Spring 2018

Load Balancing and Virtual Machine Allocation in Cloud-based Data Centers

Saily Satish Ghodke
San Jose State University

Load Balancing and Virtual Machine Allocation in Cloud-based Data Centers

By
Saily Ghodke

April 2018

The Designated Project Committee Approves the Project Titled

Load Balancing and VM Allocation in Cloud based Data Centers

By

Saily Satish Ghodke

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

April 2018

Dr. Robert Chun	Department of Computer Science
Dr. Sami Khuri	Department of Computer Science
Mr. Mandar Joshi	CloudSimple

Abstract:
As cloud services see an exponential increase in consumers, the demand for faster processing of data and a reliable delivery of services becomes a pressing concern. This puts a lot of pressure on the cloud-based data centers, where the consumers' data is stored, processed and serviced. The rising demand for high quality services and the constrained environment, make load balancing within the cloud data centers a vital concern. This project aims to achieve load balancing within the data centers by means of implementing a Virtual Machine allocation policy, based on consensus algorithm technique. The cloud-based data center system, consisting of Virtual Machines has been simulated on CloudSim – a Java based cloud simulator.

Index terms – **Cloud data centers, cloud simulators, consensus, load balancing, resource management, VM allocation**

Acknowledgements

I would like to sincerely offer my gratitude to Professor Robert Chun, for his invaluable guidance and support for this project. I would also like to thank my parents for providing me with an excellent education and encouraging me to explore and pursue my academic interests. I would like to thank my partner for his patience and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Conventionally, cloud can be said to be a shared pool of resources such as memory, storage, network, bandwidth, processing power, etc. [1]. Features such as scalability, security, reliability, utility computing and mobility has made cloud computing a very lucrative business model for IT companies [2]. Cloud services can be easily available in the form of infrastructure, platform or software, defining the three cloud service models - IaaS (Infrastructure as a Service), PaaS (Platform as a Service) and SaaS (Software as a Service). Public, private and hybrid are the three deployment models of the cloud. Services of the cloud are availed universally across a diverse class of consumers ranging from domestic to industries, hospitals, data centers, etc. All the requests of these consumers are processed in cloud data centers.

An increasing number of consumers and their requests puts a lot of pressure on the data centers for efficient and reliable delivery of services, while ensuring optimum resource utilization and minimum response time [2]. Cloud data centers are typically characterized by resource constraints and unequal, dynamic tasks. This gives rise to a need for resource management within the cloud data center. If the data center entities are under or over utilized, it will lead to an inefficient resource utilization and will ultimately result poor delivery of service. To avoid this, load balancing becomes a crucial resource management tenet [3]. This paper surveys and examines several load balancing techniques and consensus algorithms in the context of cloud data centers [4]. It proposes the use of three consensus based VM allocation algorithms to achieve load balancing.

The rest of this work is organized as follows: Section II provides the related works and explores the concepts of load balancing and consensus algorithms in the context of cloud data centers. Section III presents the proposed overview, detailing the idea and algorithms proposed in this project. Section IV describes the experimental setup and Java implementation of the proposed algorithms. Section V provides a detailed analysis and comparative results of the proposed algorithms. Section VI concludes and summarizes the paper.

II. RELATED RESEARCH AND BACKGROUND

A. Resource Management

1. Elements of Resource Management

Ullrich *et al* [3] consider resource management to be a broader idea comprising of several interconnected concepts. Some of the areas that the authors mention, are resource modeling, resource estimation, resource discovery and selection, and resource scheduling. They provide an overview of the relation between resource estimation, resource discovery and selection and resource monitoring. Resource estimation comprises of prediction of resource demand of application considering the given workload and configuration and the estimation of the amount of time that will be taken to complete the execution of the application. Resource discovery and selection involve identification of the resource based on heuristics or linear regression. Resource monitoring becomes essential in order ensure efficient resource management. Resource monitoring involves monitoring physical and virtual characteristics of resources, such as memory consumption, network bandwidth, response time, hardware utilization, etc. Resource estimation and resource monitoring can be achieved efficiently via resource modelling.

Sonkar *et al*. [2] examine the components of resource management in the context of cloud computing. The authors identify Resource monitoring and allocation as the two of the most important elements of Resource Management. According to them, resource management is the process of discovering the resource, allocating it and then monitoring it. Resource discovery is a process managed by the cloud service provider and determines a resource from the available resources that is most suitable depending on the application request. This process is responsible for notifying the server about the status of the resources or in other words, notifying the server about resource availability. Resource allocation is the actual assignment of cloud resources for the users' applications, over the internet. In the cloud computing model, resources are allocated to users according to the utility computing model. Scheduling and dispatching are the two methods used in resource allocation. While the scheduler is responsible for planning the assignment of resources to the consumers, the dispatcher is responsible for the actual allocation of resources. The allocation takes place according to a Resource Allocation Policy (RAP). This allocation policy should take into consideration the demand of consumers of cloud services, while ensuring small response time, reliable service delivery and efficient utilization of resources. It should aim at avoiding over and under-utilization of resources. In the context of the cloud, RAP typically takes the form of VM Resource scheduling and allocation [4]. To achieve maximum and efficient utilization of resources, VMs should be properly allocated. Therefore, designing an appropriate VM allocation policy for achieving load balancing becomes a vital part of efficient resource management.

2. Challenges of Resource Management

This section surveys various challenges that have been identified in the context of resource management. Sonkar and Kharat have examined a few of the key challenges that are typically encountered in a cloud-based data center: scarcity of resources, resource contention, resource fragmentation, over-provisioning and under-utilization [2]. Cloud data centers are typically

characterized by a scarcity of resources. Scarcity of resources occurs when there is an inadequate amount of resources to fulfill the demands. With the exponential increase in the number of cloud services, their consumers and the devices that they are deployed on, the number of processing resources falls short exponentially. This leads to resource contention. Resource contention within a data center occurs when multiple requests compete for the same processing resource. Resource contention takes place due to the limited nature of bandwidth, memory, processing power, etc. Entities within the cloud data centers, such as VMs, hosts, etc. must contend among themselves for these resources. On the other hand, resource fragmentation occurs when the number of resources is sufficient, but they are not available for use. Over-provisioning and under-utilization occur when the load is unequally balanced. Over-provisioning occurs when the supply of resources is lesser than their demand. Under-utilization occurs when there is a less demand and a higher availability of resources.

To address these challenges, there is a need for an efficient load balancing algorithm. Certain metrics or parameters are used to design and evaluate such algorithms.

3. Metrics for Load Balancing

Sonkar and Kharat [2] identify the following qualitative metrics to achieve load balancing in cloud data centers: throughput, associated overhead, migration time, response time, resource utilization, scalability and performance. Throughput is the measure of the total user requests processed by the data center - the higher the throughput, the better is the performance of the data center. Any algorithm has its own associated overhead. However, a successfully implemented algorithm should have least overhead. Migration time is the time needed to migrate a job from one VM to another. Least migration time is the measure of improved data center performance. Response time is the time taken by the system to respond to a users' request. Ideally, the response time should be as less as possible. Resource utilization depicts the amount of usage of the data center's resources. The load balancing algorithm should ensure that no resources are under or over utilized. An efficient load balancing algorithm should accommodate the scalability of the data center and maintain efficient resource utilization irrespective of a scale up or scale down. Performance of the data center strongly depends upon all the factors mentioned above. Thus, an efficient load balancing algorithm should ensure the best performance of the data center, signifying the effectiveness of the system.

