamic server provisioning policies. The resolution of these challenges is the key to reaching the goals of this research study. The challenges include:

- Uncertainty in workload predictions
- Challenges in simulating network resources and load.
- Challenges of intelligent load balancing in dynamic Internet Hosting Platforms.

2.7 Choosing Load predictor: Cicada Toolkit For

Previously, LaCurts (2014) presented Cicada, which is an end-to-end toolkit software that can predict an applications workload and model the application's workload based on prediction. Cicada can minimize the application completion time when a Cloud provider uses it; it will guarantee specific network performance. To minimize the completion time of applications and load balancing, Cicada minimizes the completion time in load balancing by enabling efficient variation in the underlying network based on the concept of the fastest path (LaCurts, 2014).

Cicada toolkit and an extension called Choreo will be used in this paper for predicting incoming workload (LaCurts, 2014).

2.8 Choosing Cloud Simulator: CloudSim Framework

Calheiros, Ranja, Beloglazov, DeRose and Buyya (2011) suggest that CloudSim has the ability to reliably model and simulate cloud computing infrastructure and services. CloudSim is framework that can be used to simulate and model a cloud computing infrastructure services very efficiently, this is one of the platforms that will be explored in relation to the research questions. Timeline analysis and model building are the two most frequently used methods for predicting concurrent database workloads (Duggan, Cetintemel, Papaemmanouil, & Upfal, 2011). CloudSim has proven to improve QoS requirement of applications by the fluctuation of resource and service demand patterns and CloudSim is a framework that has proven to be a valuable tool in the simulation of cloud environments and the evaluation of resource allocation methods/algorithms (Calheiros et al., 2011). CloudSim extensible simulation toolkit was introduced to simulate a workload efficiently and to model Cloud computing systems and applications. CloudSim not only offers support for system modeling of Cloud systems but also it enables users to simulate system component behavior for resource provisioning such as simulation of virtual machines (VMs). CloudSim supports a single cloud as well as inter-networked clouds, which consists of integrated clouds (Calheiros et al., 2011). CloudSim allows researchers to investigate Cloud resource provisioning and power consumption of data centers. CloudSim has proven to improve QoS requirement of applications by fluctuation of resource and service demand patterns and CloudSim is a framework that has proven to be a valuable tool in the simulation of cloud environments and the evaluation of resource allocation methods/algorithms (Calheiros et al., 2011).

Chapter 3 Presenting C-Rule Algorithm

The core contribution of this paper is a new predictive load balancing of running tasks, for the purpose of resource allocation. Predictive workload balancing enables cloud service providers to prepare their resource allocation for all different scenarios beforehand of any events. We will call the algorithm of allocating resources based on Cicada predictions **C-Rule algorithm.**

The previous chapter introduced the most reliable (in our estimation) load prediction tool called **Cicada** and a reliable cloud simulation framework called **CloudSim, which** allows researchers to investigate Cloud resource provisioning and power consumption of data centers and **its efficiency has been proven in previous research papers. C-Rule algorithm** first predicts workloads during the early stage by a predictor called Cicada. Then, Cicada uses CloudSim framework to simulate the workload balancing by our rule-based algorithm. C-Rule Algorithm focuses on preventing over-loads in a first place rather than balancing current over-loads. In this new approach, a prediction can be achieved in a less than 20 milliseconds (LaCurts, 2014) and with a help of a Cloud simulator, an overload can be in a matter of seconds. If C-Rule algorithm detects any over-loads, CloudSim can find the most accurate resource allocation in a matter of seconds which is faster than all previous algorithms. Resource allocation with a CloudSim requires less computational power than using complex statistical and mathematical formulas for resource allocation.

C-Rule algorithm can achieve the most efficient cloud resource allocation which includes number of host machines and the required number of virtual machines for each host machine with minimal resources. After finding the most system

configuration for a specific workload, C-Rule algorithm will lower number of virtual machines and amount of physical memory for every given task, up until it finds the minimum resource requirement for a specific workload.

C-Rule algorithm needs to receive efficient prediction data and if there are no historical prediction data then CloudSim will use the random algorithm for workload balancing until it receives a reliable workload data. Previous researches have proven that Random algorithm is the most efficient algorithm for peak time traffic and when Cicada cannot detect a reliable prediction, random algorithm can distribute the workload evenly between different VMs. Unlike other algorithms Random Algorithm connects cloudlets and servers randomly by assigning random numbers to each servers and can handle large number of requests and evenly distribute the workload to each node. In a load balancing dependent on the Random algorithm each client can be given list of available servers which can eliminate the need for a centralized broker.

The main purpose of a predictive workload balancing with C-Rule is that, cloud service providers can install SFlow-enabled devices on their cloud network and gather workload data from a traffic link of their cloud network and use C-Rule to simulate the workload on a simulated network based on a specific workload and later increase the amount of workload to test the maximum handling of their workload. This method of the provisioning can also prevent any overprovisioning by finding the minimum amount of computing resources for a workload.

3.1 Workload Prediction Concept Introduced by Cicada and Choero

Cicada uses the data gathered from the SFlow-enabled devices to predict incoming workload.

The data collection process will comprise of the following three steps:

1. Firstly, the SFlow-enabled devices will transmit the samples to a centralized server.
2. Secondly, the centralized server will collect detailed information about the data sample including the IP address, timestamp, and transferred bytes.
3. Thirdly, the aggregate dataset will be exported to Cicada for further estimation.

After completing the first three phases.

1. Cicada imports data from sFlow-enabled device and compares it to historical traffic data generating a workload prediction.
2. Cicada exports the prediction data to a file, which can be exported to Cloud Simulator framework.
3. C-Rule algorithm can compare the prediction data to historical predictions and if it finds any similar overload-scenario in the historical data then it can execute the previous resource allocation policy rather than a new load balancing scenario.

Both Cicada system and CloudSim framework can be installed on a same computer. The following parameters must be transferred from Cicada to CloudSim in order to stablish a reliable simulation.

TaskCPUNum	// Number of the CPU of the task and workload /
cloudletLength	*This variable contain the length of each cloudlet (the actual workload)*
cloudletInputFileSize	This variable will import //input file size from the (task and workloads) section
cloudletOutputSize	//output file from the (task and workloads) section Length of Instruction from the (task and workloads) section

Table 2: Table of Input parameters from Cicada to CloudSim Simulator.

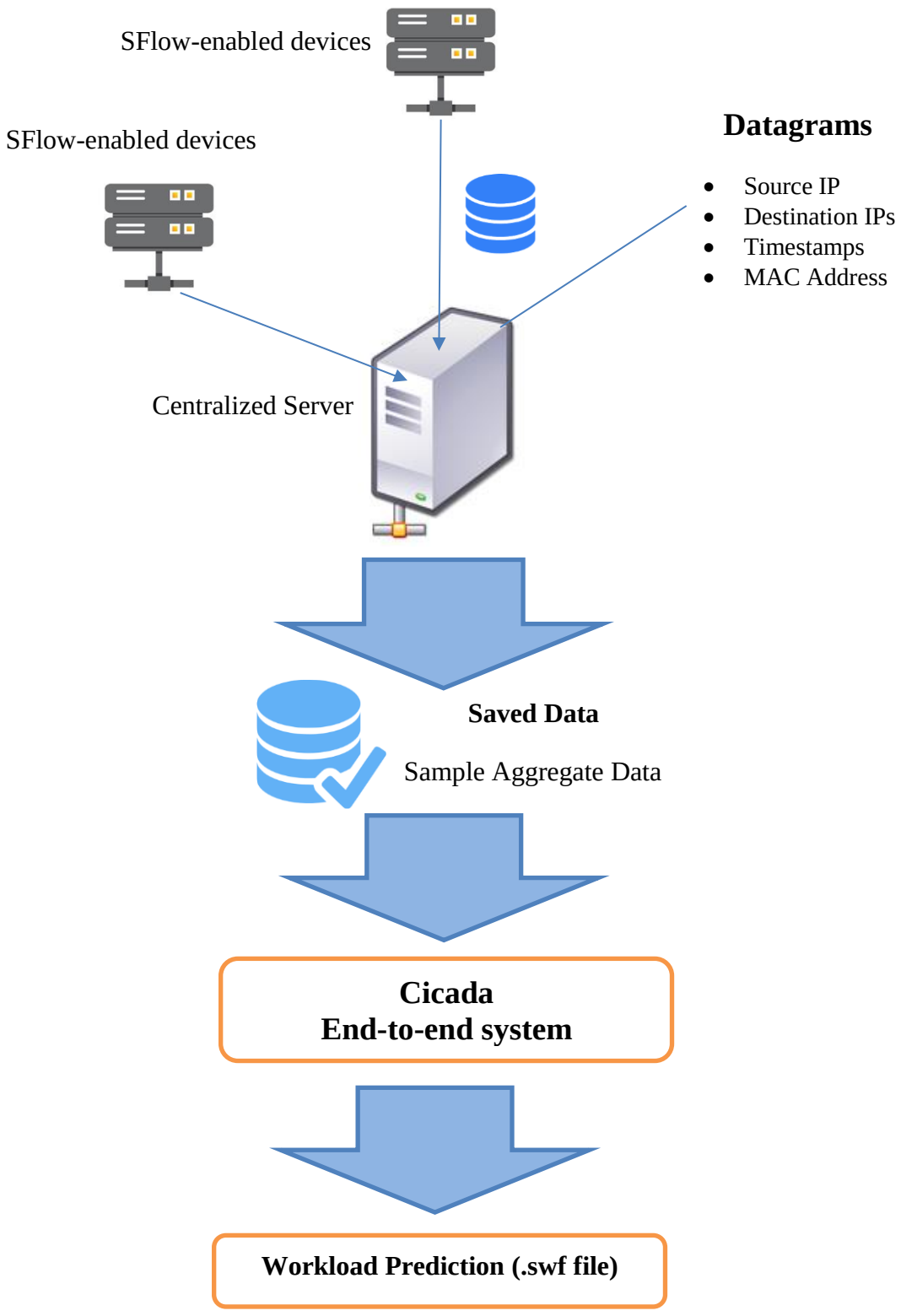

Figure 8: Cicada Data Gathering Diagram

All predictions are transmitted from Cicada to Cloud. CloudSim will run a simulation and detect any possible overloads.

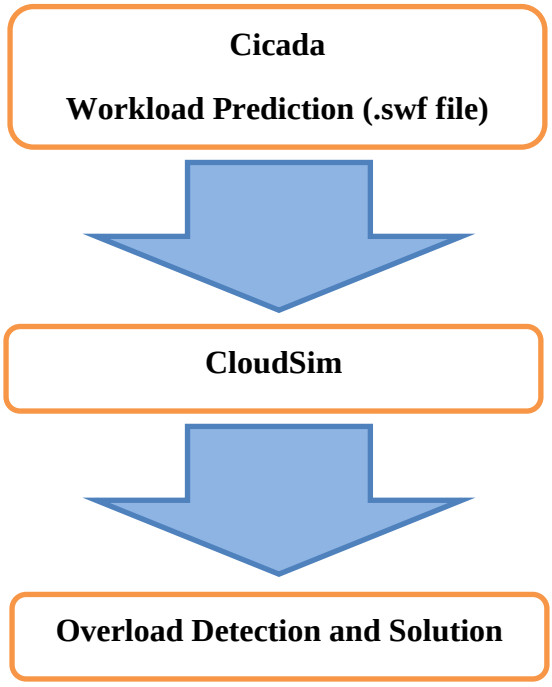

Figure 9: Data Importation from Cicada to CloudSim

If the simulation detects any overloads, it will simulate different scenarios by adding more host machines or virtual machines to achieve zero CPU waiting time. Cloud Simulation can also generate list of CPU wait times for each cloudlet each time that CloudSim adds or removes different cloud resources. All waiting times can stored as set and compared by Paired t-test to the previous data set of CPY waiting times.

In the following example the number of host machines has been increased from 1 hosts to 2 hosts. The sum of the waiting times have been decreased from 200020.38 to 40003.75. To make a statistical comparison, all waiting times will be added into a list and will be compared by Paired t-test.

In the following example the final values for t is 6.98 which indicates there has been a significant change.

Total # of Virtual Machines:		19	19		
Total Number of Host Machines		1 Hosts	2 Hosts		
CloudletID	STATUS	VmID	WaitTime	CloudletID	STATUS	VmID	WaitTime
3	Success	4	**0.00**	7	Success	8	**0.00**
1	Success	2	**0.00**	3	Success	4	**0.00**
0	Success	1	**0.00**	1	Success	2	**0.00**
2	Success	3	**0.00**	5	Success	6	**0.00**
7	Success	4	**5000.06**	11	Success	12	**0.00**
5	Success	2	**5000.62**	2	Success	3	**0.00**
4	Success	1	**5000.75**	8	Success	9	**0.00**
6	Success	3	**5000.86**	0	Success	1	**0.00**
9	Success	2	**10000.88**	9	Success	10	**5000.24**
11	Success	4	**10000.77**	6	Success	7	**5000.13**
10	Success	3	**10000.99**	4	Success	5	**5000.24**
8	Success	1	**10000.99**	10	Success	11	**5000.94**
13	Success	2	**15000.91**	19	Success	8	**5000.94**
14	Success	3	**15001.69**	15	Success	4	**5000.49**
15	Success	4	**15001.55**	17	Success	6	**5000.94**
12	Success	1	**15001.95**	13	Success	2	**5000.84**
17	Success	2	**20001.13**	18	Success	7	**10000.46**
18	Success	3	**20001.91**	14	Success	3	**10001.53**
19	Success	4	**20002.43**	12	Success	1	**10001.31**
16	Success	1	**20002.88**	16	Success	5	**10001.53**

Figure 10: Table of results from CloudSim. Adding a new host machine decreases the total wait-time.

t-Test: Paired Two Sample for Means

	1 Hosts	*2 Hosts*
Mean	10001.0185	4000.48
Variance	52642374.38	14740394
Observations	20	20
Pearson Correlation	0.944917784	
Hypothesized Mean Difference	0	
df	19	
t Stat	**6.989885375**	
P(T<=t) one-tail	5.85249E-07	
t Critical one-tail	1.729132812	
P(T<=t) two-tail	1.1705E-06	
t Critical two-tail	2.093024054	

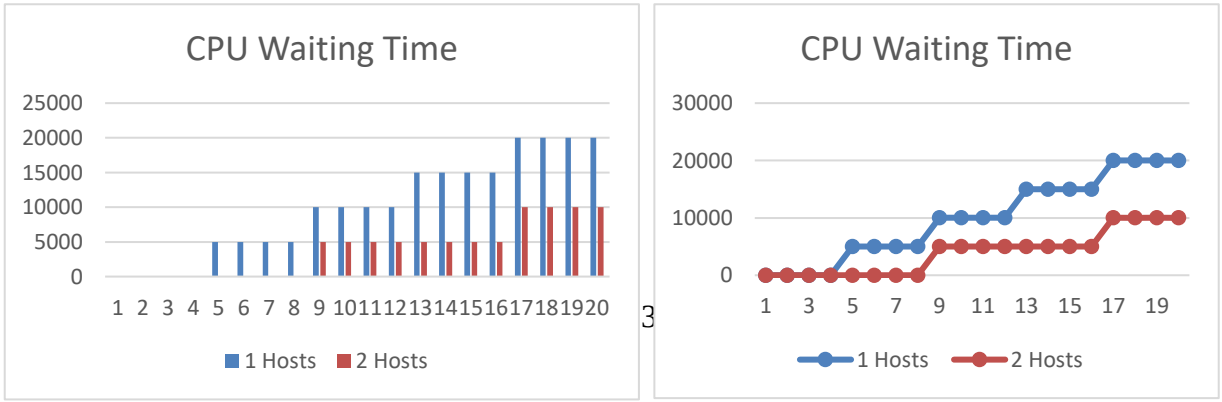

Figure 11: Chart demonstrating effect of adding a new host on total CPU waiting time.

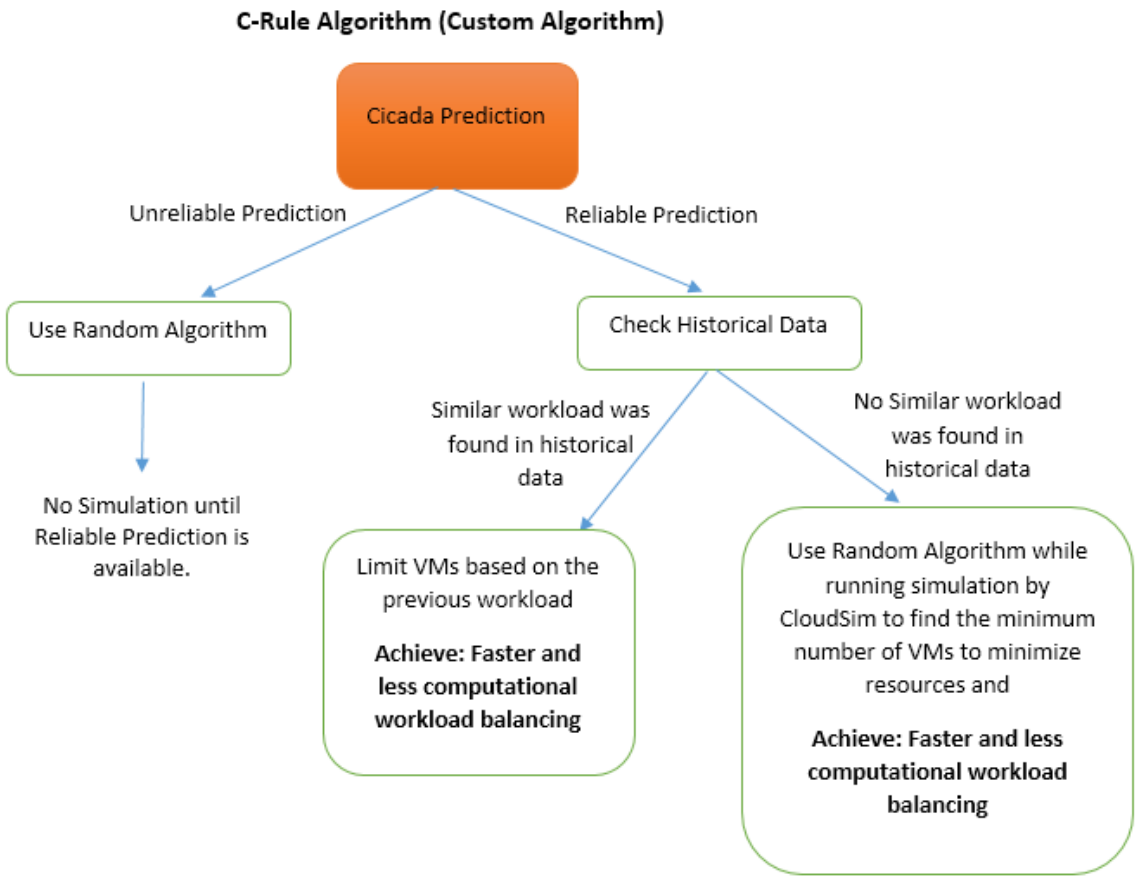

Figure 12. C-Rule Workload balancing diagram

The above diagram demonstrates the overall concept of C-Rule algorithm. Initially, Cicada generates a load prediction and if the prediction is unreliable, then it will use the random algorithm for load balancing until it receives a reliable prediction. The literature in Chapter 2, indicates that Cicada cannot provide any prediction for any burstiness of a workload and the only algorithm that can efficiently handle load balancing of a burstiness workload is the Random algorithm. Random Algorithm connects cloudlets and servers randomly by assigning random numbers to each server. Unlike Round Robin algorithm, Random algorithm can handle a large number of requests and evenly distribute the workload to each node. Similar to the RoundRobin algorithm, another advantage of the Random algorithm is that it is sufficient for machines with similar Ram and CPU specs.

The Random algorithm is the most efficient algorithm for peak time traffic and when Cicada cannot detect a reliable prediction, The Random algorithm can distribute the workload evenly between different VMs.

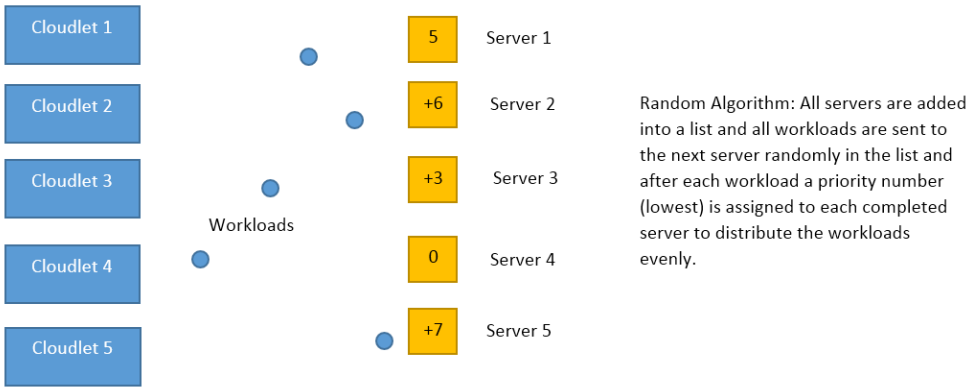

Figure 13: Random Algorithm

In the C-Rule algorithm, any unreliable prediction can be handled with the random algorithm and when a reliable workload arrives, the C-Rule algorithm can compare the incoming workload with historical workloads. If a similar historical workload is detected, the C-Rule can detect whether current resource management policy is suitable for that specific workload. If C-Rule algorithm does not find any historical data related to the incoming workload then it will begin simulating the workload in the CloudSim simulator.

Initially, C-Rule will find the optimal number for physical machines and virtual machines to achieve minimum CPU waiting time. **The ideal CPU waiting time is always zero.** Each time C-rule algorithm can also use the paired t-test to compare the new result to the previous one.

3.2 Improvements of C-Rule Algorithm Compare to All Previous Algorithms

Previously introduced load-balancing algorithms, C-Rule Algorithm focuses on preventing over-loads in a first place rather than balancing current over-loads. C-Rule Algorithm can find a solution for any workloads in a fraction of a second. In most cases, Cicada can make a prediction in less than a 25 milliseconds and it needs a minimum of only 1 hour of historical data to make a prediction. In some cases, the speed of predictions is less than 5 milliseconds. The figure below demonstrates the speed of Cicada's prediction based on the size of the Dataset.

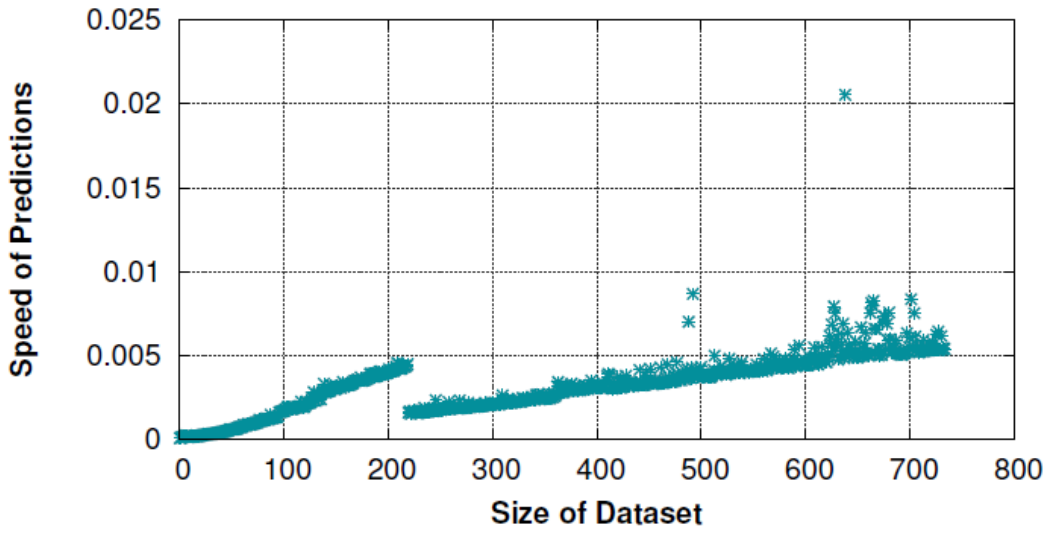

Figure 14: (LaCurts, 2014) Speed of Predictions of Cicada based on the Size of Dataset

CloudSim can also simulate a workload of less than a second depending on the processing power of the centralized server. All previously introduced algorithms need complicated mathematical and statistical computation and demand a very high computational power, where the C-Rule algorithm can require a very small processing power.

After achieving the minimum CPU waiting time. C-Rule algorithm will reduce the amount of resources in the simulations to find the minimum number of required resources for that workload to prevent any over-provisioning.

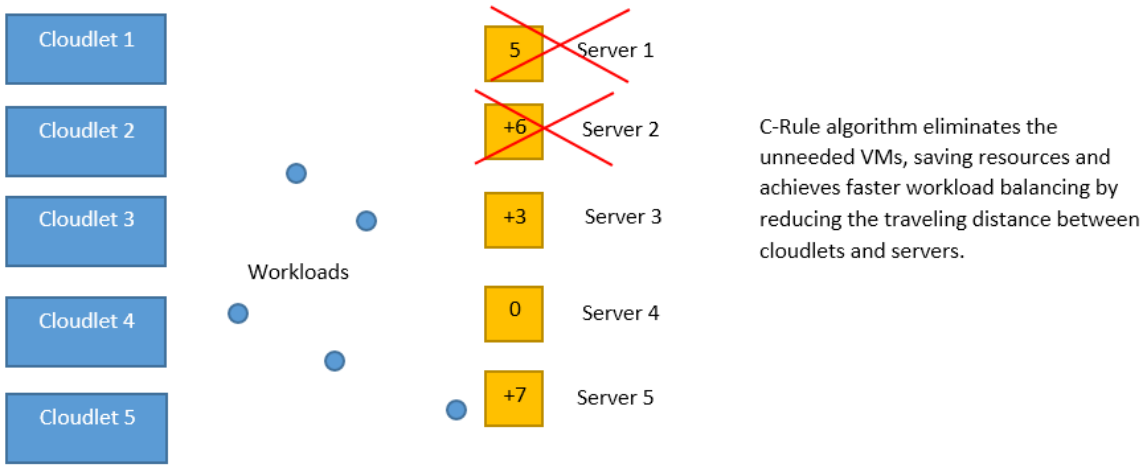

Figure 15: C-Rule eliminates excessive host machines to eliminate over-provisioning.

The following figure is an example of resource reduction by a C-Rule algorithm.

Data Center 1

7 Hosts

CPU	Quad cores (Each core has 1000 mips)
Memory RAM	32 GB of memory
Storage	50 GB of storage (50000)
Bandwidth	2 mbps (2000 kbits/s)

Number of Data center brokers: 1

20 Cloudlets (task and workloads)

Length of Instruction	20000 length of instruction
Input file size	150 kb input file size
Output file size	200 kb output file size
CPU core	1

5 Virtual machines

Storage	20 GB
Memory	1 GB RAM
Virtual CPU	1 (each with 1000 mips CPU speeds)
Cloudlets Scheduler	Timeshared

Host reduction

Data Center 1

5 Hosts

CPU	Quad cores (Each core has 1000 mips)
Memory RAM	32 GB of memory
Storage	50 GB of storage (50000)
Bandwidth	2 mbps (2000 kbits/s)

Number of Data center brokers: 1

20 Cloudlets (task and workloads)

Length of Instruction	20000 length of instruction
Input file size	150 kb input file size
Output file size	200 kb output file size
CPU core	1

5 Virtual machines

Storage	20 GB
Memory	1 GB RAM
Virtual CPU	1 (each with 1000 mips CPU speeds)
Cloudlets Scheduler	Timeshared

Figure 16: Resource Reduction by C-Rule algorithm

The following chart demonstrates a simulation result for host reduction in a successful load balancing. CPU waiting time is zero whether service provider uses 7 host machines or 5 hot machines.

Total # of Virtual Machines: 20 Total Number of Host Machines			20 7 Hosts	20 5 Hosts			
CloudletID	STATUS	VmID	WaitTime	CloudletID	STATUS	VmID	WaitTime
3	Success	9	**0.00**	7	Success	8	**0.00**
1	Success	14	**0.00**	3	Success	4	**0.00**
0	Success	15	**0.00**	1	Success	2	**0.00**
2	Success	12	**0.00**	5	Success	6	**0.00**
7	Success	5	**0.00**	11	Success	12	**0.00**
5	Success	1	**0.00**	2	Success	3	**0.00**
4	Success	11	**0.00**	8	Success	9	**0.00**
6	Success	3	**0.00**	0	Success	1	**0.00**
9	Success	8	**0.00**	9	Success	10	**0.00**
11	Success	13	**0.00**	6	Success	7	**0.00**
10	Success	10	**0.00**	4	Success	5	**0.00**
8	Success	17	**0.00**	10	Success	11	**0.00**
13	Success	18	**0.00**	19	Success	8	**0.00**
14	Success	20	**0.00**	15	Success	4	**0.00**
15	Success	6	**0.00**	17	Success	6	**0.00**
12	Success	7	**0.00**	13	Success	2	**0.00**
17	Success	4	**0.00**	18	Success	7	**0.00**
18	Success	16	**0.00**	14	Success	3	**0.00**
19	Success	2	**0.00**	12	Success	1	**0.00**
16	Success	19	**0.00**	16	Success	5	**0.00**

Table 3: Final result of resource reduction after achieving a zero processing wait-time.

3.3 Webhosting aspects of C-Rule Algorithm.

Webhosting service providers often use Virtual Machines (VMs) to provide service to their customers. Webhosting platforms use dynamic clouds to offer cloud services which require a very efficient load balancing algorithm. In web hosting services, the objective is to ensure that share load among the VMs for optimal resource utilization and lower the task completion time. C-Rule can improve web hosting service level by preventing any under-providing and over-provisioning. C-rule algorithm can simulate the workload in a

matter of seconds and find the minimum amount of memory needed for each virtual machines reducing the memory ram usage.

To implement the C-Rule algorithm in web hosting environment, the space-sharing algorithm will be used to reuse memory space between different virtual machines. The following literature, explains the difference between the space-sharing algorithm and time-sharing algorithms. Little's Law is helpful in explaining the concept of load balancing in distributed environments in which task scheduling remains an open problem. The task scheduler maps tasks to allocated resources. Scheduling refers to the process of controlling the order in which a computing system performs work (Kumar & Mishra, 2015). Task scheduling occurs via two primary modes: space-shared and time-shared. In space-shared scheduling, the system executes a cloudlet to completion before releasing the VM to execute another cloudlet. In time-shared scheduling, multiple cloudlets may execute in different time slots on the same VM (Kumar & Mishra, 2015). Therefore, space-sharing algorithms may reuse memory space but time-sharing algorithms may involve sharing execution power. More importantly, load balancing in cloud environments can use a space-sharing algorithm or time-sharing algorithms.

3.1 Comparison of Workload Prediction and Efficiently Level of C-Rule Algorithm.

The following parameters must be transferred from Cicada to CloudSim in order to stablish a reliable simulation.

`TaskCPUNum`	`// Number of the CPU of the task and workload /`

cloudletLength	*This variable contain the length of each cloudlet (the actual workload)*
cloudletInputFileSize	This variable will import //input file size from the (task and workloads) section
cloudletOutputSize	//output file from the (task and workloads) section Length of Instruction from the (task and workloads) section

Figure 17: Table of parameters required for a CloudSim simulation.

C-Rule algorithm can initially use the *cloudletLength* variable *to find the* most similar workload and use the Paired t-test to compare multiple historical workloads and choose the most similar workload for a simulation. C-Rule algorithm requires accurate load predictions to operate efficiently and any inaccurate prediction can also lower the efficient of C-Rule algorithm. For example Cicada cannot provide any efficient load balancing in burstiness time or when the number of connected machines are less than 5.

CHAPTER 4 METHODOLOGY AND DATA SAMPLE

This chapter discusses the methodology for cloud workloads prediction and simulation for developing a load balancing algorithm. The main goal of the methodology is to develop an efficient load balancing algorithm which would require less processing power. The first step for predictive workload balancing is to find and define a reliable method of prediction.

4.1 Methodology Step 1: Instruments of Prediction

The first step of the methodology is to prove that whether Cicada and Choero extension can generate reliable prediction data. Choero is a network measurement extension to Cicada which allows users to perform network measurement without access to the network infrastructure (LaCurts, 2014). Workload prediction of a cloud demands a highly efficient tool that can predict workload of a cloud network within a few minutes and can be compatible with Cicada. This paper presents a network measurement extension to Cicada called Choreo. This network measurement tool estimates TCP throughput by simply analyzing packet trains. In this paper, data collected from hypothetical deployed networks will be used to simulate CloudSim.

To demonstrate that Cicada can detect whether its predictions are reliable and can generate an alert in case if the prediction is unreliable. The following literature will describe the fundamental of Cicada and Choero extension.

4.1.1 Introduction to Cicada and its reliability

According to the research, state-of-the-art [3] Cicada's workload prediction algorithm has success rate up to 90% for static placement because for workload prediction, Cicada load balances applications traffic by measuring both spatial and temporal variations of every application. Cicada is capable of workload prediction for different type and class of cloud applications and can provide a reliable feedback indicating whether the prediction is incorrect or the prediction is reliable saving users from uncertainty. Cicada is capable of improving the average completion time of application from 8% to 14% per cent in some cases up to 61%. All these improvements are achieved without any modifications of network infrastructure.

According to paper (LaCurts, 2014), Cicada predictions for networks less than 5 virtual machines are unreliable. To eliminate the possibility of any unreliable predictions networks with more than 15 VMs will be considered for predictions and any with less than 15 will not be considered. In the data sample of this paper, a minimum of 20 virtual machines will be used for each host machines. Another research paper objects the reliability of Cicada and suggests that Cicada predictions can be unreliable during the peak hours (Katrina 2014).

In all cases, Cicada can provide a reliable feedback of its own predictions (LaCurts 2014) and detect whether or not its predictions are reliable. The conditions mentioned above address the first research question.

It is important to understand that Cicada predictions focus on individual pairs, which require VM-to-VM traffic matrices in the cloud infrastructure. It is expected that this type of data from IaaS clouds might provide rich applications compared to

data centers or other cloud computing environments (LaCurts, 2014). The dataset (sFlow) will be collected from a hypothetical data gathered from Cicada predicted data. The dataset collection process requires sFlow-enabled network switches to gather datagrams that come with the information such as the source and destination IPs, sample timestamp, and MAC address.

The data collection process entails three primary steps:

i. The sFlow-enabled switches send the samples to a centralized server
ii. The centralized server collects sample information such as the source and destination IPs, timestamp, and transferred bytes
iii. The database stores sample aggregate data

4.2 Methodology Step 2: Sample Data

The second step of the methodology the gathering of sufficient workload data get enrage a simulation in CloudSim and different data of real parallel workloads are available from this link http://www.cs.huji.ac.il/labs/parallel/workload/logs.html, however, due to the fact that the mentioned data demands a massive and a complicated simulation, in this, a hypothetical set of small data will be used for CloudSim to generate a feasible algorithm for workload balancing. The hypothetical data consists of a predicted amount of workload gathered by Cicada.

4.3 Methodology Step 3: Importation of sample data

The 3rd step of the methodology is to import sample data into CloudSim. This can be achieved by exporting the data into a file and later that file can be imported by

CloudSim simulator. All predictions of Cicada and Choero can be exported into a .swf file. At the end of the prediction, that file can be imported into CloudSim.