B. Load Balancing

1. Need for Load Balancing

Load balancing is the division or distribution of load between the various components of a system. In the context of a cloud, load balancing can be said to be the distribution of users' requests in the form of tasks that require processing, between the VMs [6]. Within the cloud data centers, load balancing comes into picture to achieve better efficiency and throughput, while ensuring the availability of resources for processing users' requests. Over or under utilization of data center entities will lead to a wastage of resources and energy. Khara and Thakkar [7] list the objectives for load balancing in cloud computing as follows: i. On demand resource availability; ii. Efficient utilization of resources under diverse load conditions; iii. Reduced cost of resources; iv. Saving energy. However, there are many challenges that need to be addressed in a cloud computing

environment. These challenges include but are not restricted to: response time, scalability, independent and interdependent nature of tasks, unequal task sizes, virtualization, etc.

2. Load Balancing in Distributed Systems

Various load balancing techniques have been discussed in the context of both distributed systems and cloud networks. Load balancing algorithms can be static or dynamic [5].

2.1 Static

Static Load Balancing algorithms are based on the task completion time and are restricted to a more stable environment. A static environment typically has no dynamic changes or parameters in play at runtime.

Some of the major static load balancing algorithms that have been studied are Round Robin, Min-Min and Max-Min algorithms. The Round Robin algorithm is based on the concept of allotting fixed time slots or slices to each task in the task queue. The focus of this algorithm is to ensure fair allocation of every task in a time constrained environment. Each task in the system, gets the same time slice to utilize the resources and complete its execution. While this may seem to be a fair load balancing technique, it does not take into consideration the priority and size of the tasks. In the Min-Min algorithm, the tasks with the least utilization are scheduled first. Min-Min displays the ideal results in the event where small tasks are more in number. The major disadvantage of Min-Min algorithm is starvation. The Min-Min algorithm is not able to understand or handle any change in the machine or task. The Max-Min algorithms is based on the priority of tasks. It prioritizes the task of the largest size. Consequently, some tasks may be starved of resources or not be scheduled at all, while some other tasks may consume all the system's resources. While the advantage is that requirements are known at compile time, the disadvantage is that it takes a lot of time to complete tasks. Thus, most of these techniques cannot be used in cloud-based data centers, where each task must be scheduled, irrespective of its size and priority. Therefore, both these parameters must be considered, to achieve an efficient and load-balanced system.

2.2 Dynamic

Aslam and Shah [5] also studied and evaluated some dynamic algorithms that address some of the issues that are left unaddressed by the static algorithms. These algorithms can be used in the cloud infrastructure as they take into consideration several parameters such as network bandwidth and runtime work distribution according to the capacity of the resource. Some of the major dynamic load balancing algorithms that have been reviewed are: Ant colony optimization, Honey bee foraging algorithm and Throttled load balancing. In the Ant Colony Optimization algorithm, there is an important assumption: all tasks possess a unique computational weight and are assumed to be independent The resources are also assigned weights. Higher resource weight indicates higher computational power. This algorithm models the behavior of ants searching for food. However, the disadvantage lies in the fact that there is no specification when it comes down to the number of ants or nodes. Also, since the network is over loaded, the search takes a lot of time. Honey bee foraging algorithm revolves around the mapping of tasks to VMs depending on task priority. Tasks with higher priority map themselves to a VM which currently has tasks with lower priorities. If a task has high priority, it will choose a VM with the least number of high priority tasks. The advantages of this algorithm are that it reduces response time, while increasing the throughput, and it achieves load balancing while also taking task priority into account. However, the major

disadvantage of this algorithm is that tasks with high priorities cannot work in the absence of VMs. The Throttled load balancer is another dynamic Load Balancing strategy that has been evaluated in the paper. This algorithm takes into consideration the capacity of machines in terms of size. If the capability and size of the machine is found suitable, the task manager allocates the task to that machine. However, the disadvantage lies in the fact that the tasks need to wait.

All these algorithms provide standalone advantages. This gives rise to the need for a load balancing algorithm that takes into consideration factors such as fairness, network bandwidth, priority and throughput. To this effect, a hybrid approach appears to be the most suitable. The new approach should be compatible with the working dynamics of a cloud-based data center system.

3. Load Balancing in Cloud Computing

Several load balancing strategies have been widely researched in the context of cloud computing. Garg *et al* [6] summarize a few of the load balancing techniques used in CloudAnalyst such as Round Robin, Active Monitoring and Throttled Load Balancing. They propose an algorithm for load balancing in the VMs, building on the original Active Load Monitoring Load Balancing algorithm. Their algorithm maintains a HashMap of VM states and allocation counts. The Datacenter controller is responsible for checking this HashMap to check for available VMs to which tasks can be allocated based on the least loaded VM. Variants of Ant Colony Optimization (ACO), Honey Bee Foraging and Particle Swarm Optimization have also been researched and enhanced to map to the cloud architecture [6][7][17].

Within a cloud data center, the cloudlets that represent the users' requests are mapped to VMs for processing and execution. Each VM has parameters like computation cost and processing capability in terms of MIPS (Million Instructions Per Second) [5]. For most of the algorithms discussed in this section, certain assumptions have been made. For instance, tasks do not have priorities associated with them, execution time of tasks can be calculated initially and each VM executes tasks on a FCFS (First Come First Serve) basis. However, in a typical cloud computing architecture, each task has its own independent size, complexity, computational requirement and priority. Tasks could have different kinds of dependencies. Simultaneously, tasks must be scheduled in such a manner that no VM is over-utilized or under-utilized. This poses a challenge for task scheduling within the cloud infrastructure. These above factors are critical challenges for task scheduling in a cloud environment.

C. Consensus Algorithms

Conventional distributed systems, as well as modern day cloud computing, perform functions such as data storage, analysis and processing, job scheduling, processing customer requests, etc. All these tasks require coordination and consensus between the component resources [8]. Consensus and coordination are required for achieving efficient resource management, replication, atomicity and ultimately - service delivery. This has given rise to a need for standardized consensus and coordination algorithms that can be applied in the context of both distributed systems and cloud computing. Over the years, several algorithms have been proposed and widely researched. Paxos

and Raft have emerged as the most widely researched and industry-adapted algorithms. The following sections provide an insight into the implementation of both these algorithms.

1. Paxos

Paxos was introduced by the American computer scientist Leslie Lamport in 1989. According to Ailijiang *et al* [8], It was one of the first formally-proven consensus algorithms that dealt with the challenges like message loss, asynchrony, data loss, etc., in a typical distributed system architecture. It rose to fame when Google incorporated it into the Google File System for implementing a fault-tolerant coordination system, which gave birth to the GFS lock system - Google Chubby. Apache ZooKeeper provided an open source implementation of the Google Chubby lock service, called 'Zab,' to address ZooKeeper's coordination challenges.