4.4 Methodology Step 4: Instrument for simulation

The 4th step of the methodology is to simulate the data in the CloudSim simulator. In this paper, CloudSim will be used to predict and simulate the network.

Generally, CloudSim refers to a set of simulation tools that can assess the performance of cloud services within a controllable or a rule-based environment. The simulation toolkit provides classes for describing users, applications, computational purposes, resources management, and data centers in order to facilitate the management and utilization of these components. That is, CloudSim provides a system and a behavior modelling cloud computing environments (Calheiros, Ranjan, De Rose, & Buyya, 2009). The simulation of cloud environments and applications can facilitate the evaluation of performance in dynamic and distributed environments. The main advantages of simulation include enhanced flexibility in terms of defining cloud configurations, ease of customization, and the cost savings that come with customized simulations.

The CloudSim framework is a layered architecture comprising of three layers of components. The lowest layer comprises of the SimJava simulation engine, which implements the core functionalities for enabling simulation of queuing and processing of events and enabling creation and communication among system components. The next layer is the GridSim layer, which consists of a toolkit for modelling Grid networks and components. This layer comprises of two sets of components: grid services such as datasets, grid information service, resource allocation, workload

traces, and core elements such as resources, traffic generator and the network (Calheiros et al., 2009). The next layer is the CloudSim, which extends the functionalities of the GridSim layer. The CloudSim enables the modelling and simulation functions in virtualized cloud environments. It also manages the execution of the core entities such as Virtual Machines (VM), applications, data centers and hosts during simulation. The CloudSim layer comprises user interface structures, VM services, cloud services, and cloud resources. The top layer is the User Code layer, a simulation stack supporting the configuration of hosts, VMs, and applications (Calheiros et al., 2009).

4.5 Methodology Step 5: Importation of data to CloudSim

The fifth step of the methodology is to import data to Cloud for simulation. The sample workload data can be imported to CloudSim with the following Java command

The main java class of the simulations must be modified to throw a FileNotFoundEception.

```
private static List<Cloudlet> createCloudLets() throws FileNotFoundException{
        //The following command reads the sample swf file.
WorkloadFileReader workloadFileReader = new
WorkloadFileReader("C:\CodeRespository\ HPC2N-2002-2.1-cln2.swf", 1);
//The following command can generate cloudlets from imported workload file
cloudletList = workloadFileReader.generateWorkload();
return cloudletList;}
```

4.6 Methodology Step 6: Simulation of Data In CloudSim

The sixth step of the methodology is the simulation of data in CloudSim, (LaCurts., 2014) proposed a toolkit for simulating cloud computing systems by supporting system and behavior modeling of VMs and other cloud system components. This study envisages using the CloudSim simulation of the cloud-based data to model the VM component. CloudSim will be configured to use spaces-shared policy for VM workload balancing which all resources will be shared equally among all VMs. One the simulation of workload is completed, the results will be printed and saved. Afterwards, CloudSim will assign the specific lower amount of resources such as VMs for the clouds based on the historical workload and simulation data. The intent will be to model the behavior of cloud computing environments. Sample workload data will be imported into the CloudSim using Java command and simulation conditions will establish the desired parameters.

CloudSim will be configured to use a lower number of VMs, memory and other resources. The result of the first and the second simulation will be compared to achieve an optimal level of resources for a workload.

4.7 Methodology Step 7: Implementing Historical Data

The 7th step of the methodology is to use the C-Rule to analyze whether it is beneficial to use prediction and historical data to assign resources to a data center. This phase will entail developing a generic space-shared algorithm (C-algorithm) using historical data to simulate the workload and assign a lower amount of resources. The premise is that the proposed algorithm will achieve lower computational cost and faster load balancing for the same amount of resources as previously predicted.

The conditions for generating a reliable simulation by CloudSim the parameters of the CloudSim architecture must be clearly defined in the following order.

Step 1: Initializing CloudSim process

Step 2: Creating Data Centers, VM Allocation Policy and Scheduling

Step 3: Creating Broker

Step 4: Creating Cloudlets by Defining the Workload

Step 5: Creating VMs and Defining the Task Scheduling Algorithm

Step 6: Starting the Simulation

Step 7: Printing the Results of the Simulation

It is important to note that CloudSim cannot support the priority of cloud services (Jun-Kwon., 2012) and can only use precomputed topology to apply network delay (Jun-Kwon., 2012), CloudSim will be unreliable when there is a need to calculate priority of cloud services and when there are not precomputed information regarding of network delay.

4.8 Summary

Chapter 4 introduced the sample data and design principle for a predictive workload balancing. Chapter 4 also covered the first research question. An efficient method of workload prediction and simulation was explained in this chapter. Chapter 4 addressed all the possibilities of an unreliable workload prediction and an unreliable cloud simulation. The source of the input data and the programing language for cloud simulation were provided in this chapter. To ease the simulation process, a small set of hypothetical data and hypothetical size datacenter will be simulated in chapter 5. Chapter 4 covered research question 1,2 and the first hypothesis were answered in chapter 4. In the next chapter a sample data center and a hypothetical workload data will be simulated in CloudSim. In the initial step, simulation will use a space-shared policy to simulate and balance a hypothetical workload. After that C-Rule algorithm will limit the number of VMs and resources to achieve a faster result for the simulation and the final result will be compared in the chapter 6.

Chapter 5 will explain the required Java programing steps for CloudSim. Scheduling algorithm will be defined in step 6 of programing while creating virtual machine, task scheduling algorithm will be defined. Chapter 4 introduced all previous algorithms and discussed a new approach for scheduling algorithm (C-Rule) algorithm.

CHAPTER 5 Implementation

5.1 Introduction to the fundamentals of Cloud Simulator.

In this chapter, we introduce the operation of the cloud simulator. The sample data center for this paper will be simulated and the initial result of the simulation will be printed at the end of the chapter.

The simulation envisaged in this study utilizes CloudSim simulation tool, a generalized framework that allows a controllable environment for the simulation and modelling of application performance (Cloud Computing and Distributed Systems (Clouds) Laboratory).

The use of CloudSim simulator in this study is justified because the simulator allows developers to focus on the design issues specific to a particular system, without concerns over the cloud-based infrastructure and services. According to Calheiros et al., the CloudSim toolkit can perform both system and behavior modelling of cloud components like virtual machines, data centers, and policies for resource provisioning (23). In particular, simulation of the cloud computing environments can provide insights into the performance of cloud components. The main advantages of cloud simulations are, improved flexibility in application configurations, ease of use and enhanced customization, as well as the cost savings achieved by reusing the models created during the design phase.

CloudSim provides a robust tool for simulating datacenters because the toolkit provides the basic classes for defining datacenters (Buyya et al. 2009). A data center refers to a remote facility comprising of a set of networked servers that an organization uses for the data processing and or storage to meet the organization's IT

needs. The term ‘datacenter’ describes the facility’s physical and the virtual infrastructure. CloudSim supports data center modelling and simulation because datacenters behave like Infrastructure as a Service (IaaS) provider; that is, the datacenter accepts VMs requests from brokers and generates the VMs in hosts.

The following steps were used to simulate the data center in CloudSim:

Step 1: Initial step for data center simulation: The initial step in CloudSim for data center simulation is to create a data center called Cloud Information Service (CIS). CIS is a registry of all the data center resources that are available on the cloud. Each resource contains a data center and each data center contains one or multiple hosts. A cloud host describes a network of servers dedicated to providing hosting services. Each host must contain virtual machines (VMs), specialized software programs or OSs that exhibit the behavior of physical computers. The CloudSim simulation process requires three fundamental parameters in order to initialize: the number of users, calendar instance, and the traceflag value. The data center instance is created via “*CreateDataCenter”,* which creates the datacenter characteristics.

Step 2: Registering a date center in the CIS registry: After creating a data center, the second step is to register the data center in the CIS registry. Data centers have unique characteristics defined by the hardware configuration of hosts within the data center.

Step 3: Creating and submitting each task to a data center: The third step is to create and submit each task to the data center **Broker** which keeps a list of cloudlet(s)**.** Data centers are assigned to a broker, which directly interacts with

data center and cloudlet(s). Essentially, the Broker conceals the VM management including the VM creation and the submission of cloudlets. It also implements policies for VM selection for running cloudlets and datacenter selection for executing submitted VMs. No sub initializations were done at the Datacenterbroker instance stage.

Step 4: Allocation of policies and importing cloudlets: The fourth step is the allocation of policies

- The datacenter uses the **VM allocation** policy to allocate machine
- The hosts uses **VM scheduling** policy
- **Cloudlet Scheduler** policy involved processing of cloudlets on the VMs
- This is the stage where cloudlets can be imported to the simulator from a .swf file, however in this paper hypothetical cloudlets will used to achieve this goal.

Step 5: Defining characteristics of VMs and resource allocation algorithm: The fifth step of the policy simulation is to define the characteristics of VMs and define the algorithm which will be used for the resource allocation. In this step we define whether each VM will be Time or Space Shared. The following parameters of VMs will be defined in this step disk size, memory ram, VM mips, VM bandwidth and number of CPU for each virtual machine.

Step 6: Start of the simulation and requesting results: The sixth step is to start the simulation, request the results as a list and then stop the simulation, this can be done by the following three command lines:

```
 CloudSim.startSimulation();
List<Cloudlet> Finalresults = dcb.getCloudletReceivedList();
```

```
CloudSim.stopSimulation();
```

Step 7: Printing results of the simulation: The seventh step is to print the result of the simulation

The following diagram demonstrates the procedures and operations behind the workload balancing. All resources are registered in Cloud Information Service and then sent to the broker. All cloudlets will be registered in the broker directly.

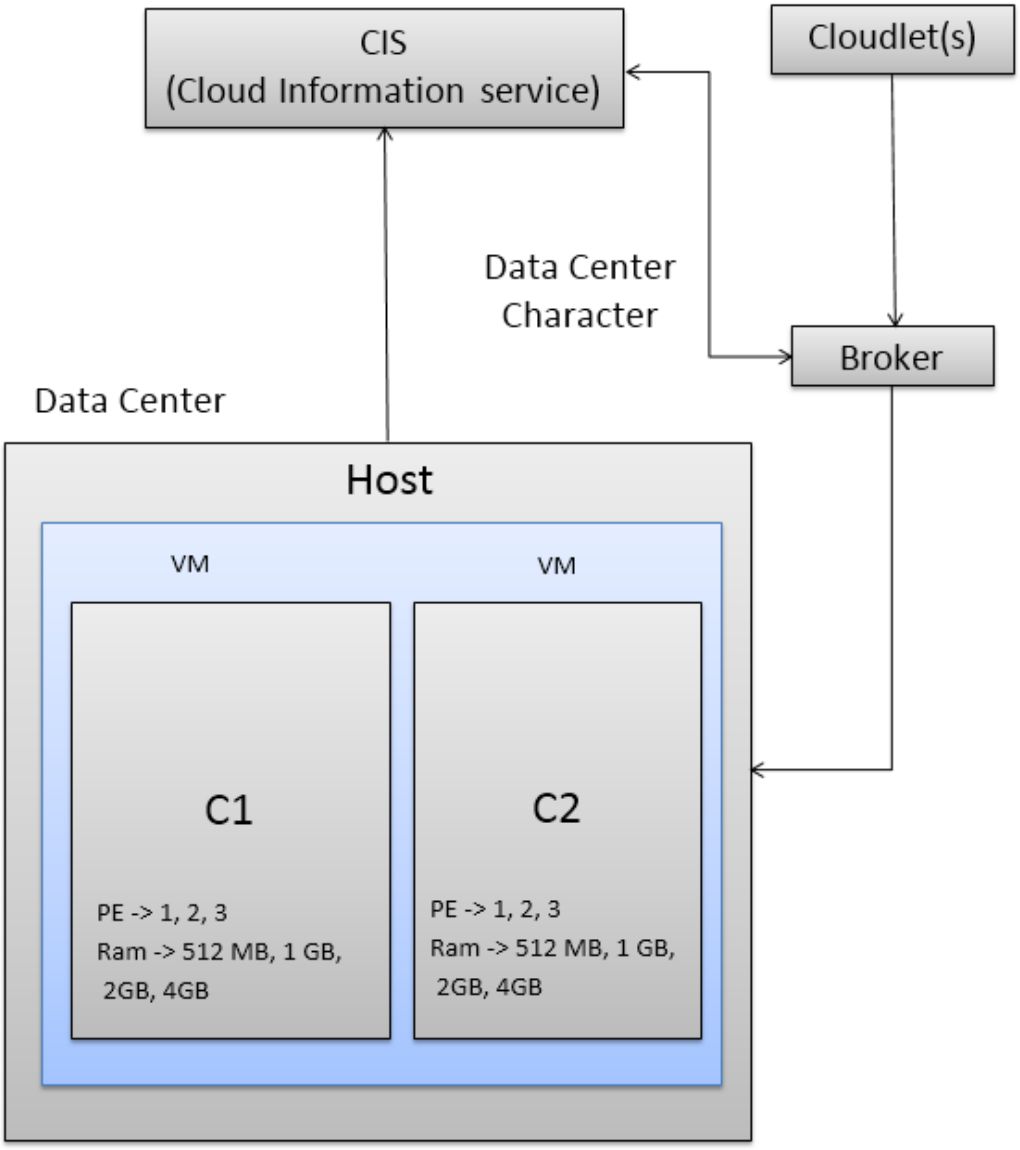

Figure 18 CloudSim DataCenter 1 Diagram

5.2 CloudSim Programming and Implementation:

This section entails a description of the main steps of java code implementations required for datacenter simulation. In the present study, the following data center model was simulated in the CloudSim.

********** Length of Instruction from the (Task and Workloads) Section **********
Cloudlet Length 5000000
# of Task CPU: 1
Input file size: 100000
Output file size: 300000
********** Each Host **********
Memory RAM: 32 GB
Bandwidth: 8 Mbps
Storage (SSD/HDD): 2000 GB
********** Each VM (Virtual Machine) **********
Disk Disk : 20 GB
Memory RAM : 1 Gb
VM MIPS: 1000
VM Bandwidth: 1 Mbps
# of VM CPU: 1

Each Hosts

CPU	Quad cores (Each core has 1000 mips)
Memory RAM	32 GB of memory
Storage	50 GB of storage (50000)
Bandwidth	8 mbps (2000 kbits/s)

Number of Data center brokers: 1

20 Cloudlets (task and workloads)

Length of Instruction	5000000 length of instruction
Input file size	100000 kb input file size
Output file size	300000 kb output file size
CPU core	1

5 Virtual machines

Storage	20 GB
Memory	1 GB RAM
Virtual CPU	1 (each with 1000 mips CPU speeds)
Cloudlets Scheduler	Timeshared

Figure 19 Data center 1 specifications table

Step 1:

5.2.1 Initializing CloudSim process

The first step is the CloudSim initialization process, a stage that involves initialize the simulation toolkit for the experiments. In this step a CloudSim was initialized using the *CloudSim.init()* method. The initialization approach entails defining the number of users based on the *num-user* parameter, determining the simulation start time defined by the *calendar* parameter, and using the *trace_ flag* parameter to track the simulation events. That it, the initialization method takes in the number of cloud users in integer value and an instance of calendar and a Boolean value.

CloudSim.init(num _user, calendar, trace _flag);

The screen will display the following message upon executing step 1:

"Initializing..."

Step 2:

5.2.2 Creating data centers, VM allocation policy and scheduling

The second step in the CloudSim implementation entails creating the data center, which consists of the physical hosts that represent the computing resources. In this stage, at least one data center should be created using the *createDatacenter* (*"dataCentre_name"*) method. This approach to datacenter creation returns a datacenter object. Notably, this step entails defining the characteristics of the datacenter as well as articulating the VM allocation policy and scheduling policy. Firstly, the MIPS of each CPU were provisioned by the following command:

```
PeProvisionerSimple ProcessorProvisioner = new
PeProvisionerSimple(1000);
Secondly, each CPU core was designed to have its own ID. After
CPU ID assignment, each CPU was added into a list.
Pe CPUcore1 = new Pe(0, ProcessorProvisioner);
        Pe CPUcore2 = new Pe(1, ProcessorProvisioner);
        Pe CPUcore3 = new Pe(2, ProcessorProvisioner);

        Pe CPUcore4 = new Pe(3, ProcessorProvisioner);
peList.add(CPUcore1);
        peList.add(CPUcore2);
        peList.add(CPUcore3);
        peList.add(CPUcore4);
After CPU core ID assignment, each Host was provisioned and
added to a list using the following command:
Host host1 = new Host(0, new
RamProvisionerSimple(HostRAM), new
        BwProvisionerSimple(HostBandwidth),
        HostStorage, peList, new
VmSchedulerSpaceShared(peList));
```

Host host2 = new Host(1, new
RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
HostStorage, peList, new
VmSchedulerSpaceShared(peList));
hostlist.add(host1);
hostlist.add(host2);
In part, Datacenter Characteristics were defined and all Virtual machines lists and time zone and cost of each part added to the data center.
DatacenterCharacteristics acharacteristic = new
DatacenterCharacteristics(architecture, os, vmm, hostlist, timeZone,
EachComputercostPerSec, costPerMem,
costPerStorage, costPerBw);
LinkedList<Storage>SANstroage = new LinkedList<Storage>();
Datacenter aDatacenter=null;
try {//exception starts
aDatacenter = new Datacenter("DataCenter1", acharacteristic, new VmAllocationPolicySimple(hostlist), SANstroage, 1);
} catch (Exception e1) {e1.printStackTrace();}//end of exception
returnaDatacenter;

Step 3:

5.2.3 Creating Broker

To create a broker, the name of the data center was defined without creating any spaces. The following command was used to create a broker for the Datacenter.

DatacenterBroker dcb=new DatacenterBroker

("DataCenterBroker1");

To handle the exception created by the

DatacenterBroker dcb=null;

try {dcb = new DatacenterBroker("DataCenterBroker1");} catch

(Exception e) {e.printStackTrace();}

Step 4:

5.2.4 Creating Cloudlets by Defining the Workload

In this step, each cloudlet was generated randomly and cloudlet length, a long variable, which represented the length of instruction from the (task and workloads) section. The utilization type was full as shown in the following command:

```
List<Cloudlet> cloudletList = new ArrayList<Cloudlet>();
UtilizationModelFull  fullUtilize =new UtilizationModelFull();
```

In this part, random cloudlets will be generated with a random size and each cloudlet will contain a user id and will be added to the list.

```
for (int cloudletId =0;cloudletId <20;cloudletId ++) {
Random r= new Random ();
Cloudlet anewcloudlet = new Cloudlet(cloudletId ,
cloudletLength+r.nextInt(1000), cloudletLength,
cloudletInputFileSize, cloudletOutputSize,
fullUtilize,fullUtilize,fullUtilize);
anewcloudlet.setUserId(dcb.getId());
cloudletList.add(anewcloudlet); }
```

The following java command can import Cicada data into the simulator from a .swf file. The main java class must be modified to throw a FileNotFoundEception.

```
Private static List<Cloudlet> createCloudLets() throws FileNotFoundException{

//The following command reads the sample swf file.

WorkloadFileReader workloadFileReader = new

WorkloadFileReader("C:\CodeRespository\ HPC2N-2002-2.1-cln2.swf", 1);

//The following command can generate cloudlets from imported workload file

cloudletList = workloadFileReader.generateWorkload();

return cloudletList;}.
```

In this paper we used hypothetical workload data for our simulation and the above commands will be in the program.

Step 5:

5.2.5 Creating VMs and Defining the Task Scheduling Algorithm

This part uses the task-scheduling algorithm proposed in this paper. The first part of the simulation used the space-shared algorithm and a comparison was conducted between the C-algorithm and the space-shared algorithm.

```
//********** Task scheduling algorithm will be defined here **********

//**************************Algorithm*********************************

            for(intvmId =0;vmId<NumberOfVM; vmId ++)
                    {Vm VirtualMachine= new Vm(vmId, dcb.getId(),
                       VMmips,
            VCPU,
            VMRam,
            VMbandwidth,
                            vmdiskSize,
            VMM,
                new CloudletSchedulerSpaceShared());
            vmList.add(VirtualMachine);}
            dcb.submitCloudletList(cloudletList);
            dcb.submitVmList(vmList);
            //**********************End of Algorithm
            **********************************
```

Step 6:

5.2.6 Starting the Simulation

```
//  Part 6.0: Simulation starts in part 6 even simulation (engine)
```

In step 6, the simulation process started in a simulation engine.

```
CloudSim.startSimulation();
List<Cloudlet>Finalresults = dcb.getCloudletReceivedList();
CloudSim.stopSimulation();
```

Step 7:

5.2.7 Printing Results of the Simulation

In the final stage, the results of the simulation are printed on the screen by the following command:

```
intcloudletNo=0;
     DecimalFormat TwoDecimalFormatter = new
DecimalFormat("#0.00");
for (Cloudlet c: Finalresults) {
Log.printLine("Result of cloudlet No:"+cloudletNo);
     Log.printLine("****************************");
     Log.printLine("ID:" +c.getCloudletId()  +  " , VM:"
+c.getVmId()+1+ " ,
status:" +" , Excecution Time:
"+TwoDecimalFormatter.format(c.getActualCPUTime())+" ,
start:
"+TwoDecimalFormatter.format(c.getExecStartTime())+" , Stop:
 "+TwoDecimalFormatter.format(c.getFinishTime()) );
Log.printLine("****************************");
cloudletNo++;}
                }//END OF THE PUBLIC STAT
********** Length of Instruction from the (Task and Workloads) Section **********
Cloudlet Length 5000000
# of Task CPU: 1
Input file size: 100000
Output file size: 300000
********** Each Host ****************************
Memory RAM : 32 GB
Bandwidth : 8 Mbs
Storage (SSD/HDD): 2000 GB
********** Each VM (Virtual Machine) **********
Disk Disk : 20 GB
Memory RAM : 1 Gb
VM Mips : 1000
VM Bandwidth : 1 Mbs
# of VM CPU : 1
```

The result of the simulation with a `Space shared Algorithm is the following`

CloudletID	STATUS	VmID	WaitTime	StartTime	FinishTime
6	Success	7	0.00	0.10	5000.12
5	Success	6	0.00	0.10	5000.23
4	Success	5	0.00	0.10	5000.44
3	Success	4	0.00	0.10	5000.63
7	Success	8	0.00	0.10	5000.76
0	Success	1	0.00	0.10	5000.87
1	Success	2	0.00	0.10	5000.98

2	Success	3	0.00	0.10	5001.09
13	Success	6	5000.13	5000.23	10000.48
14	Success	7	5000.02	5000.12	10001.06
8	Success	1	5000.77	5000.87	10001.21
12	Success	5	5000.34	5000.44	10001.32
9	Success	2	5000.88	5000.98	10001.52
11	Success	4	5000.53	5000.63	10001.63
15	Success	8	5000.66	5000.76	10001.63
10	Success	3	5000.99	5001.09	10001.82
16	Success	1	10001.11	10001.21	15001.29
17	Success	2	10001.42	10001.52	15001.89
19	Success	4	10001.53	10001.63	15002.20
18	Success	3	10001.72	10001.82	15002.68

The Total Execution Waiting Time of this Algorithm is: **80010.10**

Table 4: Initial Simulation Result for Space Shared Algorithm

Each host of the datacenter model has the following specifications.

- Quad cores of 1000 MIPS
- 32GB memory RAM
- 20GB storage
- 8 mbps bandwidth.
- A single datacenter Broker
- 20 Cloudlets measuring 5000000 in instruction length
- 100000 kb file input size
- 300000 kb output file size.

Further, the database model comprises of

- 20 VMs
- Storage capacity of 20GB
- 1GB memory RAM
- 1 virtual CPU installed with 1000 MIPS speed
- Space-sharing Cloudlets Scheduler.

5.3 Summary

This chapter introduced and explained the concept for java programing required for importation of workload prediction and simulation in CloudSim simulator. The sample workload data for the CloudSim was introduced in this chapter. This chapter explained the fundamental requirement for Cloud simulation and the sample data center was simulated and the result of the sample simulation was included at the end of the chapter.

CHAPTER 5 RESULTS

In this chapter we discuss the sample data and the final findings of this paper.

For our simulation a hypothetical prediction data will be imported into the CloudSim simulator and C-Rule algorithm introduced in chapter 3 and 4 will detect whether to use previous data and parameters or to run a new simulation. Cicada extension has the ability to detect whether that its prediction is reliable or not. The C-Rule algorithm will first determine whether the predictions of Cicada is reliable. In case if Cicada's prediction is reliable then C-Rule algorithm will use historical data and the result of previous predictions to determine the parameters and the amount of the resources for workload balancing. If the algorithm does not find any similar historical data then it will run a simulation for find the minimum amount of cloud resources required for that workload to reach CPU waiting time of zero.

To prove the efficiency of C-Rule algorithm will use spaced-sharing policy for VM scheduling in order to simulate a hypothetical prediction data generated by Cicada extension.

In this part we simulate Datacenter 1 model from chapter 4 by a space shared Algorithm with the following specifications.

```
********** Length of Instruction from the (Task and Workloads) Section
**********
Cloudlet Length 5000000
# of Task CPU: 1
Input file size: 100000
Output file size: 300000
********** Each Host **************************
Memory RAM : 32 GB
Bandwidth : 8 Mbs
Storage (SSD/HDD): 2000 GB
********** Each VM (Virtual Machine) **********
```

```
Disk Disk : 20 GB
Memory RAM : 1 Gb
VM Mips : 1000
VM Bandwidth : 1 Mbs
# of VM CPU : 1
```

6.1 Result 1: Without C-Rule Algorithm

The result of the simulation with a `Space shared Algorithm is the following`

CloudletID	STATUS	VmID	WaitTime	StartTime	FinishTime
6	Success	7	0.00	0.10	5000.12
5	Success	6	0.00	0.10	5000.23
4	Success	5	0.00	0.10	5000.44
3	Success	4	0.00	0.10	5000.63
7	Success	8	0.00	0.10	5000.76
0	Success	1	0.00	0.10	5000.87
1	Success	2	0.00	0.10	5000.98
2	Success	3	0.00	0.10	5001.09
13	Success	6	5000.13	5000.23	10000.48
14	Success	7	5000.02	5000.12	10001.06
8	Success	1	5000.77	5000.87	10001.21
12	Success	5	5000.34	5000.44	10001.32
9	Success	2	5000.88	5000.98	10001.52
11	Success	4	5000.53	5000.63	10001.63
15	Success	8	5000.66	5000.76	10001.63
10	Success	3	5000.99	5001.09	10001.82
16	Success	1	10001.11	10001.21	15001.29
17	Success	2	10001.42	10001.52	15001.89
19	Success	4	10001.53	10001.63	15002.20
18	Success	3	10001.72	10001.82	15002.68

The Total Execution Waiting Time of this Algorithm is: **80010.10**

Table 5: New Result of simulation with space shared algorithm with the following

In the data above we can clearly see that the ExcTime of this Algorithm is 812:43 seconds, the above approach requires additional time due to the fact that first a simulation needs to be completed first and then the system must decide in a small amount of time whether or not to use this approach or to use a different workload balancing approach.

In the next paragraph the C-Algorithm will check whether any similar workload balancing has been done previously or not. Once the C-Algorithm finds a similar workload balancing scenario, then it will use the previous result.

In this step C-algorithm will reduce the total number of virtual machines to **7 virtual machines**, **total memory** for each machine will be reduced to **2 Gb of RAM** and the total disk size of each virtual machine to will be reduced to **20 Gb.**