According to its original creator, Paxos was originally designed for implementing a fault tolerant distributed system that addresses coordination and consensus challenges like asynchrony, message-loss, process crash / recovery, etc. [9]. The Paxos system is modelled on a typical asynchronous, non-Byzantine distributed system. It consists of a collection of processes that can propose different types of values. The consensus algorithm eventually selects only one of these proposed values. The Paxos system operates under the following constraints: i. Only a proposed value can be selected; ii. Only a single value is selected; iii. The processes remain unaware of the value, until it actually has been chosen. The processes in the Paxos system can either be proposers, acceptors or learners, and every process can act as more than one agent. The communication between these agents or processes takes place via messages called prepare request and accept request. The proposer sends proposals to a set of acceptors, and each acceptor accepts the first value that it receives. An acceptor can accept multiple proposals. The acceptor can only accept a proposal having a higher value than the value of the previously accepted proposal. The value that is accepted the most by the acceptors is chosen as the winner. The learner then asks the acceptors for the chosen value.

2. Raft

Several variants of the original Paxos consensus algorithm have been developed and used by the industry since the former's invention. Raft is one such algorithm developed by Stanford university students, to design an algorithm that was easier to understand and implement than Paxos [10]. Raft is different than Paxos in terms of features such as log replication, management, reduced number of states, stronger coherency, and guaranteeing safety for changing cluster memberships. Ongaro and Ousterhout claim that Raft is easier than Paxos, in terms of understanding. They also propose that Paxos, in addition to being difficult to understand, also requires some major complex changes to enable its use in modern systems.

The Raft system consists of several servers. Time is divided into terms of arbitrary integer value. Initially, all the servers begin as followers. For each term, one of the servers can be a leader and the rest of the servers are followers. The leader handles all the clients' requests, while the followers play a passive role. The servers communicate with each other via Remote Procedure Calls (RPCs). A heartbeat mechanism is used by Raft, to conduct the elections. The leaders initiate an AppendEntries RPC to provide the heartbeat. During the elections, RequestVote RPCs are initiated by the candidates. Each server can vote for only one candidate on a First-Come-First-Serve (FCFS)

basis. The candidate that receives the maximum votes from majority of servers, wins the election, for the term. The leaders maintain their authority by sending periodic heartbeat messages to the followers. If a follower does not receive any communication from the leader for a certain time, election timeout, it initiates a new election. Apart from the leader election, Raft also provides features such as log replication and safety for machine states.

3. Need for Consensus Algorithms in the Cloud

Consensus algorithms have been used widely, since decades, to implement efficient and robust distributed systems. These algorithms aim to achieve an agreement or consensus between entities or resources within a system [8]. A typical distributed network system consists of several autonomous server nodes, where each server node is associated with its own computational power and processing capability. Such a network system is called a data center and this data center is hosted on a cloud. Consumers avail different kinds of services from cloud service providers and these services get processed on the cloud-based data centers in the form of tasks. These tasks are typically diverse in terms of dependencies, priority, processing requirements and size. The traffic consisting of such tasks is dynamic and is handled by the cloud. To ensure efficient utilization of resources and reliable service delivery, load balancing is very essential. Inefficient load balancing can cause the workload to be unequally distributed to the servers which may lead to over-provisioning as well as starvation. This can lead to serious consequences such as high power-consumption, data center failure, slow and best-effort service delivery, etc. To achieve optimum load balancing, these component server nodes need to communicate efficiently between each other. Thus, consensus algorithms come into play.

In their paper [12], A. Loon and M. Mahyuddin have proposed a time-averaging, dynamic load balancing consensus algorithm that takes into consideration network scalability and provides a flexible mapping to diverse network topologies. This algorithm has been modelled on a network consisting of server nodes. This network supports fast consensus convergence rate and scalability. However, this algorithm has been proposed in the context of a distributed network. The challenge is to design a load balancing consensus algorithm in the context of the cloud.

4. Consensus Algorithms in Industry Applications

Marandi *et al* [11] have evaluated four open-source versions of Paxos - S-Paxos, OpenReplica, Ring Paxos, and Libpaxos, that are currently being used or researched in the context of the cloud. These algorithms were evaluated for different task sizes and different configurations of nodes within a cloud data center, in the form of some popular online services. They discovered that standard application of Paxos variants into cloud data centers introduce some unexpected behavior and increase the delay. The most important challenge for implementing Paxos based algorithms was discovered to be the diverse nature of clients' requests and the environment in which it is deployed. S-Paxos, is a Paxos implementation written in Java, specifically for load balancing. However, this algorithm fails to perform efficiently when there are a large number of nodes due to speed limitations. OpenReplica is another Paxos based open-source library, written in Python, that provides an object-oriented interface to the system application. Ring Paxos is a Java based library based on unicast network communication. It achieves leader election via ZooKeeper. The Ring is expected to operate corresponding to the speed of the slowest acceptor. However, the Ring gets reconfigured if an acceptor leaves. This greatly affects performance. Another Paxos library evaluated in the paper is LibPaxos. It is written in C and does not implement leader election.

Overwhelming and crashing of slow acceptors were discovered to be the demerits of this algorithm. Currently, in effect, most cloud-based application services make use of Apache's ZooKeeper as a provider of consensus and coordination services. According to Ailijiang *et al* [8], about 31% of Apache's current BigData and Cloud projects use ZooKeeper. Apache Accumulo, BookKeeper, Apache Solr, etc. are some of the prominent products that make use of ZooKeeper. Google's cluster management system Borg, makes use of Chubby, an implementation of Paxos, for master election and master state replication. Some of the important Google services that use Paxos and its variants are Google File System, BigTable and Google Spanner.

D. Cloud Simulators

1. Need for Cloud Simulators

Due to features such as scalability, utility computing, improved mobility, etc., cloud computing has emerged to be a lucrative business model as well as an immensely helpful service for consumers. As the scope and diversity in the use of cloud computing and its services increases, it becomes crucial to modify the existing cloud model, to incorporate new technology and address new demands. Before deploying such models, there is a need to experiment, develop and test the models [13]. However, experimentation and testing of such kind becomes very restrictive if attempted to be performed on real life platforms. In addition to this, using real cloud services as test beds will tend to incur heavy costs in terms of currency.

Ullrich *et al* [3] identify the use of simulation tools as one of the best solutions for resource modelling to perform efficient resource estimation and allocation, to achieve efficient resource management. Simulation requires a far lesser number of hardware resources and provides an excellent test bed for performance estimation.

Thus, cloud simulators successfully address this challenge by acting as tools to experiment, test and evaluate different cloud computing features and capabilities.

2. Comparison between Cloud Simulators

Maarouf *et al*., have identified various cloud simulation platforms and evaluated them to assess diverse parameters [14]. CloudSim, CloudAnalyst and NetworkCloudSim are some of the popular cloud simulators that have been studied by these authors. Several assessment parameters such as flexibility, VM allocation policy, network configuration, GUI (Graphical User Interface), etc. have been used to evaluate these simulation platforms.

CloudSim is one of the original and popular cloud simulators that was proposed in 2009. It is written in Java. It enables users to configure, model and test the simulation according to their requirements. For instance, users can design their own cloud architecture consisting of data centers, hosts and VMs. In addition to this, they can design and test their own allocation and scheduling policies. CloudSim is based on Discrete Event Simulation (DES). Features such as high scalability and low simulation overhead, make it a popular choice among cloud researchers. CloudSim has been the base platform for the development of several different flavors of cloud simulators.