6.2 Result 2: With C-Rule algorithm

The following result is generated after the simulation.

```
 ************** Length of Instruction from the (Task and Workloads) Section
**************
Cloudlet Length 5000000
# of Task CPU: 1
Input file size: 100000
Output file size: 300000
************** Each Host ****************************
Memory RAM : 32 GB
Bandwidth : 8 Mbs
Storage (SSD/HDD): 2000 GB
************** Each VM (Virtual Machine) **************
Disk Disk : 20 GB
Memory RAM : 1 Gb
VM Mips : 1000
VM Bandwidth : 1 Mbs
# of VM CPU : 1
**********************
System Architecture: 64 bits
OS Type: Ubuntu Server
VM Software: VMware
************** C-Algorithm Load Balancer ****************************
This program will find the optimal amount of VMs and Hosts
Press Enter to continue
********************************
NEW SIMULATION
Total Number of Virtual Machine : to be used: 20
Total Number of Host Machine to be used: 1
Initialising...
Starting CloudSim version 3.0
DataCenter1 is starting...
DataCenterBroker1 is starting...
Entities started.
0.0: DataCenterBroker1: Cloud Resource List received with 1 resource(s)
0.0: DataCenterBroker1: Trying to Create VM #1 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #2 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #3 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #4 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #5 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #6 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #7 in DataCenter1
```

```
0.0: DataCenterBroker1: Trying to Create VM #8 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #9 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #10 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #11 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #12 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #13 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #14 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #15 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #16 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #17 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #18 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #19 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #20 in DataCenter1
[VmScheduler.vmCreate] Allocation of VM #5 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #6 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #7 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #8 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #9 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #10 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #11 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #12 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #13 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #14 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #15 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #16 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #1 failed by MIPS
0.1: DataCenterBroker1: VM #1 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #2 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #3 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #4 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: Creation of VM #5 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #6 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #7 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #8 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #9 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #10 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #11 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #12 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #13 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #14 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #15 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #16 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #17 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #18 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #19 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #20 failed in Datacenter #2
0.1: DataCenterBroker1: Sending cloudlet 0 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 1 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 2 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 3 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 4 to VM #1
```

```
0.1: DataCenterBroker1: Sending cloudlet 5 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 6 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 7 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 8 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 9 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 10 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 11 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 12 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 13 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 14 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 15 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 16 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 17 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 18 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 19 to VM #4
5000.376: DataCenterBroker1: Cloudlet 1 received
5000.602: DataCenterBroker1: Cloudlet 0 received
5000.715: DataCenterBroker1: Cloudlet 2 received
5000.914: DataCenterBroker1: Cloudlet 3 received
10000.585: DataCenterBroker1: Cloudlet 5 received
10001.053: DataCenterBroker1: Cloudlet 6 received
10001.276: DataCenterBroker1: Cloudlet 4 received
10001.59: DataCenterBroker1: Cloudlet 7 received
15000.954: DataCenterBroker1: Cloudlet 9 received
15001.144: DataCenterBroker1: Cloudlet 10 received
15001.316: DataCenterBroker1: Cloudlet 8 received
15001.936000000002: DataCenterBroker1: Cloudlet 11 received
20001.472: DataCenterBroker1: Cloudlet 14 received
20001.703: DataCenterBroker1: Cloudlet 12 received
20001.813: DataCenterBroker1: Cloudlet 13 received
20002.058999999997: DataCenterBroker1: Cloudlet 15 received
25002.263999999996: DataCenterBroker1: Cloudlet 17 received
25002.373999999993: DataCenterBroker1: Cloudlet 16 received
25002.48399999999: DataCenterBroker1: Cloudlet 18 received
25002.593999999986: DataCenterBroker1: Cloudlet 19 received
25002.593999999986: DataCenterBroker1: All Cloudlets executed. Finishing...
25002.593999999986: DataCenterBroker1: Destroying VM #1
25002.593999999986: DataCenterBroker1: Destroying VM #2
25002.593999999986: DataCenterBroker1: Destroying VM #3
25002.593999999986: DataCenterBroker1: Destroying VM #4
DataCenterBroker1 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
DataCenter1 is shutting down...
DataCenterBroker1 is shutting down...
Simulation completed.
Simulation completed.
Result of cloudlet No
************************
CloudletID STATUS VmID WaitTime StartTime FinishTime
1 Success 2 0.00 0.10 5000.38
0 Success 1 0.00 0.10 5000.60
2 Success 3 0.00 0.10 5000.72
3 Success 4 0.00 0.10 5000.91
```

```
5 Success 2 5000.28 5000.38 10000.58
6 Success 3 5000.61 5000.72 10001.05
4 Success 1 5000.50 5000.60 10001.28
7 Success 4 5000.81 5000.91 10001.59
9 Success 2 10000.48 10000.58 15000.95
10 Success 3 10000.95 10001.05 15001.14
8 Success 1 10001.18 10001.28 15001.32
11 Success 4 10001.49 10001.59 15001.94
14 Success 3 15001.04 15001.14 20001.47
12 Success 1 15001.22 15001.32 20001.70
13 Success 2 15000.85 15000.95 20001.81
15 Success 4 15001.84 15001.94 20002.06
17 Success 2 20001.71 20001.81 25002.26
16 Success 1 20001.60 20001.70 25002.37
18 Success 3 20001.37 20001.47 25002.48
19 Success 4 20001.96 20002.06 25002.59
The Total Excecution Waiting Time of this Algorithm is : 200017.91
***************** Unsuccessfull Load Balancing *****************
***************** Total excecution wait time is not zero yet ******************
There is a total waiting time of :200017.91
Used 1 host(s) Machines
***************** Simulation will restart now ***********************
Press Enter to Restart the simulation with: 2 Host(s)
*********************************
NEW SIMULATION
Total Number of Virtual Machine : to be used: 20
Total Number of Host Machine to be used: 2
Initialising...
Starting CloudSim version 3.0
DataCenter1 is starting...
DataCenterBroker1 is starting...
Entities started.
0.0: DataCenterBroker1: Cloud Resource List received with 1 resource(s)
0.0: DataCenterBroker1: Trying to Create VM #1 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #2 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #3 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #4 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #5 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #6 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #7 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #8 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #9 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #10 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #11 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #12 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #13 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #14 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #15 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #16 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #17 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #18 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #19 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #20 in DataCenter1
[VmScheduler.vmCreate] Allocation of VM #9 to Host #1 failed by MIPS
```

```
[VmScheduler.vmCreate] Allocation of VM #9 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #10 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #10 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #11 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #11 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #12 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #12 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #13 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #13 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #14 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #14 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #15 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #15 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #16 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #16 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #2 failed by MIPS
0.1: DataCenterBroker1: VM #1 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #2 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #3 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #4 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #5 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #6 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #7 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #8 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: Creation of VM #9 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #10 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #11 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #12 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #13 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #14 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #15 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #16 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #17 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #18 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #19 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #20 failed in Datacenter #2
0.1: DataCenterBroker1: Sending cloudlet 0 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 1 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 2 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 3 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 4 to VM #5
0.1: DataCenterBroker1: Sending cloudlet 5 to VM #6
0.1: DataCenterBroker1: Sending cloudlet 6 to VM #7
0.1: DataCenterBroker1: Sending cloudlet 7 to VM #8
0.1: DataCenterBroker1: Sending cloudlet 8 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 9 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 10 to VM #3
```

```
0.1: DataCenterBroker1: Sending cloudlet 11 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 12 to VM #5
0.1: DataCenterBroker1: Sending cloudlet 13 to VM #6
0.1: DataCenterBroker1: Sending cloudlet 14 to VM #7
0.1: DataCenterBroker1: Sending cloudlet 15 to VM #8
0.1: DataCenterBroker1: Sending cloudlet 16 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 17 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 18 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 19 to VM #4
5000.121: DataCenterBroker1: Cloudlet 6 received
5000.231000000001: DataCenterBroker1: Cloudlet 5 received
5000.436000000001: DataCenterBroker1: Cloudlet 4 received
5000.626: DataCenterBroker1: Cloudlet 3 received
5000.763: DataCenterBroker1: Cloudlet 7 received
5000.8730000000005: DataCenterBroker1: Cloudlet 0 received
5000.983000000001: DataCenterBroker1: Cloudlet 1 received
5001.093000000002: DataCenterBroker1: Cloudlet 2 received
10000.480000000001: DataCenterBroker1: Cloudlet 13 received
10001.059000000001: DataCenterBroker1: Cloudlet 14 received
10001.209: DataCenterBroker1: Cloudlet 8 received
10001.319000000001: DataCenterBroker1: Cloudlet 12 received
10001.515000000001: DataCenterBroker1: Cloudlet 9 received
10001.625000000002: DataCenterBroker1: Cloudlet 11 received
10001.625000000002: DataCenterBroker1: Cloudlet 15 received
10001.821000000002: DataCenterBroker1: Cloudlet 10 received
15001.288: DataCenterBroker1: Cloudlet 16 received
15001.893: DataCenterBroker1: Cloudlet 17 received
15002.198: DataCenterBroker1: Cloudlet 19 received
15002.675000000001: DataCenterBroker1: Cloudlet 18 received
15002.675000000001: DataCenterBroker1: All Cloudlets executed. Finishing...
15002.675000000001: DataCenterBroker1: Destroying VM #1
15002.675000000001: DataCenterBroker1: Destroying VM #2
15002.675000000001: DataCenterBroker1: Destroying VM #3
15002.675000000001: DataCenterBroker1: Destroying VM #4
15002.675000000001: DataCenterBroker1: Destroying VM #5
15002.675000000001: DataCenterBroker1: Destroying VM #6
15002.675000000001: DataCenterBroker1: Destroying VM #7
15002.675000000001: DataCenterBroker1: Destroying VM #8
DataCenterBroker1 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
DataCenter1 is shutting down...
DataCenterBroker1 is shutting down...
Simulation completed.
Simulation completed.
Result of cloudlet No
**************************
CloudletID STATUS VmID WaitTime StartTime FinishTime
6 Success 7 0.00 0.10 5000.12
5 Success 6 0.00 0.10 5000.23
4 Success 5 0.00 0.10 5000.44
3 Success 4 0.00 0.10 5000.63
7 Success 8 0.00 0.10 5000.76
0 Success 1 0.00 0.10 5000.87
```

```
1 Success 2 0.00 0.10 5000.98
2 Success 3 0.00 0.10 5001.09
13 Success 6 5000.13 5000.23 10000.48
14 Success 7 5000.02 5000.12 10001.06
8 Success 1 5000.77 5000.87 10001.21
12 Success 5 5000.34 5000.44 10001.32
9 Success 2 5000.88 5000.98 10001.52
11 Success 4 5000.53 5000.63 10001.63
15 Success 8 5000.66 5000.76 10001.63
10 Success 3 5000.99 5001.09 10001.82
16 Success 1 10001.11 10001.21 15001.29
17 Success 2 10001.42 10001.52 15001.89
19 Success 4 10001.53 10001.63 15002.20
18 Success 3 10001.72 10001.82 15002.68
The Total Excecution Waiting Time of this Algorithm is : 80010.10
***************** Unsuccessfull Load Balancing *****************
***************** Total excecution wait time is not zero yet *****************
There is a total waiting time of :80010.10
Used 2 host(s) Machines
***************** Simulation will restart now ***********************
Press Enter to Restart the simulation with: 3 Host(s)
**********************************
NEW SIMULATION
Total Number of Virtual Machine : to be used: 20
Total Number of Host Machine to be used: 3
Initialising...
Starting CloudSim version 3.0
DataCenter1 is starting...
DataCenterBroker1 is starting...
Entities started.
0.0: DataCenterBroker1: Cloud Resource List received with 1 resource(s)
0.0: DataCenterBroker1: Trying to Create VM #1 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #2 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #3 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #4 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #5 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #6 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #7 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #8 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #9 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #10 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #11 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #12 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #13 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #14 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #15 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #16 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #17 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #18 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #19 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #20 in DataCenter1
[VmScheduler.vmCreate] Allocation of VM #13 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #13 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #13 to Host #3 failed by MIPS
```

```
[VmScheduler.vmCreate] Allocation of VM #14 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #14 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #14 to Host #3 failed by MIPS

[VmScheduler.vmCreate] Allocation of VM #15 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #15 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #15 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #16 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #16 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #16 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #3 failed by MIPS
0.1: DataCenterBroker1: VM #1 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #2 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #3 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #4 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #5 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #6 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #7 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #8 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #9 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #10 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #11 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #12 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: Creation of VM #13 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #14 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #15 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #16 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #17 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #18 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #19 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #20 failed in Datacenter #2
0.1: DataCenterBroker1: Sending cloudlet 0 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 1 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 2 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 3 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 4 to VM #5
0.1: DataCenterBroker1: Sending cloudlet 5 to VM #6
0.1: DataCenterBroker1: Sending cloudlet 6 to VM #7
0.1: DataCenterBroker1: Sending cloudlet 7 to VM #8
0.1: DataCenterBroker1: Sending cloudlet 8 to VM #9
0.1: DataCenterBroker1: Sending cloudlet 9 to VM #10
0.1: DataCenterBroker1: Sending cloudlet 10 to VM #11
0.1: DataCenterBroker1: Sending cloudlet 11 to VM #12
```

```
0.1: DataCenterBroker1: Sending cloudlet 12 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 13 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 14 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 15 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 16 to VM #5
0.1: DataCenterBroker1: Sending cloudlet 17 to VM #6
0.1: DataCenterBroker1: Sending cloudlet 18 to VM #7
0.1: DataCenterBroker1: Sending cloudlet 19 to VM #8
5000.168000000001: DataCenterBroker1: Cloudlet 11 received
5000.2970000000005: DataCenterBroker1: Cloudlet 7 received
5000.407000000001: DataCenterBroker1: Cloudlet 6 received
5000.626: DataCenterBroker1: Cloudlet 4 received
5000.736000000001: DataCenterBroker1: Cloudlet 5 received
5000.846000000001: DataCenterBroker1: Cloudlet 3 received
5000.846000000001: DataCenterBroker1: Cloudlet 9 received
5000.953: DataCenterBroker1: Cloudlet 1 received
5000.953: DataCenterBroker1: Cloudlet 2 received
5001.063000000001: DataCenterBroker1: Cloudlet 0 received
5001.063000000001: DataCenterBroker1: Cloudlet 10 received
5001.173000000002: DataCenterBroker1: Cloudlet 8 received
10000.752: DataCenterBroker1: Cloudlet 19 received
10001.015000000001: DataCenterBroker1: Cloudlet 17 received
10001.125000000002: DataCenterBroker1: Cloudlet 16 received
10001.125000000002: DataCenterBroker1: Cloudlet 14 received
10001.235000000002: DataCenterBroker1: Cloudlet 15 received
10001.345000000003: DataCenterBroker1: Cloudlet 18 received
10001.455000000004: DataCenterBroker1: Cloudlet 13 received
10001.565000000004: DataCenterBroker1: Cloudlet 12 received
10001.565000000004: DataCenterBroker1: All Cloudlets executed. Finishing...
10001.565000000004: DataCenterBroker1: Destroying VM #1
10001.565000000004: DataCenterBroker1: Destroying VM #2
10001.565000000004: DataCenterBroker1: Destroying VM #3
10001.565000000004: DataCenterBroker1: Destroying VM #4
10001.565000000004: DataCenterBroker1: Destroying VM #5
10001.565000000004: DataCenterBroker1: Destroying VM #6
10001.565000000004: DataCenterBroker1: Destroying VM #7
10001.565000000004: DataCenterBroker1: Destroying VM #8
10001.565000000004: DataCenterBroker1: Destroying VM #9
10001.565000000004: DataCenterBroker1: Destroying VM #10
10001.565000000004: DataCenterBroker1: Destroying VM #11
10001.565000000004: DataCenterBroker1: Destroying VM #12
DataCenterBroker1 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
DataCenter1 is shutting down...
DataCenterBroker1 is shutting down...
Simulation completed.
Simulation completed.
Result of cloudlet No
*************************
CloudletID STATUS VmID WaitTime StartTime FinishTime
11 Success 12 0.00 0.10 5000.17
7 Success 8 0.00 0.10 5000.30
6 Success 7 0.00 0.10 5000.41
```

```
4 Success 5 0.00 0.10 5000.63
5 Success 6 0.00 0.10 5000.74
3 Success 4 0.00 0.10 5000.85
9 Success 10 0.00 0.10 5000.85
1 Success 2 0.00 0.10 5000.95
2 Success 3 0.00 0.10 5000.95
0 Success 1 0.00 0.10 5001.06
10 Success 11 0.00 0.10 5001.06
8 Success 9 0.00 0.10 5001.17
19 Success 8 5000.20 5000.30 10000.75
17 Success 6 5000.64 5000.74 10001.02
16 Success 5 5000.53 5000.63 10001.13
14 Success 3 5000.85 5000.95 10001.13
15 Success 4 5000.75 5000.85 10001.24
18 Success 7 5000.31 5000.41 10001.35
13 Success 2 5000.85 5000.95 10001.46
12 Success 1 5000.96 5001.06 10001.57
The Total Excecution Waiting Time of this Algorithm is : 40005.08
***************** Unsuccessfull Load Balancing *****************
***************** Total excecution wait time is not zero yet ******************
There is a total waiting time of :40005.08
Used 3 host(s) Machines
****************** Simulation will restart now ***********************
Press Enter to Restart the simulation with: 4 Host(s)
*********************************
NEW SIMULATION
Total Number of Virtual Machine : to be used: 20
Total Number of Host Machine to be used: 4
Initialising...
Starting CloudSim version 3.0
DataCenter1 is starting...
DataCenterBroker1 is starting...
Entities started.
0.0: DataCenterBroker1: Cloud Resource List received with 1 resource(s)
0.0: DataCenterBroker1: Trying to Create VM #1 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #2 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #3 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #4 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #5 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #6 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #7 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #8 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #9 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #10 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #11 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #12 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #13 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #14 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #15 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #16 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #17 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #18 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #19 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #20 in DataCenter1
```

```
[VmScheduler.vmCreate] Allocation of VM #17 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #17 to Host #4 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #18 to Host #4 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #19 to Host #4 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #1 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #2 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #3 failed by MIPS
[VmScheduler.vmCreate] Allocation of VM #20 to Host #4 failed by MIPS
0.1: DataCenterBroker1: VM #1 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #2 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #3 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #4 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: VM #5 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #6 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #7 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #8 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: VM #9 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #10 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #11 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #12 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: VM #13 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #14 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #15 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #16 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: Creation of VM #17 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #18 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #19 failed in Datacenter #2
0.1: DataCenterBroker1: Creation of VM #20 failed in Datacenter #2
0.1: DataCenterBroker1: Sending cloudlet 0 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 1 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 2 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 3 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 4 to VM #5
0.1: DataCenterBroker1: Sending cloudlet 5 to VM #6
0.1: DataCenterBroker1: Sending cloudlet 6 to VM #7
0.1: DataCenterBroker1: Sending cloudlet 7 to VM #8
0.1: DataCenterBroker1: Sending cloudlet 8 to VM #9
0.1: DataCenterBroker1: Sending cloudlet 9 to VM #10
0.1: DataCenterBroker1: Sending cloudlet 10 to VM #11
0.1: DataCenterBroker1: Sending cloudlet 11 to VM #12
0.1: DataCenterBroker1: Sending cloudlet 12 to VM #13
0.1: DataCenterBroker1: Sending cloudlet 13 to VM #14
0.1: DataCenterBroker1: Sending cloudlet 14 to VM #15
0.1: DataCenterBroker1: Sending cloudlet 15 to VM #16
0.1: DataCenterBroker1: Sending cloudlet 16 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 17 to VM #2
```

```
0.1: DataCenterBroker1: Sending cloudlet 18 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 19 to VM #4
5000.147: DataCenterBroker1: Cloudlet 2 received
5000.2570000000005: DataCenterBroker1: Cloudlet 1 received
5000.2570000000005: DataCenterBroker1: Cloudlet 6 received
5000.367000000001: DataCenterBroker1: Cloudlet 12 received
5000.367000000001: DataCenterBroker1: Cloudlet 7 received
5000.543000000001: DataCenterBroker1: Cloudlet 0 received
5000.651000000001: DataCenterBroker1: Cloudlet 9 received
5000.777: DataCenterBroker1: Cloudlet 5 received
5000.9: DataCenterBroker1: Cloudlet 14 received
5001.01: DataCenterBroker1: Cloudlet 4 received
5001.01: DataCenterBroker1: Cloudlet 8 received
5001.01: DataCenterBroker1: Cloudlet 13 received
5001.01: DataCenterBroker1: Cloudlet 3 received
5001.01: DataCenterBroker1: Cloudlet 15 received
5001.120000000001: DataCenterBroker1: Cloudlet 10 received
5001.120000000001: DataCenterBroker1: Cloudlet 11 received
10000.471000000001: DataCenterBroker1: Cloudlet 18 received
10000.752000000002: DataCenterBroker1: Cloudlet 16 received
10000.972000000002: DataCenterBroker1: Cloudlet 17 received
10001.858000000002: DataCenterBroker1: Cloudlet 19 received
10001.858000000002: DataCenterBroker1: All Cloudlets executed. Finishing...
10001.858000000002: DataCenterBroker1: Destroying VM #1
10001.858000000002: DataCenterBroker1: Destroying VM #2
10001.858000000002: DataCenterBroker1: Destroying VM #3
10001.858000000002: DataCenterBroker1: Destroying VM #4
10001.858000000002: DataCenterBroker1: Destroying VM #5
10001.858000000002: DataCenterBroker1: Destroying VM #6
10001.858000000002: DataCenterBroker1: Destroying VM #7
10001.858000000002: DataCenterBroker1: Destroying VM #8
10001.858000000002: DataCenterBroker1: Destroying VM #9
10001.858000000002: DataCenterBroker1: Destroying VM #10
10001.858000000002: DataCenterBroker1: Destroying VM #11
10001.858000000002: DataCenterBroker1: Destroying VM #12
10001.858000000002: DataCenterBroker1: Destroying VM #13
10001.858000000002: DataCenterBroker1: Destroying VM #14
10001.858000000002: DataCenterBroker1: Destroying VM #15
10001.858000000002: DataCenterBroker1: Destroying VM #16
DataCenterBroker1 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
DataCenter1 is shutting down...
DataCenterBroker1 is shutting down...
Simulation completed.
Simulation completed.
Result of cloudlet No
************************
CloudletID STATUS VmID WaitTime StartTime FinishTime
2      Success      3      0.00   0.10         5000.15
1      Success      2      0.00   0.10         5000.26
6      Success      7      0.00   0.10         5000.26
12     Success      13     0.00   0.10         5000.37
7      Success      8      0.00   0.10         5000.37
```

```
0      Success      1      0.00   0.10         5000.54
9      Success      10     0.00   0.10         5000.65
5      Success      6      0.00   0.10         5000.78
14     Success      15     0.00   0.10         5000.90
4      Success      5      0.00   0.10         5001.01
8      Success      9      0.00   0.10         5001.01
13     Success      14     0.00   0.10         5001.01
3      Success      4      0.00   0.10         5001.01
15     Success      16     0.00   0.10         5001.01
10     Success      11     0.00   0.10         5001.12
11     Success      12     0.00   0.10         5001.12
18     Success      3      5000.05 5000.15     10000.47
16     Success      1      5000.44 5000.54     10000.75
17     Success      2      5000.16 5000.26     10000.97
19     Success      4      5000.91 5001.01     10001.86
The Total Excecution Waiting Time of this Algorithm is : 20001.56
***************** Unsuccessfull Load Balancing *****************
***************** Total excecution wait time is not zero yet *****************
There is a waiting time of :20001.56
Used 4 host(s) Machines
***************** Simulation will restart now ***********************
Press Enter to Restart the simulation with: 5 Host(s)
*********************************
NEW SIMULATION
Total Number of Virtual Machine: to be used: 20
Total Number of Host Machine to be used: 5
Initialising...
Starting CloudSim version 3.0
DataCenter1 is starting...
DataCenterBroker1 is starting...
Entities started.
0.0: DataCenterBroker1: Cloud Resource List received with 1 resource(s)
0.0: DataCenterBroker1: Trying to Create VM #1 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #2 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #3 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #4 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #5 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #6 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #7 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #8 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #9 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #10 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #11 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #12 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #13 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #14 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #15 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #16 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #17 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #18 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #19 in DataCenter1
0.0: DataCenterBroker1: Trying to Create VM #20 in DataCenter1
0.1: DataCenterBroker1: VM #1 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #2 has been created in Datacenter #2, Host #2
```

```
0.1: DataCenterBroker1: VM #3 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #4 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: VM #5 has been created in Datacenter #2, Host #5
0.1: DataCenterBroker1: VM #6 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #7 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #8 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #9 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: VM #10 has been created in Datacenter #2, Host #5
0.1: DataCenterBroker1: VM #11 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #12 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #13 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #14 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: VM #15 has been created in Datacenter #2, Host #5
0.1: DataCenterBroker1: VM #16 has been created in Datacenter #2, Host #1
0.1: DataCenterBroker1: VM #17 has been created in Datacenter #2, Host #2
0.1: DataCenterBroker1: VM #18 has been created in Datacenter #2, Host #3
0.1: DataCenterBroker1: VM #19 has been created in Datacenter #2, Host #4
0.1: DataCenterBroker1: VM #20 has been created in Datacenter #2, Host #5
0.1: DataCenterBroker1: Sending cloudlet 0 to VM #1
0.1: DataCenterBroker1: Sending cloudlet 1 to VM #2
0.1: DataCenterBroker1: Sending cloudlet 2 to VM #3
0.1: DataCenterBroker1: Sending cloudlet 3 to VM #4
0.1: DataCenterBroker1: Sending cloudlet 4 to VM #5
0.1: DataCenterBroker1: Sending cloudlet 5 to VM #6
0.1: DataCenterBroker1: Sending cloudlet 6 to VM #7
0.1: DataCenterBroker1: Sending cloudlet 7 to VM #8
0.1: DataCenterBroker1: Sending cloudlet 8 to VM #9
0.1: DataCenterBroker1: Sending cloudlet 9 to VM #10
0.1: DataCenterBroker1: Sending cloudlet 10 to VM #11
0.1: DataCenterBroker1: Sending cloudlet 11 to VM #12
0.1: DataCenterBroker1: Sending cloudlet 12 to VM #13
0.1: DataCenterBroker1: Sending cloudlet 13 to VM #14
0.1: DataCenterBroker1: Sending cloudlet 14 to VM #15
0.1: DataCenterBroker1: Sending cloudlet 15 to VM #16
0.1: DataCenterBroker1: Sending cloudlet 16 to VM #17
0.1: DataCenterBroker1: Sending cloudlet 17 to VM #18
0.1: DataCenterBroker1: Sending cloudlet 18 to VM #19
0.1: DataCenterBroker1: Sending cloudlet 19 to VM #20
5000.1050000000005: DataCenterBroker1: Cloudlet 8 received
5000.206: DataCenterBroker1: Cloudlet 13 received
5000.206: DataCenterBroker1: Cloudlet 14 received
5000.316000000001: DataCenterBroker1: Cloudlet 11 received
5000.316000000001: DataCenterBroker1: Cloudlet 4 received
5000.426000000001: DataCenterBroker1: Cloudlet 0 received
5000.426000000001: DataCenterBroker1: Cloudlet 10 received
5000.426000000001: DataCenterBroker1: Cloudlet 2 received
5000.426000000001: DataCenterBroker1: Cloudlet 7 received
5000.426000000001: DataCenterBroker1: Cloudlet 12 received
5000.536000000002: DataCenterBroker1: Cloudlet 9 received
5000.646000000001: DataCenterBroker1: Cloudlet 16 received
5000.646000000001: DataCenterBroker1: Cloudlet 17 received
5000.759: DataCenterBroker1: Cloudlet 19 received
5000.869000000001: DataCenterBroker1: Cloudlet 5 received
5000.869000000001: DataCenterBroker1: Cloudlet 6 received
```

```
5000.869000000001: DataCenterBroker1: Cloudlet 3 received
5000.979000000001: DataCenterBroker1: Cloudlet 15 received
5000.979000000001: DataCenterBroker1: Cloudlet 1 received
5001.089000000002: DataCenterBroker1: Cloudlet 18 received
5001.089000000002: DataCenterBroker1: All Cloudlets executed. Finishing...
5001.089000000002: DataCenterBroker1: Destroying VM #1
5001.089000000002: DataCenterBroker1: Destroying VM #2
5001.089000000002: DataCenterBroker1: Destroying VM #3
5001.089000000002: DataCenterBroker1: Destroying VM #4
5001.089000000002: DataCenterBroker1: Destroying VM #5
5001.089000000002: DataCenterBroker1: Destroying VM #6
5001.089000000002: DataCenterBroker1: Destroying VM #7
5001.089000000002: DataCenterBroker1: Destroying VM #8
5001.089000000002: DataCenterBroker1: Destroying VM #9
5001.089000000002: DataCenterBroker1: Destroying VM #10
5001.089000000002: DataCenterBroker1: Destroying VM #11
5001.089000000002: DataCenterBroker1: Destroying VM #12
5001.089000000002: DataCenterBroker1: Destroying VM #13
5001.089000000002: DataCenterBroker1: Destroying VM #14
5001.089000000002: DataCenterBroker1: Destroying VM #15
5001.089000000002: DataCenterBroker1: Destroying VM #16
5001.089000000002: DataCenterBroker1: Destroying VM #17
5001.089000000002: DataCenterBroker1: Destroying VM #18
5001.089000000002: DataCenterBroker1: Destroying VM #19
5001.089000000002: DataCenterBroker1: Destroying VM #20
DataCenterBroker1 is shutting down...
Simulation: No more future events
CloudInformationService: Notify all CloudSim entities for shutting down.
DataCenter1 is shutting down...
DataCenterBroker1 is shutting down...
Simulation completed.
Simulation completed.
Result of cloudlet No
**************************
CloudletID STATUS VmID WaitTime StartTime FinishTime
8       Success     9      0.00   0.10         5000.11
13      Success     14     0.00   0.10         5000.21
14      Success     15     0.00   0.10         5000.21
11      Success     12     0.00   0.10         5000.32
4       Success     5      0.00   0.10         5000.32
0       Success     1      0.00   0.10         5000.43
10      Success     11     0.00   0.10         5000.43
2       Success     3      0.00   0.10         5000.43
7       Success     8      0.00   0.10         5000.43
12      Success     13     0.00   0.10         5000.43
9       Success     10     0.00   0.10         5000.54
16      Success     17     0.00   0.10         5000.65
17      Success     18     0.00   0.10         5000.65
19      Success     20     0.00   0.10         5000.76
5       Success     6      0.00   0.10         5000.87
6       Success     7      0.00   0.10         5000.87
3       Success     4      0.00   0.10         5000.87
15      Success     16     0.00   0.10         5000.98
1       Success     2      0.00   0.10         5000.98
```

18 Success 19 0.00 0.10 5001.09 **The Total Execution Waiting Time of this Algorithm is: 0.00**
********** Successfully Achieved Load Balancing (0 waiting time) **********
Total # of host machine used: 5
Total # of VM used: 20

Table 6: Final Simulation result after applying the C-Algorithm

Paired t-test can also be applied to compare the initial result of the simulation with 1 host machine to the final result with 5 physical host machines.

	Waiting time with 1 host	Waiting Time with 5 Hosts	D	D^2
	0	**0**	0	0
	0	**0**	0	0
	0	**0**	0	0
	0	**0**	0	0
	5000.28	**0**	5000.28	25002800.08
	5000.61	**0**	5000.61	25006100.37
	5000.5	**0**	5000.5	25005000.25
	5000.81	**0**	5000.81	25008100.66
	10000.48	**0**	10000.48	100009600.2
	10000.95	**0**	10000.95	100019000.9
	10001.18	**0**	10001.18	100023601.4
	10001.49	**0**	10001.49	100029802.2
	15001.04	**0**	15001.04	225031201.1
	15001.22	**0**	15001.22	225036601.5
	15000.85	**0**	15000.85	225025500.7
	15001.84	**0**	15001.84	225055203.4
	20001.71	**0**	20001.71	400068402.9
	20001.6	**0**	20001.6	400064002.6
	20001.37	**0**	20001.37	400054801.9
	20001.96	**0**	20001.96	400078403.8
Mean	10000.8945	0		
Total	200017.89	0		

t-Test: Paired Two Sample for Means

	Waiting time with 1 host	*Waiting Time with 5 Hosts*
Mean	10000.8945	0
Variance	52640016.21	0
Observations	20	20
Pearson Correlation	#DIV/0!	
Hypothesized Mean Difference	0	
df	19	
t Stat	**6.164471323**	
P(T<=t) one-tail	3.17195E-06	
t Critical one-tail	1.729132812	
P(T<=t) two-tail	6.3439E-06	
t Critical two-tail	2.093024054	

Table 7: T-Test Comparison of final results after applying the C-Rule algorithm

The value of the paired t-test is **6.16 which indicates a big different with 95% percent level of confidence.**

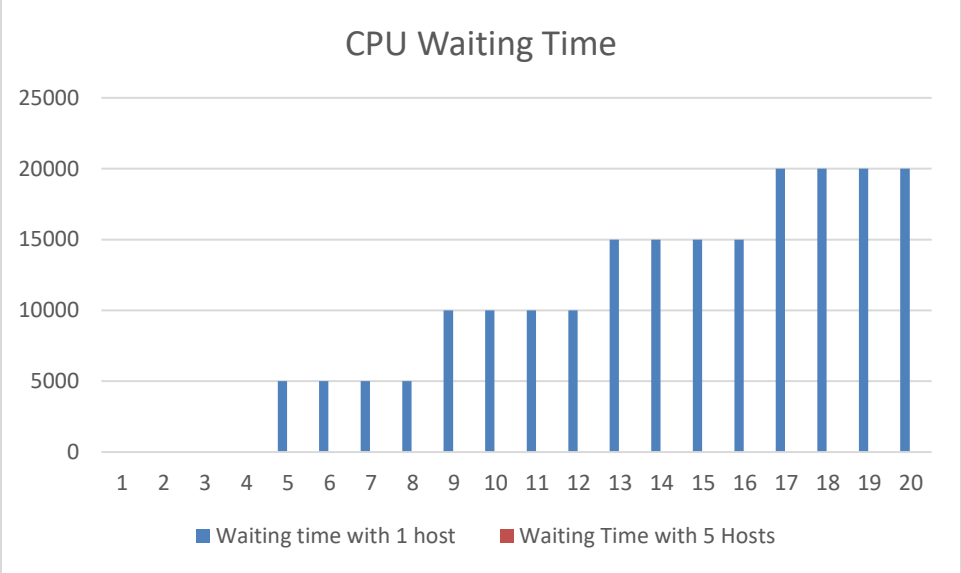

Figure 20: Chart of Final CPU Waiting Time after adding 5 host machines.

If no previous simulation data were to be found then C-Algorithm would run its own simulation to workload balance if there was no previous predictions, random

algorithm is used temporary for load balancing while CloudSim runs a simulation to find the optimal amount of resource allocation for the workload.

6.3 CONCLUSION

Chapter 2-6 answered all research questions and hypothesis. All conditions for unreliable and reliable prediction and simulation were discussed in chapter 3.

The literature from Chapter 3 indicated that Cicada can generate a prediction in less than 20 milliseconds and the result from Chapter 5 demonstrated that CloudSim can detect the minimum number of host machines and memory RAM required for a workload in a matter of seconds. The C-Rule algorithm managed to lower the number of physical host machines and memory ram by %50 in some cases, resulting in a **much faster workload balancing**.

This new approach helped cloud services achieve faster and more reliable workload balancing, allowing them to utilize their resources more efficiently by preventing any over-provisioning. Cloud service providers can use the workload prediction and if any similar workload exists in the historical data then C-Rule algorithm can simply use the result from the previous prediction. If no previous data exists in the database then C-Rule algorithm can use the prediction data and simulate a workload balancing and use that simulation data in future for a faster workload balancing. The C-Rule algorithm can balance a workload in a matter of seconds rather than several minutes.

6.4 Summary

The objectives of this work are three-fold: to investigate under what conditions to use Cicada for predicting workloads in dynamic Internet hosting platforms, to determine the conditions to use CloudSIM for reliable workload simulation, and to identify the challenges of rule-based algorithm for load balancing for Internet-based platforms compared to Cicada predictions. The methodology envisaged in this work entails three phases: workload prediction using Cicada, simulation using CloudSIM framework, and the development of a space-shared algorithm (C-algorithm) for dynamic workload balancing in cloud environments. The final objective is to reduce cloud resource assignment for a specific workload while reducing the CPU waiting time.

Chapter 6 demonstrated a successful simulation of the data center. Results from chapter 6 proved that prediction workload balancing can achieved faster results and can require less computational power. Results from this chapter covered the 3rd research question.

REFERENCES

Al-Qudah, Z., Alzoubi, H. A., Allman, M., Rabinovich, M., & Liberatore, V. (2009). Efficient application placement in a dynamic hosting platform. In ’09 *Proceedings of the 18th ACM International Conference on World Wide Web, Madrid, Spain,* pp. 281-290.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, *41*(1), 23-50.

Calheiros, R. N., Ranjan, R., De Rose, C. A. F., & Buyya, R. (2009). CloudSim: A novel framework for modeling and simulation of cloud computing infrastructure and services. Technical Report GRIDS-TR-2009-1, Grid Computing and Distributed Systems Laboratory.

Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., & Doyle, R. P. (2001, October). Managing energy and server resources in hosting centers. In *ACM SIGOPS Operating Systems Review*, *35*(5), 103-116.

Devi, C., & Uthariaraj, R. (2016). Load balancing in cloud computing environment using improved weighted Round Robin Algorithm for non-preemptive dependent tasks. *Hindawi Publishing Corporation,* http://dx.doi.org/10.1155/2016/3896065.

Doyle, B., & Lopes, C. V. (2005). Survey of technologies for Web application development. *ACM Journal, 2*(3), 1-43.

Duggan, J., Cetintemel, U., Papaemmanouil, O. & Upfal, E. (2011). Performance prediction for concurrent database workloads. *SIGMOD '11. June 12-16, 2011, Athens, Greece*. 978 (1): 337-348.

Issawi, S. F., Halees, A. A., & Radi, M. (2015). An efficient adaptive load-balancing algorithm for cloud computing under bursty workloads. *Engineering, Technology, & Applied Science Research, 5*(3), 795-800.

Jena, S. R., & Ahmad, Z. (2013). Response time minimization of different load balancing algorithms in cloud computing environment. *International Journal of Computer Applications, 69*(17), 22-27.

LaCurts, K. L. (2014, June). *Application workload prediction and placement in cloud computing systems* (Unpublished doctoral dissertation). Massachusetts Institute of Technology, Cambridge Massachusetts.

Lee, R., & Jeng, B. (2011). Load-balancing tactics in cloud. In *Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge CyberC Discovery,* pp. 447-454.

Mahmood, Z. (2011). Cloud computing: characteristics and deployment approaches. In *the 11th IEEE International Conference on Computer and Information Technology,* pp. 121-126.

Mathur, S., Larji, A. A., & Goyal, A. (2017). Static load balancing using SA Max-Min algorithm. *International Journal for Research in Applied Science & Engineering Technology, 5*(4), 1886-1893.

Nae, V., Prodan, R., & Fahringer, T. (2010, October). Cost-efficient hosting and load balancing of massively multiplayer online games. In the *11th IEEE/ACM International Conference on Grid Computing* (GRID), Brussels, Belgium, pp. 9-16.

Nema, R., & Edwin, S. T. (2016). A new efficient virtual machine load balancing algorithm for a cloud computing environment. *International Journal of Latest Research in Engineering and Technology, 2*(2), 69-75.

Olston, C., Manjhi, A., & Garrod, C. (2005). A scalability service for dynamic web applications. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR).* Retrieved from http://www.cs.cmu.edu/~manjhi/publications/ss-cidr05.pdf

Oluwatolani, O., Babajide, A., & Philip, A. (2012). Development of a scalable architecture for dynamic web-based applications. *International Journal of Information and Communication Technology Research, 2*(3), 304-311.

Pasha, N., Agaarwal A., & Rastogi, R. (2014). Round Robin approach for VM load balancing algorithm in cloud computing environment. *International Journal of Advanced Research in Computer Science and Software Engineering, 4*(5), 34-39.

Patel, D., & Rajawat, A. (2015). Efficient throttled load-balancing algorithm in cloud environment. *International Journal of Modern Trends in Engineering and Research,* 2(3), 463-480.

Rajeshkannan, R., & Aramudhan, M. (2016). Comparative study of load balancing algorithms in cloud-computing environment. *India Journal of Science and Technology, 9*(20), 1-7.

Sheng, Q. Z., Qiao, X., Vasilakos, A. V., Szabo, C., Bourne, S., & Xu, X. (2014). Web services composition: A decade's overview. *Information Sciences, 280,* 218-238.

Singh, S., & Jangwal, T. (2012). Cost breakdown of public cloud computing and private cloud computing and security issues. *International Journal of Computer Science and Information Technology, 4*(2), 17-31.

Sleit, A., Misk, N., Badwan, F., & Khalil, T. (2013). Cloud computing challenges with emphasis on Amazon EC2 and Windows Azure. *International Journal of Computer Networks & Communications, 5*(5), 35-43.

Wolke, A., Bichler, M., & Setzer, T. (2015). Planning vs. dynamic control: Resource allocation in corporate clouds. Retrieved from http://dss.in.tum.de/files/bichler-research/2015.IEEE_TCC.Wolke.Proactive.pdf

Buyya Rajkumar, Ranjan Rajiv, & Calheiros, N. Rodrigo. "Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities." In *IEEE International Conference on High*

Performance Computing & Simulation(HPCS), Leipzig, German, 2009 June 21-24. Doi: 10.1109/HPCSIM.2009.5192685.

Katrina, L., Mogul, J.C., Balakrishnan, H., & Turner, Y. (2014). Cicada: Predictive Guarantees for Cloud Network Bandwidth. cambridge: Massachuse t t s i n stitute of technology.

Jun-Kwon, J., Jung, S., Kim, T., & Chung, T. (2012). A Study on the Cloud Simulation with a Network Topology Generator (11 ed., Vol. 6). International Journal of Computer and Information Engineering.

Blaszczyszyn, B., Javonavic, M., & Karray, M. K. (2014). How user throughout depends on the traffic demand in large cellular networks. In *12th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks*, Hammamet, Tunisia. Doi: 10.1109/WIOPT.2014.6850355.

Gamal, M., Rizk, R., Mahdi, H., & Elhady, B. (2017). Bio-inspired load balancing algorithm in cloud computing. In *Proceedings of the International Conference on Advanced Intelligent Systems and Informatics*, Cairo, Egypt, 2017, pp. 579-589.

Hashem, W., Nashaat, H., & Rizk. (2017). Honey bee based load balancing in cloud computing. *KSII Transactions on Internet and Information Systems, 11*(12), 5694-5711.

Hwang, K. (2017). *Cloud computing for machine learning and cognitive applications: A machine Learning approach.* London: MIT Press.