CloudAnalyst is one such cloud simulator. It has been derived from CloudSim and SimJava and is written in Java. It aims at providing an added functionality of modelling internet applications. User-friendly GUI and flexibility make it an attractive option for users. Three load balancing algorithms are used in the CloudAnalyst: Active load balancing, Throttled load balancing and VM load balancing.

Another simulator based on CloudSim is the NetworkCloudSim. It provides features such as High Performance Computing (HPC) and scalability. This simulation is most suitable for more realistic applications and HPC. It supports two levels of scheduling: Host-level and VM-level.

EMUSIM, CDOSim, GreenCloud, etc. are some of the other loud simulators that were studied and evaluated in their paper. However, these simulators mainly focus on energy consumption and provide lesser flexibility in terms of configuration and policy design and testing.

3. CloudSim – A Java based Cloud Simulator

CloudSim is one of the most popular cloud simulators, programmed in Java [15]. The basic purpose of designing CloudSim was for designing, testing and evaluating resource performance and provisioning policies within the cloud. The simulated architecture components of CloudSim are data centers, hosts, processing elements and VMs. CloudSim has a hierarchical data center. It consists of a data center. This data center consists of a network of distributed hosts. Each host has its own processing elements as well as VMs. CloudSim allows users to configure this architecture and modify the number and mapping of the architectural components. Cloudlets are the tasks that represent users' requests. These cloudlets are mapped to the VMs, to get processed. The VMs are allocated on the hosts. This allocation takes place according to a VM allocation policy. The tasks are scheduled according to a scheduling policy. CloudSim allows users to design, modify and test their own allocation and scheduling policies. Features such as high scalability, flexibility and behavioral modelling make CloudSim a popular choice for users. CloudSim supports both single clouds as well as federation of clouds.

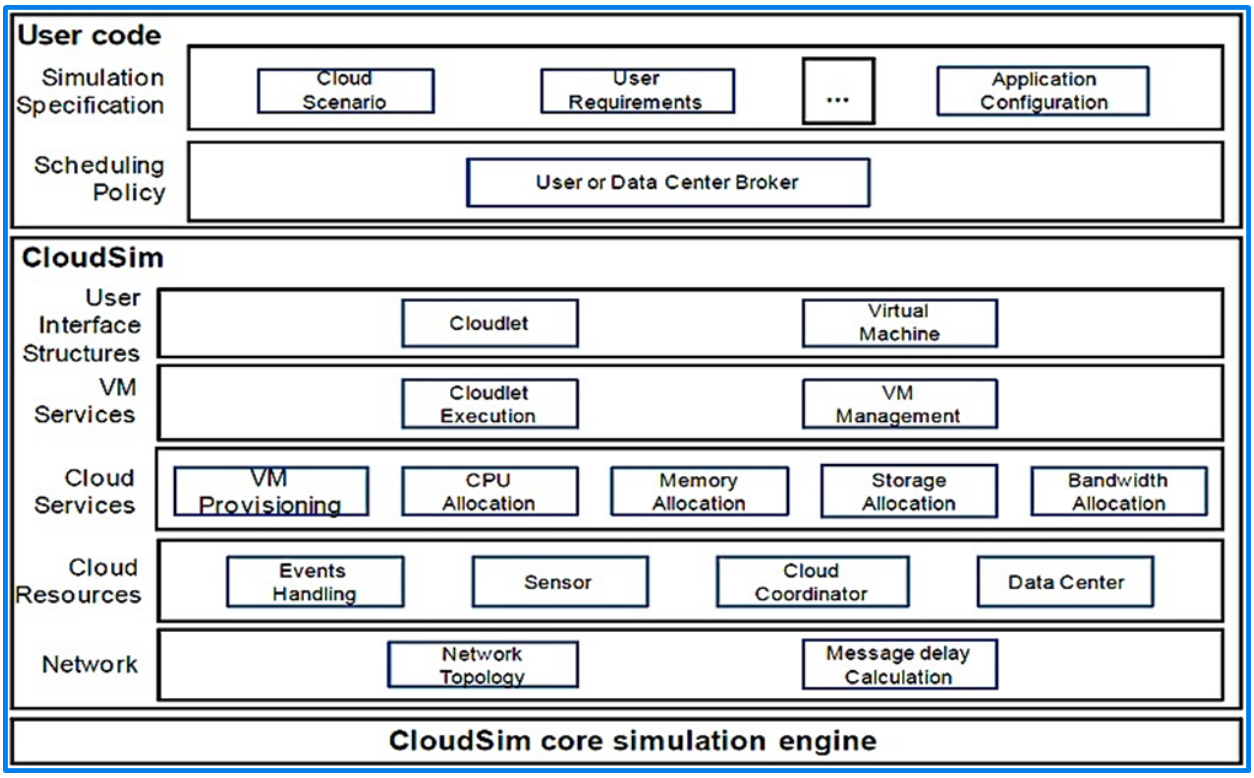

Figure 1. CloudSim Architecture [21]

Figure 1 depicts the three-layered CloudSim architecture. The lowest layer is the core simulation engine. It is responsible for the actual simulation and cannot be modified by the users. The middle layer is comprised of user interface structures for the cloudlets and VMs, VM services like cloudlet execution and VM management, cloud services, cloud resources and network. The top most layer is the user code, which is where the users can modify and test their algorithms. This layer allows for configuration according to the users' requirements and provides the platform for developing and testing various scheduling and allocation algorithms

III. OVERVIEW OF THE PROPOSED APPROACH

A. Overview

The cloud architecture comprises of many entities, technologies and service models that vastly differ from conventional distributed systems. While considering resource management and load balancing in the cloud, one needs to consider several characteristics of the cloud [16]. For instance, tasks that are sent to the cloud data centers are typically unequal in terms of size and nature, as well as in the requirement of network bandwidth and processing power, and priority. Virtualization technology has made it possible to sustain the cloud architecture and enabled high availability and scalability [17]. As the cloud enables more and more businesses to scale up and expand their consumer base, the consumer demands and requirements also increase. This puts a lot of load on the cloud data centers. Efficient resource utilization becomes vital to ensure scalability, reliable delivery of services and good performance of the cloud data centers. This is where resource allocation and load balancing algorithms come into play.

Three consensus algorithms loosely based on concepts borrowed from the conventional consensus algorithms Paxos and Raft, have been proposed in this paper. These algorithms aim to achieve load balancing within a cloud data center consisting of multiple hosts that host the VMs on which the users' tasks are processed. The proposed algorithms enable the data center hosts to achieve consensus among themselves, without a central controlling agent, eliminating a single point of failure. The consensus algorithms dictate the VM allocation scheme within the data center and provide an efficient distribution of load to all the hosts within the data center. These VM allocation schemes are modelled on first fit, best fit and worst fit scheduling strategies [18]. These algorithms have been developed and tested on CloudSim, a Java based Cloud Simulator. The paper provides a detailed analysis and comparison of the proposed algorithms, based on simulation runtime and total VM allocation as evaluation metrics.

The following sections provide a detailed explanation of the three algorithms and their evaluation.

B. Architectural Setup

The architectural setup for experimentation consists of a cloud data center. The load balancing algorithms are implemented within the data center. The data center consists of multiple hosts. These hosts in turn consist of multiple VMs. The users' tasks or cloudlets are mapped to the VMs. These VMs in turn get allocated to the hosts for their execution.

C. Algorithms

Three consensus algorithms have been proposed in this paper. These algorithms are based on three allocation strategies.

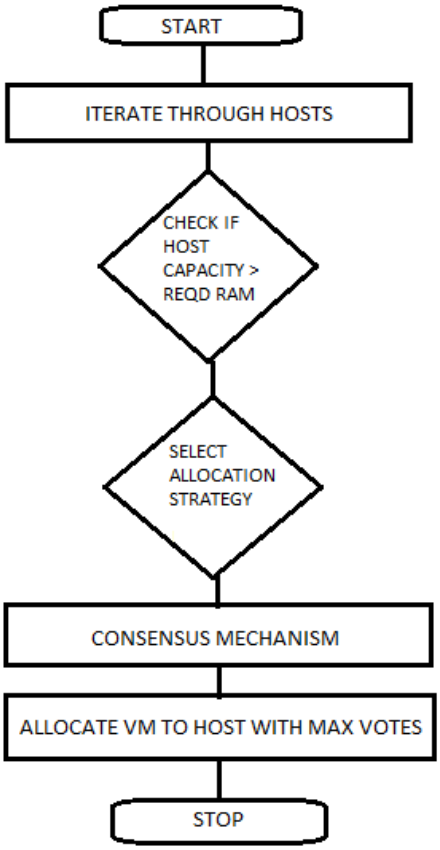

Figure 2. Flowchart for Load Balancing Algorithms

1. First Fit Consensus

The First Fit allocation strategy allocates the VMs in a First Come First Serve manner. Every VM is allocated to the first host that has the capacity to host that VM. For every capable host, all the hosts in the data center vote for this host, and the host with the maximum votes is chosen for allocating the VM.

2. Best Fit Consensus

The Best Fit strategy aims at making the allocated VMs utilize the maximum amount of memory on the host machine. A best fit host is found among all the available hosts based on the available memory on the host, post VM allocation. For this purpose, the algorithm checks the difference between the available host capacity and the memory requirement of the VM and selects the host with the minimum difference. In other words, the host takes up the largest VM that it can host. For every ideal (best fit) host, the host votes for itself, and all other hosts vote for that host. In the end of the consensus, the host with maximum votes will be chosen for VM allocation.

3. Worst Fit Consensus

The Worst Fit strategy aims at allocating maximum number of VMs on a given host. A worst fit host is found among all the available hosts based on the available memory on the host, post VM

allocation. For this purpose, the algorithm checks the difference between the available host capacity and the memory requirement of the VMs. The host with the maximum difference is selected. In other words, the host takes up the smallest VM that it can host. For every ideal (best fit) host, the host votes for itself, and all other hosts vote for that host. In the end of the consensus, the host with maximum votes will be chosen for VM allocation.

4. Consensus Mechanism

The input to the algorithm is the VM list. For every VM, the algorithm iterates through the hosts. It checks if the capacity of the host is more than the RAM required by the VM. If the capacity of the host is more than the RAM requirement of the VM, the algorithm initiates a voting mechanism between the hosts, unique to the allocation strategy chosen. Each host can vote for itself as well as for other hosts. The votes are stored in a data structure called the "VoteMap." At the end of the voting, the host having the maximum votes for the VM is chosen for the allocation for that VM. If any VM is left unallocated, it gets queued up and is put up for allocation in the next round of consensus. The simulation ends after all the VMs have been allocated.

D. Evaluation Metrics

1. Simulation Runtime

Faster service delivery is one of the major requirements of cloud service consumers. In addition to this, in a cloud data center, some tasks may have time criticality associated with them. Therefore, time consumption becomes a critical factor in evaluating cloud service delivery. In a simulation environment, the equivalent of time consumption is simulation runtime. Therefore, simulation runtime has been used as one of the evaluation metrics. The simulation runtime is the total time taken from the beginning of the simulation to the end, that includes the time taken to accept the users' tasks, create VMs, map the tasks to the VMs, conduct consensus between the hosts, allocate the VMs, reallocate any pending VM and destroy the VMs. The simulation runtime is measured in milliseconds (ms).

2. Total VM Allocation

Another evaluation metric that has been used is the VM allocation. This is the metric that evaluates the load balancing that has been achieved in the data center. The VM allocation measures the total RAM that has been allocated on each host, given an allocation policy. The difference between the least RAM value allocated and the maximum RAM value allocated provides an insight into the degree of load balancing. VM allocation per host is measured in Megabytes (MB).

IV. EXPERIMENTATION AND EVALUATION

A. Experimental Setup

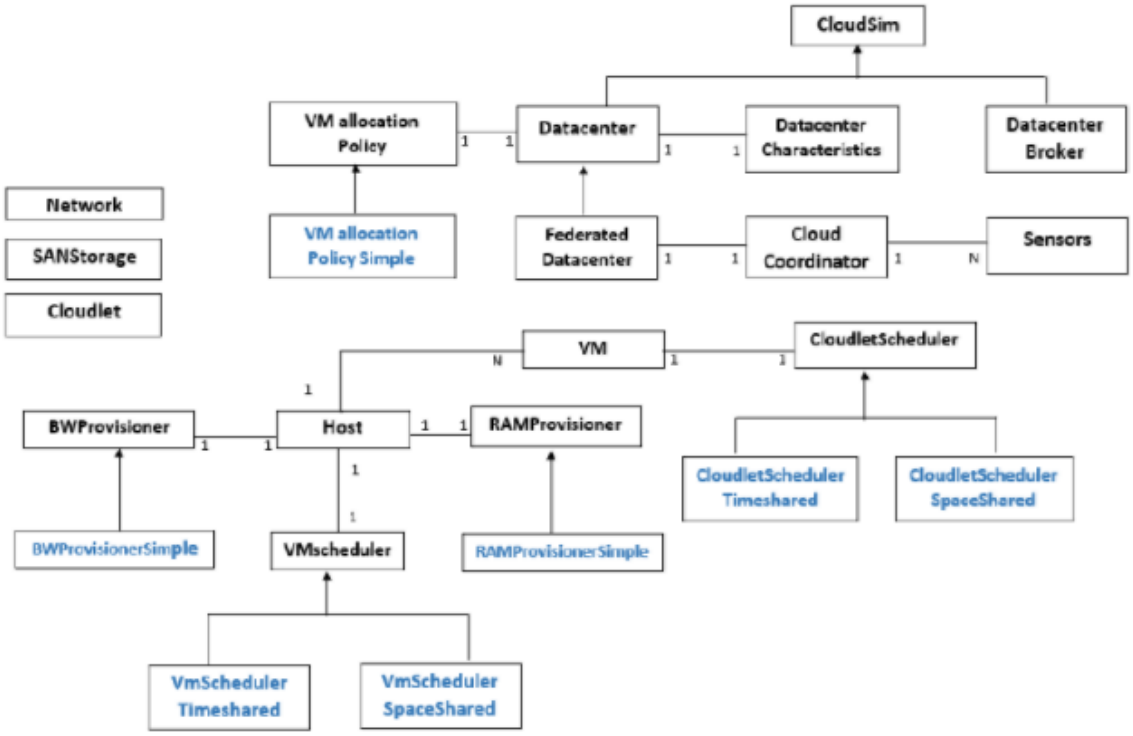

Figure 3. CloudSim Setup of Classes [19]

Figure 3 depicts the setup of classes in CloudSim. These classes represent the default setup of the simulator. The users have the ability to modify the existing classes as well as to add new classes. The different classes in CloudSim perform different functionalities such as resource provisioning, VM allocation, creation of cloudlets, mapping cloudlets to VMs and scheduling them [20]. Following are some of the important classes in CloudSim:

Cloudlet:
It is a public class that is responsible for the creation of cloudlets or tasks, along with defining the characteristics of these cloudlets. It stores and displays the ID of the VM to which it is mapped.