Kaur, R., & Luthra, P. (2014). Load balancing in cloud computing. In *Proceedings of the International Conference on Recent Trends in information, Telecommunication, Computing,* Association of Computer Electronics and Electrical Engineers, pp. 375-381.

Kumar, S., & Mishra, A. (2015). Application of min-min and max-min algorithm for task scedhuling in cloud environment under time-shared and space-shared VM models. *International Journal of Computing Academic Research,* 182-190.

Nema, L., & Sharma, A. (2016). Efficient load balancing based on improved honey bee method in cloud computing. *International Science Press, 9*(22), 151-161.

Shiny, S. (2013). Load balancing in cloud computing: A review. *Journal of Computer Engineering, 15*(2), 22-29.

Wahab, M. N., Mexiani, S. N., & Atyabi, A. (2015). A comprehensive review of swarm optimization algorithms. *PLoS One, 10*(5), pp. 1-36.

Wang, T., Liu, Z., Chen, Y., Xu, Y. & Dai, X. (2014). Load balancing task scheduling based on genetic algorithm in cloud computing. In *IEEE 12th International Conference on Dependable, Autonomic, and Secure Computing,* Dalian China. Doi: 10.1109/DASC.2014.35.

Zhou, X., Lin, F., Yang, L., Nie, J., Tan, Q., Zeng, W., & Zhang, N. (2016). Load balancing method of cloud storage based on analytical hierarchy process and hybrid hierarchical genetic algorithm. *SpringerPlus, 5*, pp. 1-23. Doi:10.1186/s40064-016-3619-x.

Fei, L., Scherson, I.D., & Fuentes, J. (2017). Dynamic Creation of Virtual Machines in Cloud Computing Systems. Las Vegas: IEEE.

10.1109/ICSEng.2017.13

Jiang, Z. (2013). GreenCloud for Simulating QoS-based NaaS in Cloud Computing. Leshan City: 2013 Ninth International Conference on Computational Intelligence and Security. 10.1109/CIS.2013.167

Wenhong Tian, Minxian Xu, Aiguo Chen, Guozhong Li, Xinyang Wang, Yu Chen. Open-source simulators for Cloud computing: Comparative study and challenging issues, Simulation Modelling Practice and Theory, 2015, 58: 239-254

K. S. Umadevi and P. Chaturvedi (2017) Predictive load balancing algorithm for cloud computing: 2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS), Vellore, 2017, pp. 1-5. doi: 10.1109/ICMDCS.2017.8211727

Matthias, S., Klink, M., Tomforde, S., & Hahner, J. (2016). Predictive Load Balancing in Cloud Computing Environments based on Ensemble Forecasting (4 ed., Vol. 11). Augsburg,: IEEE International Conference on Autonomic Computing.

Alexandre, D., Tomasik, J., Cohen, J., & Dufoulon, F. (2017). Load prediction for energy-aware scheduling for Cloud computing platforms. Orsay: IEEE 37th International Conference on Distributed Computing System.

APPENDIX A - Java Classes of CloudSim

Introduction to programing language of CloudSim

The programing language of CloudSim is Java language and to understand the simulation process, we need to introduce the class packages of CloudSim. This section introduces all available packages of cloud of CloudSim and explains the behavior of each java class related to the topic of this paper.

The following table contains the classes defined in the CloudSim:

Classes	Description
Org.cloudbus.cloudsim Datecenter.Java DatacenterBroker.jave Cloudlet.Java File.Java Host.JavaStorage.Java	This class contains two different categories of classes, which generate different simulation behaviors and processes. The first category is the simulation components and the following classes can fit in the simulation category.
VmAllocationPolicy.java VmAllocationPolicySimple.java VmScheduler.java VmSchedulerSpaceShared.java VmSchedulerTimeShared.java VmSchedulerTimeSharedOverSubscription.java	Scheduling and utilization policy. The following components can fit in these categories:
Org.cloudbus.cloudsim.core	This class package set contains core classes for CloudSim, which can handle the core functionalities of the CloudSim toolkit.
Org.cloudbus.cloudsim.core.predicates:	This class package set is responsible for matching and selecting events from deferred queue list for execution.
Predicate.java	This class selects events from a deferred queue.
PredicateAny.java	This class will match a prediction to events in the deferred event queue.
Org.cloudbus.cloudsim.distributions:	This class package contains a set of classes that are responsible for the implementation of different distribution techniques, used commonly in network events. These classes can produce specific distribution technique including Lomax Distributions, Exponential, Random Number Generator, and Gamma Distribution as follows:
ContinuousDistribution.java	Continuous Distribution
ExponentialDistr.java	Exponential Distribution

GammaDiskr.java	Gama Distribution
LongnormalDistr.java	Long normal Distribution
LomaxDistribution.java	Lomax Distribution
ParetoDistr.java	Pare to Distribution
UniformDistr.java	Uniform Distribution
Org.cloudbus.cloudsim.lists	This class package contains sets of classes that generate objects, each containing lists of different components generated by the program. That is, the class contains operations lists on the lists of resources. Each generated object is stored in the memory. The package consists of codes designed for modeling cloud entities such as hosts, VMs, and datacenters. It also encompasses various scheduling and provisioning policies (Mishra & Sahoo 3). Users can extent or overwrite the classes to define additional cloud entities or create new policies.
CloudletList.java	class - stores Cloudlet list
HostList.java	class - stores Host List
ResCloudletList.java	class - stores list resources of cloudlets
VmList.java class	stores list of virtual machines
Org.cloudbus.cloudsim.network	This package set contains set of classes, which produce different network routing behavior. The set encompasses classes for the network topology including the delay matrix as well as the routing algorithm and topological information.
DelayMatrix_Float.java	class can produce a delay-topology storing routing behavior
FloydWarshall_Float.java	class uses Floyd War shall algorithm, which can calculate all pair delay
GraphReaderBrite.java	class is a file reader.
TopologicalGraph.java	class draws a graph, which contains nodes and edges.
TopologicalLink.java	class represents link edges from a graph.
TopologicalNode.java	class represents network nodes in a topological generated network and it can read information from a file.
Org.cloudbus.cloudsim.network.datacenter	This class package set contains sets of classes, which are extension of org.cloudbus.cloudim package set. This package set is used to simulate behavior of geographically distributed service providers.
AggregateSwitch.java	Class – simulates switch of a Datacenter network.
AppCloudlet.java	class simulates an application, which users submit for execution within a datacenter environment.
EdgeSwitch.java	class – simulates edge switch of a datacenter network and exchanges packets by interacting with other switches.
HostPackage.java	class – stores the information about cloudlets that are communication with each other and its main job is to represent packages, which travel within the virtual network with a host.
NetDatacenterBroker.java	class – functions as a broker that is acting from behalf of Datacenter provider and it makes VM management hidden.
NetDatacenterBroker.java	class – functions as a broker that is acting from behalf of Datacenter provider and it makes VM management hidden.

NetworkCloudlet.java	class simulates complex applications where each cloudlet will represent an application task and each task has several stages.
NetworkCloudletSpaceSharedScheduler.java	java class is used for spaced shared scheduling.
NetworkHost.java	class – this is an extension class for simulating datacenter networks.
NetworkPacket.java	class – this class simulates traveling packets among servers.
NetworkVm.java	class – this class extends VM and simulates datacenters of networks.
NetworkVmAllocationPolicy.java	class – it chooses the hosts for least PEs from virtual machines.
RootSwitch.java	class – simulates external root switch of a Datacenter.
TaskStage.java	class – represents different stages of a cloudlet during the execution time.
Org.cloudbus.cloudsim.power	class– this class package set simulates functionality of power aware components. It encompasses extendable classes that can simulate a power aware DC and policies.
Org.cloudbus.cloudsim.provisioners	class – class package set can simulate bandwidth-provisioning policy of virtual machines.
Org.cloudbus.cloudsim.utilclass	class package set can simulate and measure execution time of a cloud environment.
ExecutionTimeMeasurer.java	class can measure the execution time.
WorkloadFileReader.java	class can create a list of jobs by importing traces of resources from a file.
WorkloadModel.java	class can define a workload model by generating and dispatching list of jobs to a resource.

Table 8: Important Java Classes of CloudSim Simulator

APPENDIX B - Tools: Installation of CloudSim and Commons Math files.

Step-by-step Installation Instructions of Cloud Sim on a PC computer:

- Step 1: The first step for installing CloudSim on a computer is to remove all other versions of Java from your computer before you can install JDK version of Java from your computer.
    - Restart your computer
    - Make sure that all previous files and folders of previous Java installation is removed from your computer
        - Remove C:\Program Files\Java
        - Remove C:\Program Files (x86)\Java

Step 2: Use Ninite tool for easy installation of Java JDK x64 8 and Eclipse compiler

- Go to https://ninite.com/
- Under the developer tools select both Java JDK x64 8 and Eclipse compiler
- Under the runtimes select Java 8
- Click on get your Ninite.
- Download and install Ninite
- This installation will automatically install two different versions of Java on your computer.
    - Java SE Development Kit 8 Update 152
    - Java 8 Update 152

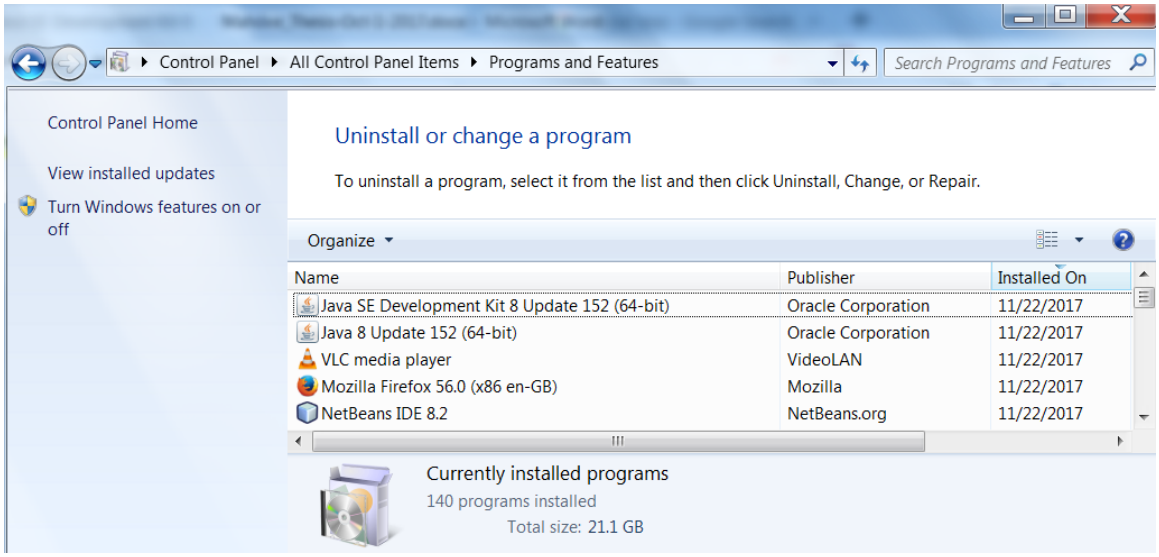

Figure 21. Screenshot of Java 64-bit

- Step 3: Ninite will also automatically install eclipse and it will add proper Java runtime environment variables on windows.
- Step 4: Download CloudSim

Download CloudSim 3.0.3.zip from

https://code.google.com/archive/p/cloudsim/downloads

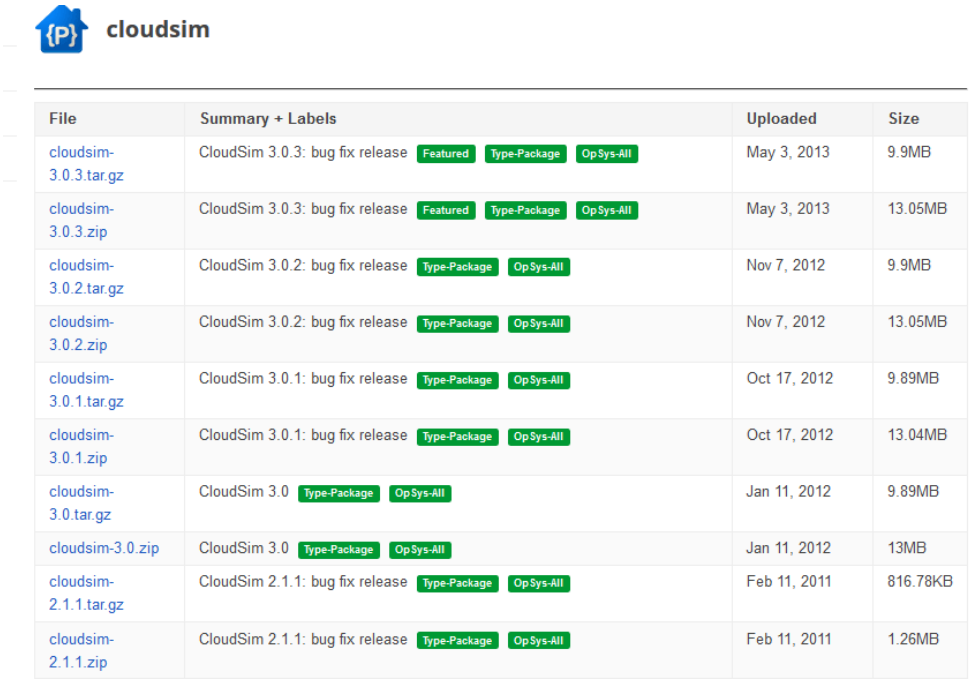

Figure 22. Google Code offers open-source project hosting

Another option is to download CloudSim from https://github.com/Cloudslab/cloudsim/releases

GitHub is an open-source software development platform that also allows hosting and reviewing of codes, as well as project management. When downloaded from GitHub, the downloaded package comes with the source code, examples, API html files, and jars.

- Step 5:

Download Apache Commons Math 3.3 from http://commons.apache.org/proper/commons-math/download_math.cgi.

Common Math provides a lightweight library for addressing the common math problems that do not exist in Java. The library requires Java 1.5+.

- Step 6:
- Unzip all zip files to C:\CodeRespository\
- Step 7:
    - Click on File → new Java project
        - In front of Project name choose your project name
    - Select use an execution environment JRE: JavaSe-1.8
    - Click on next
    - Click on Libraries
    - Click on Add external Jars and select C:\CodeRespository\cloudsim-3.0.3\jars\cloudsim-3.0.3.rar
- Click on finish

- Open your project on the left side and right Click on SRC folder
- Click on New and select Class option
    - In front of the name type your desired class name and select
    - both "public static void main(String[] args)
    - inherited abstract methods
    - Generate commands
- Click on finish and this will create an empty class to which can run Cloudsim Java code.