DataCenter:
It is a Cloud Resource in which hosts are virtualized. The characteristics of the data center can be configured in this class. The VMs are scheduled according to the specific allocation policy in this class.

DataCenterBroker:
The data center broker is responsible for assigning or mapping the cloudlets to the VMs.

Host:
This class is associated with the DataCenter. It is capable of hosting VMs. It executes all the actions related to the scheduling of VMs according to the VM allocation policy.

VMAllocationPolicy:
This is an abstract class that implements the hosts' provisioning policy. This class is responsible for the allocation and deallocation of VMs within the hosts.

The "Host" class used by the data center class of CloudSim has been modified to include the Vote Map, to store all the votes for a VM. This map stores votes of all the hosts currently in the data center and is used to determine which host is ideal for hosting a given VM.

Two new classes have been added to CloudSim's existing classes, to not disturb the internal working of the rest of the setup.

1. CloudVMLoadBalancer

The "CloudVMLoadBalancer" class is the main class of this system. It is responsible for creating the objects of Data center, Hosts, VMs, and Cloudlets. This class is responsible for starting the CloudSim simulation and passing in an allocation policy object to the Data Center class.

This class holds some very important data structures used in this project. Following are the data structures from the class CloudMLoadBalancer:
1.1 VM List – It is a list of all the created VMs in the Data center.
1.2 Cloudlet list – It is a list of all the user tasks / cloudlets which are to be mapped to the VMs.
1.3 Host List – it is the list of all the hosts in the data center.

2. AllocationPolicy

The "AllocationPolicy" class is the class that is responsible for assigning the caller specified allocation scheme and finding the ideal host for the VM. The AllocationPolicy class inherits CloudSim's "VMAllocationPolicy" class which is responsible for the internal logic of VM creation. This class holds an internal method of CloudSim, "allocateHostForVM," which takes input as the VM to be allocated and chooses a data center specified allocation scheme to choose an ideal host for the VM. This class holds all the three allocation scheme methods, First fit consensus, Best fit consensus, and Worst fit consensus. Depending on the caller supplied allocation scheme, one of these methods is called and the given VM is allocated according to the allocation scheme.

The AllocationPolicy class holds five key data structures of this system. Following are the data structures from the class AllocationPolicy:
2.1 VM table - This table maps every VM to its allocated host.
2.2 Used RAM map – This map the amount of RAM every VM has used.

2.3 Free RAM map – This map stores the free RAM of every host. It is initialized to have values equal to the total available RAM on a host and then as VMs are allocated to the hosts this map is updated. This map is used to determine if a host as enough memory to allocate a new VM.
2.4 Host Allocation table – This table keeps track of all the hosts who have a VM allocated to them. This table is used to implement load balancing in the system. This table is used to prevent a host from being overcrowded with VMs.
2.5 Unallocated VMs – Any unallocated VM is added to this Queue. This queue is checked periodically to see if an unallocated VM is present, and if so, it is allocated to the next available host.

The "deallocateHostForVM" method is also a part of this class. The "deallocateHostForVM" method is used to deallocate a host after the tasks on the VM have been completed. Whenever a host becomes free this method checks if there is any unallocated VM in the "unallocated VM Queue" and queues up the VM for allocation again.

B. Implementation of Algorithms

By default, CloudSim uses First Come First Serve and Round Robin as its VM allocation strategy [16]. This project proposes three new consensus-based algorithms for VM allocation, to implement load balancing in the data center. This section provides the algorithms that have been proposed in the project.

1. First Fit Consensus Algorithm

The First Fit Consensus algorithm allocates VMs to hosts in a first come first serve manner. The input to the algorithm is the list of VMs.

In the first stage of the algorithm, for every VM, the algorithm checks if a host has already been allocated. If the host has the capacity to hold that VM, it votes for that VM, according to a FCFS strategy. This takes place for all the hosts in the hostslist. Each host has their own votemap that stores the votes. At the end of stage one of the algorithm, the hosts reach a consensus and the host with the maximum votes is selected for the allocation of that VM. The allocated VMs are removed from the list of VMs.

The second stage of the algorithm begins when every host has been allocated a VM at least once. This is done for achieving a balanced distribution of load between all the hosts. For the remaining VMs, consensus is carried out among the hosts in a manner similar to stage one, in accordance with the first come first serve allocation strategy.

Algorithm

Input - VM to be allocated.
a. set result = false.
b. Check if all hosts have VMs allocated to them from the Host Allocation Table.
    y: If all hosts have VMs allocated, then make all hosts available to accept more VMs.
c. Iterate through hosts list
d. Check if the host has no VM already allocated to it in the HostAllocationTable
    y: if available capacity of host is more than the requirement of VM,
        y: current host will Vote for itself and put the vote in current hosts VoteMap .
e. Since one host has already voted for itself, all other Hosts also vote for this host, and the VoteMap for all hosts is updated.
f. Get the maximum voted host to create the VM.
g. if VM was created successfully on the maximum voted host.
    y: update the VM table with the current VM id and the max voted host
    update the usedRAM table with the required RAM of the VM.
    update the freeRAM table of the host to show the new RAM after subtracting the required RAM of the VM from the free RAM of the host.
    update the host allocation table with the host, saying there is a VM allocated to this host.
    result = true
h. return result

2. Best Fit Consensus Algorithm

The Best Fit Consensus algorithm allocates VMs to hosts in a way that the allocated VMs utilize the maximum amount of memory on the host machine. The input to the algorithm is the list of VMs.

In the first stage of the algorithm, for every VM, the algorithm checks if a host has already been allocated. If the host has the capacity to hold that VM, it votes for that VM, according to the Best Fit strategy. The algorithm checks the difference between the available host capacity and the memory requirement of the VM and selects the host with the minimum difference. This takes place for all the hosts in the hostslist. Each host has their own votemap that stores the votes. At the end of stage one of the algorithm, the hosts reach a consensus and the host with the maximum votes is selected for the allocation of that VM. The allocated VMs are removed from the list of VMs.

The second stage of the algorithm begins when every host has been allocated a VM at least once. This is done for achieving a balanced distribution of load between all the hosts. For the remaining VMs, consensus is carried out among the hosts in a manner similar to the stage one, in accordance with the best fit allocation strategy.

Algorithm

Input - VM to be allocated.

a. set result = false and bestIdx = -1
b. Check if all hosts have VMs allocated to them from the Host Allocation Table.
    y: If all hosts have VMs allocated, then make all hosts available to accept more VMs.
    n: if not, just continue
c. Iterate through hosts list
d. Check if the host has no VM already allocated to it in the HostAllocationTable
    y: if available capacity of host is more than the requirement of VM,
        y: current host will Vote for itself and put the vote in current hosts VoteMap
e. if bestIdx = -1
    y: set best idx = current hosts index
    n: check if bestIdx hosts memeory > current host
        y: set bestIdx = current idx.
f. check if bestIdx is not -1
    y: get the host corresponding to the bestIdx from the host list.
    All the hosts will vote for the bestIdx host and all hosts will get the updated VoteMap.
g. Get the maximum voted host to create the VM.
h. if VM was created successfully on the maximum voted host.
    y: update the VM table with the current VM id and the max voted host
    update the usedRAM table with the required RAM of the VM.
    update the freeRAM table of the host to show the new RAM after subtracting the required ram of the VM from the free RAM of the host.
    update the host allocation table with the host, saying there is a VM allocated to this host.
    result = true
i. return result

3. Worst Fit Consensus Algorithm

The Worst Fit Consensus algorithm allocates VMs to hosts in such a way that the maximum number of VMs get allocated to a given host. The input to the algorithm is the list of VMs.

In the first stage of the algorithm, for every VM, the algorithm checks if a host has already been allocated. If the host has the capacity to hold that VM, it votes for that VM, according to the Worst Fit strategy. The algorithm checks the difference between the available host capacity and the memory requirement of the VM and selects the host with the maximum difference. This takes place for all the hosts in the hostslist. Each host has their own votemap, that stores the votes. At the end of stage one of the algorithm, the hosts reach a consensus and the host with the maximum votes is selected for the allocation of that VM. The allocated VMs are removed from the list of VMs.

The second stage of the algorithm begins when every host has been allocated a VM at least once. This is done for achieving a balanced distribution of load between all the hosts. For the remaining VMs, consensus is carried out among the hosts in a manner similar to the stage one, in accordance with the worst fit allocation strategy.

Algorithm

Input - VM to be allocated.
a. set result = false and worstIdx = -1
b. Check if all hosts have VMs allocated to them from the Host Allocation Table.
    y: If all hosts have VMs allocated, then make all hosts available to accept more VMs.
    n: if not, just continue
c. Iterate through hosts list
d. Check if the host has no VM already allocated to it in the HostAllocationTable
    y: if available capacity of host is more than the requirement of VM,
        y: current host will Vote for itself and put the vote in current hosts VoteMap
e. if worstIdx = -1
    y: set worst idx = current hosts index
    n: check if worstIdx hosts memeory > current host
        y: set worstIdx = current idx.
f. check if worstIdx is not -1
    y: get the host corresponding to the worstIdx from the host list.
    All the hosts will vote for the worstIdx host and all hosts will get the updated VoteMap.
g. Get the maximum voted host to create the VM.
h. if VM was created successfully on the maximum voted host.
    y: update the VM table with the current VM id and the max voted host
    update the usedRAM table with the required RAM of the VM.
    update the freeRAM table of the host to show the new RAM after subtracting the required RAM of the VM from the free RAM of the host.
    update the host allocation table with the host, saying there is a VM allocated to this host.
    result = true
i. return result

C. Test Cases

This section provides the test cases and their outcomes, for each of the three algorithms.

1. Test Case 1: Available HostSize > VM RAM requirement

Test Data:
To test the outcomes, the number of hosts taken was five, with a memory of 2048 MB each. Ten VMs, each of size 800MB, were created.

This is the ideal test case, where VMs get mapped according to the allocation scheme.

Table 1. Test Case 1

Allocation Scheme	Expected Outcome	Observed Outcome
First Fit	All the VMs get allocated	All the VMs get allocated
Best Fit	All the VMs get allocated	All the VMs get allocated
Worst Fit	All the VMs get allocated	All the VMs get allocated

2. Test Case 2: Available HostSize < VM RAM Requirement

Test Data:
To test the outcomes, the number of hosts taken was five, with a memory of 2048 MB each. Five VMs of size 1048 MB and five VMs of sizes ranging from 800 MB to 1200 MB were created.

Table 2. Test Case 2

Allocation Scheme	Expected Outcome	Observed Outcome
First Fit	Unallocated VMs get queued and are allocated once a host is free	3 VMs were left unallocated and get queued and are allocated once a host is free
Best Fit	Unallocated VMs get queued and are allocated once a host is free	1 VMs was left unallocated and queued and was allocated once a host is free
Worst Fit	Unallocated VMs get queued and are allocated once a host is free	2 VM were left unallocated and get queued and are allocated once a host is free

3. Test Case 3: Available HostSize = VM RAM Requirement

Test Data:
To test the outcomes, the number of hosts taken were five, with a memory of 2048 MB each. Ten VMs, each of size equal to HostSize, were created.

Table 3. Test Case 3

Allocation Scheme	Expected Outcome	Observed Outcome
First Fit	5 VMs were left unallocated and queued up for reallocation	5 VMs were left unallocated and queued and are allocated once a host is free
Best Fit	5 VMs were left unallocated and queued up for reallocation	5 VMs were left unallocated and queued and are allocated once a host is free
Worst Fit	5 VMs were left unallocated and queued up for reallocation	5 VMs were left unallocated and queued and are allocated once a host is free

4. Test Case 4: Random VM RAM requirements

Test Data:
To test the outcomes, the number of hosts taken was five, with a memory of 2048 MB each. Ten VMs, of random sizes, were created.

Table 4. Test Case 4

Allocation Scheme	Expected Outcome	Observed Outcome
First Fit	All the VMs get allocated	3 VMs were left unallocated and queued and are allocated once a host is free
Best Fit	All the VMs get allocated	1 VMs was left unallocated and queued and was allocated once a host is free
Worst Fit	All the VMs get allocated	2 VMs were left unallocated and queued and are allocated once a host is free

V. RESULTS AND ANALYSIS

For testing and evaluation, the simulation setup consisted of a total of five hosts within the data center. Each host had an initial memory of 2048 MB. The total number of VMs created was 20 and the total number of cloudlets created was 400. Thus, each VM was mapped to 20 cloudlets.

A. First Fit Consensus

Table 5. VM Allocation for First Fit Consensus

Hosts	Host 1	Host 2	Host 3	Host 4	Host 5
Allocated VMs	0, 5, 10, 16	1, 6, 11, 15	2, 7, 12, 19	3, 8, 13, 17	4, 9, 14, 18
Total Size of VMs (MB)	1525	1175	2175	2700	3000

Table 5 presents the VMs allocated and the total VM allocation on each host. The total VM allocation is the addition of the sizes of all the VMs that have been allocated on the host.