APPENDIX C – Java code for the simulation

The following Java code is for the simulation of the CloudSim.

```
import java.io.FileNotFoundException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.CloudletSchedulerSpaceShared;
import org.cloudbus.cloudsim.CloudletSchedulerTimeShared;
import org.cloudbus.cloudsim.Datacenter; // for private data center
import org.cloudbus.cloudsim.DatacenterBroker;
import org.cloudbus.cloudsim.DatacenterCharacteristics;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicySimple;
import org.cloudbus.cloudsim.VmSchedulerSpaceShared;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.provisioners.BwProvisionerSimple;
import org.cloudbus.cloudsim.provisioners.PeProvisionerSimple;
import org.cloudbus.cloudsim.util.WorkloadFileReader;
import org.cloudbus.cloudsim.UtilizationModelFull;


public class ThesisPartOne {

                    /*
                     *  All Variables are moved here for ease of changing
                     */
static int NumOfHostAddedSoFar=0;
static int NumOfHostToUseNextTime=1;
static boolean resimulate=true;
static double TotalExcTime=0;
static int numUser=1;  // Number of Users

//          **************Host Specs ****************************
static int   HostRAM = 32000;          //32 GB of RAM        for each host
static int   HostBandwidth=8000;      // 8 for test 2 Mbs of Bandwidth for each host
```

```
static long  HostStorage = 2000000;   // 2000 GB of Storage   for each host

static int  NumberOfVM=20;       //20  give best results
static long vmdiskSize = 20000;  //20000 Virtual Machine Storage   size
static int  VMRam=1000;         //2000 Virtual Machine Ram       Size
static int  VMmips =1000;       //Virtual Machine mips      Size
static int  VMbandwidth = 1000;  //Virtual Machine bandwidth Size
static int  VCPU = 1;           //Virtual Machine number of CPU
static String VMM ="XEN";
static String architecture="64 bits";   //This defines the architecture of the VM
static String os ="Ubuntu Server";      //OS of the VM
static String vmm="VMware";             //Software of the virtual machine
static double timeZone =7.0;
static double EachComputercostPerSec = 2.0;
static double costPerMem=0.75;
static double costPerStorage=0.10;
static double costPerBw=0.11;

static long  cloudletLength= 5000000; //Length of Instruction from the (task and workloads)
static int   TaskCPUNum=1;              // Number of the CPU of the task and workload /
static long  cloudletInputFileSize=100000;  //input  file size from the (task and workloads) section
static long  cloudletOutputSize =300000;  //output file from the (task and workloads) section
public static void main(String[] args) throws FileNotFoundException {
Log.printLine("\n**************  Length of Instruction from the (Task and Workloads) Section
**************");
Log.printLine("Cloudlet Length   "+cloudletLength);
Log.printLine("# of Task CPU:    "+TaskCPUNum);
Log.printLine("Input file size:  "+cloudletInputFileSize);
Log.printLine("Output file size: "+cloudletOutputSize);
Log.printLine("\n**************  Each Host  ******************************");
Log.printLine("Memory RAM       :   "+HostRAM/1000+" GB");
Log.printLine("Bandwidth        :   "+HostBandwidth/1000+" Mbs");
Log.printLine("Storage (SSD/HDD):   "+HostStorage/1000+" GB");
Log.printLine("\n**************  Each VM (Virtual Machine) **************");
Log.printLine("Disk Disk    : "+vmdiskSize/1000+" GB"); //Virtual Machine Storage size
Log.printLine("Memory RAM   : "+VMRam/1000+"  Gb");
Log.printLine("VM Mips      : "+VMmips+"  ");
Log.printLine("VM Bandwidth : "+VMbandwidth/1000+"  Mbs  ");
Log.printLine("# of VM CPU  : "+VCPU+"");
Log.printLine("**********************\n");
Log.printLine("System Architecture: "+architecture);
Log.printLine("OS Type:       "+os);
Log.printLine("VM Software:   "+vmm);
Log.printLine("************** C-Algorithm Load Balancer
*****************************\n");
Log.printLine("This program will find the optimal amount of VMs and Hosts \n");
Log.printLine("Press Enter to continue");
                        try{System.in.read();} catch(Exception e){}
                        try{System.in.read();} catch(Exception e){}
```

```
            while (resimulate==true) {

Log.printLine("************************************");
Log.printLine("NEW SIMULATION");
Log.printLine("Total Number of Virtual Machine: to be used: "+ NumberOfVM);
Log.printLine("Total Number of Host Machine    to be used:
"+NumOfHostToUseNextTime+"\n");

//******************Part I: Initializing the CloudSim package**********************
//In The following part we the entities
Calendar cal = Calendar.getInstance();
boolean traceFlag=false;

//This will print Initialising... On the screen
CloudSim.init(numUser, cal, traceFlag);

          //******************PART II: Datacenter is created in this part
          //******************Characteristics of a DataCenters is created in the
CreateDataCenter();
          // HostList is created in this part and HostList elements will be processed
          // VM allocation policy and scheduling will be defined here
Datacenter dc=CreateDataCenter();
//***************Part 3 : Create Broker***************
DatacenterBroker dcb=null;
      try {dcb = new DatacenterBroker("DataCenterBroker1");} catch (Exception e)
{e.printStackTrace();}

   //*********************************Part 4

List<Cloudlet> cloudletList = new ArrayList<Cloudlet>();
UtilizationModelFull  fullUtilize =new UtilizationModelFull();
    for(int cloudletId =0;cloudletId <20;cloudletId ++) {

Random r= new Random();
Cloudlet anewcloudlet = new Cloudlet(cloudletId , cloudletLength+r.nextInt(1000),
TaskCPUNum, cloudletInputFileSize, cloudletOutputSize, fullUtilize,fullUtilize,fullUtilize);
anewcloudlet.setUserId(dcb.getId());



                cloudletList.add(anewcloudlet);  // instead of generating random cloudlets we are
importing hp simulation data
                                        }//end of for loop


  // In this part we will import hp data
   //      WorkloadFileReader workloadFileReader = new
WorkloadFileReader("C:\\CodeRespository\\ThesisPartOne\\HPC2N-2002-2.2-cln.swf", 1);
//    cloudletList=workloadFileReader.generateWorkload();
```

```
//**********  Part 5 of the simulation VMs are created and
//********** Task scheduling algorithm will be defined here **********

List<Vm>  vmList =new ArrayList<Vm>();

            /*
             *   All variables have been moved to the beginning of the
                Program Beginning
             */



    for(int vmId =1;vmId <NumberOfVM+1; vmId ++)
                {Vm VirtualMachine= new Vm(vmId, dcb.getId(),
                                        VMmips,
                                        VCPU,
                                        VMRam,
                                        VMbandwidth,
                                        vmdiskSize,
                                        VMM,
                                        new CloudletSchedulerSpaceShared());
                                        vmList.add(VirtualMachine);}
                                        dcb.submitCloudletList(cloudletList);  // cloudlets are submitted
to the broker in a list
                                        dcb.submitVmList(vmList);


//  Part 6.0: Simulation starts in part 6 even simulation (engine)

    CloudSim.startSimulation();
    List<Cloudlet> Finalresults = dcb.getCloudletReceivedList();
    CloudSim.stopSimulation();

//*********************************************************************************
   // Part 7.0  Print results when the simulation is over(output)
    int cloudletNo=0;

    DecimalFormat TwoDecimalFormatter = new DecimalFormat("#0.00"); //defining decimal
format
    String status;
    String space;
    String Startspace;
    String FinishTimespace;
    String ExcTimespace;
    Log.printLine("Result of cloudlet No");
    Log.printLine("***************************");
    Log.printLine("CloudletID  STATUS  VmID              WaitTime      StartTime
FinishTime");
    for (Cloudlet c: Finalresults)
        {         //c.getResourceId()   //
```

```
                //c.
            status="Fail";
            space="      ";
            Startspace="        ";
            FinishTimespace="         ";
            ExcTimespace="          ";
            if (c.getCloudletId()>9) { space="     ";     } //More spaces if getCloudletId is a two digit
number
            if (c.getExecStartTime()<0.9 ) {Startspace="         ";}
            if (c.getExecStartTime()>0.9 ) {Startspace="     ";}
            if (c.getFinishTime()>99) {FinishTimespace="        ";}

            if (c.getFinishTime()>9999) {FinishTimespace="          ";}
            if (c.getWaitingTime()==0) {ExcTimespace="              ";}
            if (c.getWaitingTime()>999) {ExcTimespace="               ";}
            if (c.getCloudletStatus() == c.SUCCESS){
            status="Success";  //c.getActualCPUTime()   c.getResourceId()

//      c.getWaitingTime()
              Log.printLine("    "+c.getCloudletId()  + space +status+"    "+c.getVmId()+
""+ExcTimespace+TwoDecimalFormatter.format(c.getWaitingTime())+
                          "
"+TwoDecimalFormatter.format(c.getExecStartTime())+Startspace+"
"+TwoDecimalFormatter.format(c.getFinishTime())+FinishTimespace );
             }  //c.getCloudletStatus() == c.SUCCESS)

              cloudletNo++;
              TotalExcTime=TotalExcTime+c.getWaitingTime();

                    }
              Log.printLine("\nThe Total Excecution Waiting Time of this Algorithm is :
"+TwoDecimalFormatter.format(TotalExcTime));

              if (TotalExcTime==0) {

                    Log.printLine("\n********** Successfully Achieved Load Balancing (0
waiting time) **********");
                    Log.printLine("\nTotal # of host machine used: "+ NumOfHostToUseNextTime);
                    Log.printLine("\nTotal # of VM used: "+NumberOfVM);
                    resimulate=false;
             }
              if (TotalExcTime!=0) {
                          Log.printLine("\n********** Unsuccessfull Load Balancing
**********");
                              Log.printLine("********** Total excecution wait time is not
zero yet **********");
                              Log.printLine("There is a waiting time of
:"+TwoDecimalFormatter.format(TotalExcTime));
                              Log.printLine("Used "+NumOfHostToUseNextTime+" host(s)
Machines");
```

```
                                NumOfHostToUseNextTime++;

                               if(NumOfHostToUseNextTime==13) {
                                  Log.printLine("Maximum of 13 host machines are allowed");
                                  resimulate=false;
                                   }

                              Log.printLine("\n****************** Simulation will restart now
**********************");
                               Log.printLine("Press Enter to Restart the simulation with:
"+NumOfHostToUseNextTime+" Host(s)");
                             try{System.in.read();} catch(Exception e){}
                              try{System.in.read();} catch(Exception e){}

                               TotalExcTime=0;
            }




          }
    }//END OF THE PUBLIC STAT

   private static Datacenter CreateDataCenter()
         { //Beginning of CreateDataCenter(), Datacenter characteristics is defined here
                  List<Pe> peList = new ArrayList<Pe>();
                  //The following  lines will Define the MIPS of each CPU
                  //Each core has 1000 MIPS
                  PeProvisionerSimple ProcessorProvisioner = new PeProvisionerSimple(1000);
//1000 MIPS
                  //Each core will have its own ID
                  //In the following part we will assign an ID for each core
                  Pe CPUcore1 = new Pe(0, ProcessorProvisioner);
                  Pe CPUcore2 = new Pe(1, ProcessorProvisioner);
                  Pe CPUcore3 = new Pe(2, ProcessorProvisioner);
                  Pe CPUcore4 = new Pe(3, ProcessorProvisioner);
                  //We add each core to the Pe list
                  peList.add(CPUcore1);
                  peList.add(CPUcore2);
                  peList.add(CPUcore3);
                  peList.add(CPUcore4);

                  /*
                   *  Variables are moved to the beginning of the program for ease of changing
                   *
                   */
List<Host> hostlist = new ArrayList<Host>();

//Each Host is created in this stage
```

```
                Host host1 = new Host(1, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host2 = new Host(2, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host3 = new Host(3, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host4 = new Host(4, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host5 = new Host(5, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host6 = new Host(6, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host7 = new Host(7, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host8 = new Host(8, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host9 = new Host(9, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host10 = new Host(10, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host11 = new Host(11, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host12 = new Host(12, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host13 = new Host(13, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host14 = new Host(14, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host15 = new Host(15, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host16 = new Host(16, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
                Host host17 = new Host(17, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
```

```
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
            Host host18 = new Host(18, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
            Host host19 = new Host(19, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));
            Host host20 = new Host(20, new RamProvisionerSimple(HostRAM), new
BwProvisionerSimple(HostBandwidth),
                                HostStorage, peList, new VmSchedulerSpaceShared(peList));

            if (NumOfHostToUseNextTime==1) {hostlist.add(host1);
NumOfHostAddedSoFar++;}
            if (NumOfHostToUseNextTime==2) {
                                    hostlist.add(host1); NumOfHostAddedSoFar++;
                                hostlist.add(host2); NumOfHostAddedSoFar++;
                                }

            if (NumOfHostToUseNextTime==3) {
                                    hostlist.add(host1); NumOfHostAddedSoFar++;
                        hostlist.add(host2); NumOfHostAddedSoFar++;
                        hostlist.add(host3); NumOfHostAddedSoFar++;
                                }
            if (NumOfHostToUseNextTime==4)
                                {

                                    hostlist.add(host1); NumOfHostAddedSoFar++;
                                    hostlist.add(host2); NumOfHostAddedSoFar++;
                                    hostlist.add(host3); NumOfHostAddedSoFar++;
                                    hostlist.add(host4); NumOfHostAddedSoFar++;}

            if (NumOfHostToUseNextTime==5)
                                {hostlist.add(host1); NumOfHostAddedSoFar++;
                            hostlist.add(host2); NumOfHostAddedSoFar++;
                                hostlist.add(host3); NumOfHostAddedSoFar++;
                                hostlist.add(host4); NumOfHostAddedSoFar++;
                                hostlist.add(host5); NumOfHostAddedSoFar++;}

            if (NumOfHostToUseNextTime==6)  {hostlist.add(host1);
NumOfHostAddedSoFar++;

hostlist.add(host2); NumOfHostAddedSoFar++;
                        hostlist.add(host3); NumOfHostAddedSoFar++;
                        hostlist.add(host4); NumOfHostAddedSoFar++;
                        hostlist.add(host5); NumOfHostAddedSoFar++;
                        hostlist.add(host6); NumOfHostAddedSoFar++;}
            if (NumOfHostToUseNextTime==7)  {
                                    hostlist.add(host1); NumOfHostAddedSoFar++;
                                hostlist.add(host2); NumOfHostAddedSoFar++;
                            hostlist.add(host3); NumOfHostAddedSoFar++;
```

```
                    hostlist.add(host4); NumOfHostAddedSoFar++;
                    hostlist.add(host5); NumOfHostAddedSoFar++;
                    hostlist.add(host6); NumOfHostAddedSoFar++;
                    hostlist.add(host7); NumOfHostAddedSoFar++;}

            if (NumOfHostToUseNextTime==8)  {
hostlist.add(host1); NumOfHostAddedSoFar++;
hostlist.add(host2); NumOfHostAddedSoFar++;
hostlist.add(host3); NumOfHostAddedSoFar++;
hostlist.add(host4); NumOfHostAddedSoFar++;
hostlist.add(host5); NumOfHostAddedSoFar++;
hostlist.add(host6); NumOfHostAddedSoFar++;
hostlist.add(host7); NumOfHostAddedSoFar++;
hostlist.add(host8); NumOfHostAddedSoFar++;}

            if (NumOfHostToUseNextTime==9)  {
hostlist.add(host1); NumOfHostAddedSoFar++;
hostlist.add(host2); NumOfHostAddedSoFar++;
hostlist.add(host3); NumOfHostAddedSoFar++;
hostlist.add(host4); NumOfHostAddedSoFar++;
hostlist.add(host5); NumOfHostAddedSoFar++;
hostlist.add(host6); NumOfHostAddedSoFar++;
hostlist.add(host7); NumOfHostAddedSoFar++;
hostlist.add(host8); NumOfHostAddedSoFar++;
hostlist.add(host9); NumOfHostAddedSoFar++;}
            if (NumOfHostToUseNextTime==10)  {
hostlist.add(host1); NumOfHostAddedSoFar++;
hostlist.add(host2); NumOfHostAddedSoFar++;
hostlist.add(host3); NumOfHostAddedSoFar++;
hostlist.add(host4); NumOfHostAddedSoFar++;
hostlist.add(host5); NumOfHostAddedSoFar++;
hostlist.add(host6); NumOfHostAddedSoFar++;
hostlist.add(host7); NumOfHostAddedSoFar++;
hostlist.add(host8); NumOfHostAddedSoFar++;
hostlist.add(host9); NumOfHostAddedSoFar++;
hostlist.add(host10); NumOfHostAddedSoFar++;}

            if (NumOfHostToUseNextTime==11)  {
hostlist.add(host1); NumOfHostAddedSoFar++;
hostlist.add(host2); NumOfHostAddedSoFar++;
hostlist.add(host3); NumOfHostAddedSoFar++;
hostlist.add(host4); NumOfHostAddedSoFar++;
hostlist.add(host5); NumOfHostAddedSoFar++;
hostlist.add(host6); NumOfHostAddedSoFar++;
hostlist.add(host7); NumOfHostAddedSoFar++;
hostlist.add(host8); NumOfHostAddedSoFar++;
hostlist.add(host9); NumOfHostAddedSoFar++;
hostlist.add(host10); NumOfHostAddedSoFar++;
hostlist.add(host11); NumOfHostAddedSoFar++;}
            if (NumOfHostToUseNextTime==12)  {
```

```
        hostlist.add(host1); NumOfHostAddedSoFar++;
        hostlist.add(host2); NumOfHostAddedSoFar++;
        hostlist.add(host3); NumOfHostAddedSoFar++;
        hostlist.add(host4); NumOfHostAddedSoFar++;
        hostlist.add(host5); NumOfHostAddedSoFar++;
        hostlist.add(host6); NumOfHostAddedSoFar++;
        hostlist.add(host7); NumOfHostAddedSoFar++;
        hostlist.add(host8); NumOfHostAddedSoFar++;
        hostlist.add(host9); NumOfHostAddedSoFar++;
        hostlist.add(host10); NumOfHostAddedSoFar++;
        hostlist.add(host11); NumOfHostAddedSoFar++;
        hostlist.add(host12); NumOfHostAddedSoFar++;}


                if (NumOfHostToUseNextTime==20)  {
        hostlist.add(host1); NumOfHostAddedSoFar++;
        hostlist.add(host2); NumOfHostAddedSoFar++;
        hostlist.add(host3); NumOfHostAddedSoFar++;
        hostlist.add(host4); NumOfHostAddedSoFar++;
        hostlist.add(host5); NumOfHostAddedSoFar++;
        hostlist.add(host6); NumOfHostAddedSoFar++;
        hostlist.add(host7); NumOfHostAddedSoFar++;
        hostlist.add(host8); NumOfHostAddedSoFar++;
        hostlist.add(host9); NumOfHostAddedSoFar++;
        hostlist.add(host10); NumOfHostAddedSoFar++;
        hostlist.add(host11); NumOfHostAddedSoFar++;
        hostlist.add(host12); NumOfHostAddedSoFar++;
        hostlist.add(host13); NumOfHostAddedSoFar++;
        hostlist.add(host14); NumOfHostAddedSoFar++;
        hostlist.add(host15); NumOfHostAddedSoFar++;
        hostlist.add(host16); NumOfHostAddedSoFar++;
        hostlist.add(host17); NumOfHostAddedSoFar++;
        hostlist.add(host18); NumOfHostAddedSoFar++;
        hostlist.add(host19); NumOfHostAddedSoFar++;
        hostlist.add(host20); NumOfHostAddedSoFar++;}


  // hostlist.add(host5); NumOfHostAddedSoFar++;
//    hostlist.add(host6); NumOfHostAddedSoFar++;
       //    hostlist.add(host7); NumOfHostAddedSoFar++;
       //    hostlist.add(host8); NumOfHostAddedSoFar++;
       //    hostlist.add(host9); NumOfHostAddedSoFar++;
       //    hostlist.add(host10); NumOfHostAddedSoFar++;
       //  hostlist.add(host11); NumOfHostAddedSoFar++;

                //In this simulation there are only 2 hosts later more hosts can be added

   //These variables show the cost of
   /*
    double timeZone =7.0;
```

```
        double EachComputercostPerSec = 2.0;
        double costPerMem=0.75;
        double costPerStorage=0.10;
        double costPerBw=0.11;
                */

DatacenterCharacteristics acharacteristic =
                                new DatacenterCharacteristics(architecture, os, vmm, hostlist,
timeZone,
                                                EachComputercostPerSec, costPerMem,
                                                costPerStorage, costPerBw);
                LinkedList<Storage> SANstroage = new LinkedList<Storage>();
                Datacenter aDatacenter=null;

        try {//exception starts
                        aDatacenter = new Datacenter("DataCenter1", acharacteristic,
                        new VmAllocationPolicySimple(hostlist), SANstroage, 1);
                        } catch (Exception e1) {e1.printStackTrace();}//end of exception
        return aDatacenter;} //end of CreateDataCenter()  }
```