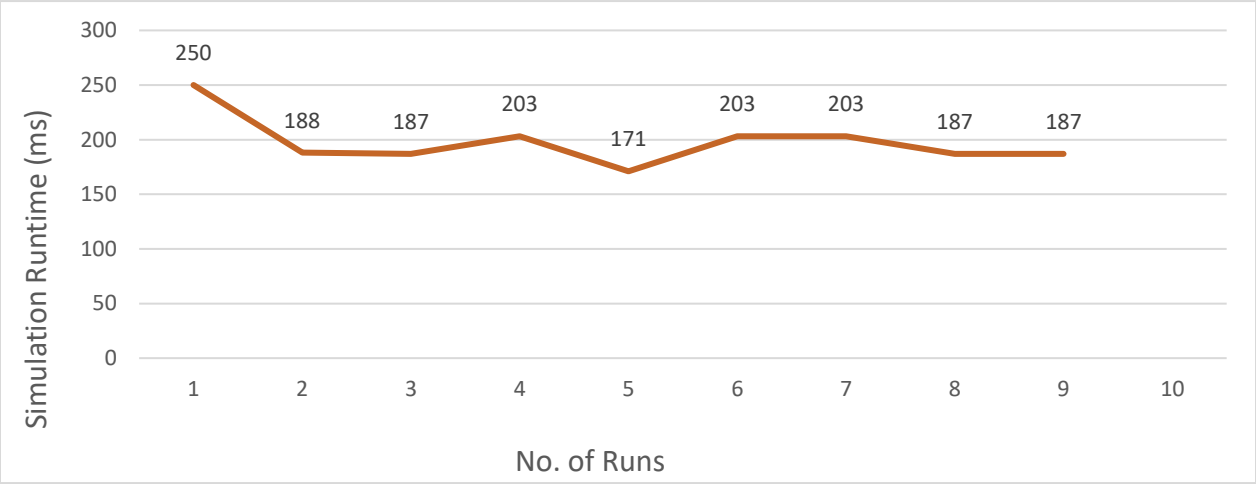

Figure 4. Simulation runtime for First Fit Consensus

The graph in Figure 4 presents the average simulation runtime taken by the algorithm, based on ten runs of the simulation.

Average Simulation Runtime = 250 + 188 + 187 + 203 + 171 + 203 + 203 + 187 + 187 + 188 / 10 = 191.7 ms

B. Best Fit Consensus

Table 6. VM Allocation for Best Fit Consensus

Hosts	Host 1	Host 2	Host 3	Host 4	Host 5
Allocated VMs	0, 5, 10, 17	1,9, 11, 15	2, 8, 13, 16	3, 7, 14, 19	4, 6, 12, 18
Total Size of VMs (MB)	2300	2000	1875	2275	2700

Table 6 presents the VMs allocated and the total VM allocation in Megabytes (Mb) on each host. The total VM allocation is the addition of the sizes of all the VMs that have been allocated on the host.

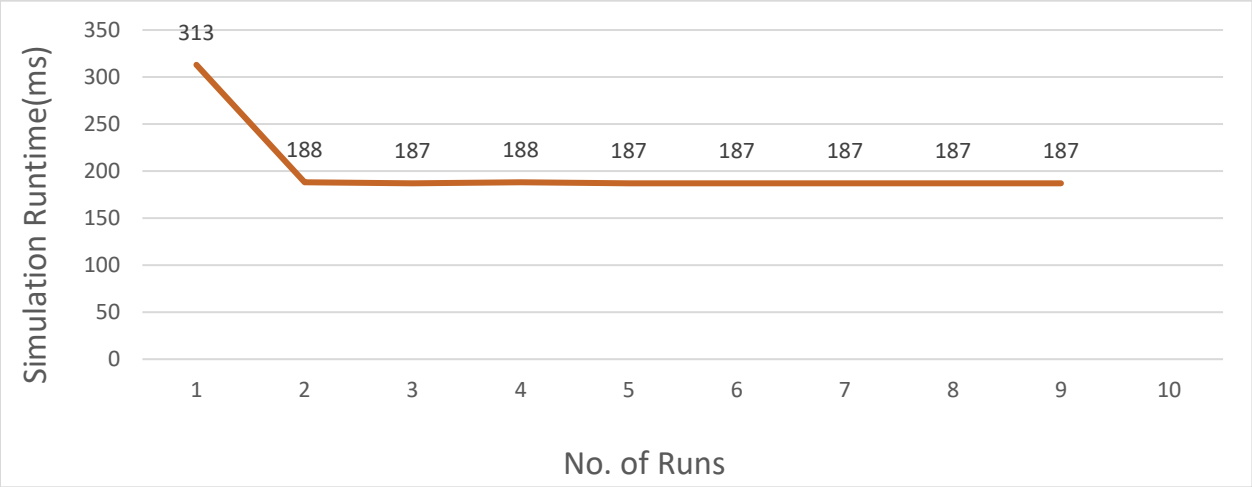

Figure 5. Simulation runtime for Best Fit Consensus

The graph in Figure 5 presents the average simulation runtime in milliseconds (ms) taken by the algorithm, based on ten runs of the simulation.

Average Simulation Runtime = 313 + 188 + 187 + 188 + 187 + 187 + 187 + 187 + 187 + 187 / 10 = 218.5 ms

C. Worst Fit Consensus

Table 7. VM Allocation for Worst Fit Consensus

Hosts	Host 1	Host 2	Host 3	Host 4	Host 5
Allocated VMs	0, 9, 14, 18	1, 5, 12, 17	2, 6, 11, 16	3, 7, 10, 15	4, 8, 13, 19
Total Size of VMs (MB)	1525	2650	1275	1600	2675

Table 7 presents the VMs allocated and the total VM allocation in Megabytes (MB) on each host. The total VM allocation is the addition of the sizes of all the VMs that have been allocated on the host.

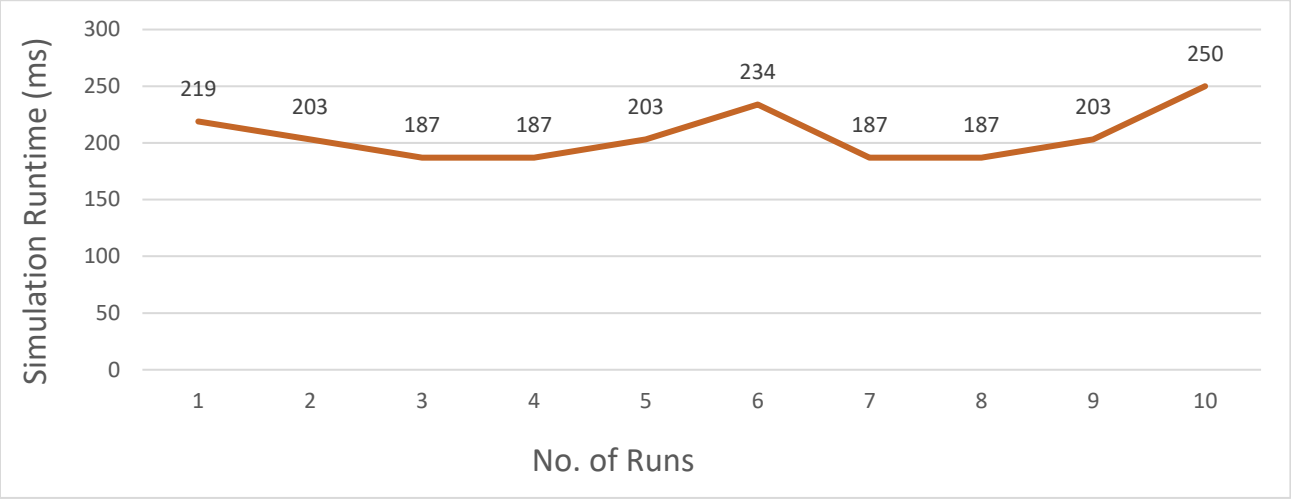

Figure 6. Simulation runtime for Worst Fit Consensus

The graph in Figure 6 presents the average simulation runtime in milliseconds (ms) taken by the algorithm, based on ten runs of the simulation.

Average Simulation Runtime = 219 + 203 + 187 + 187 + 203 +234 + 187 + 187 + 203 + 250 / 10 = 206 ms

D. Comparison and Analysis

1. Simulation Runtime

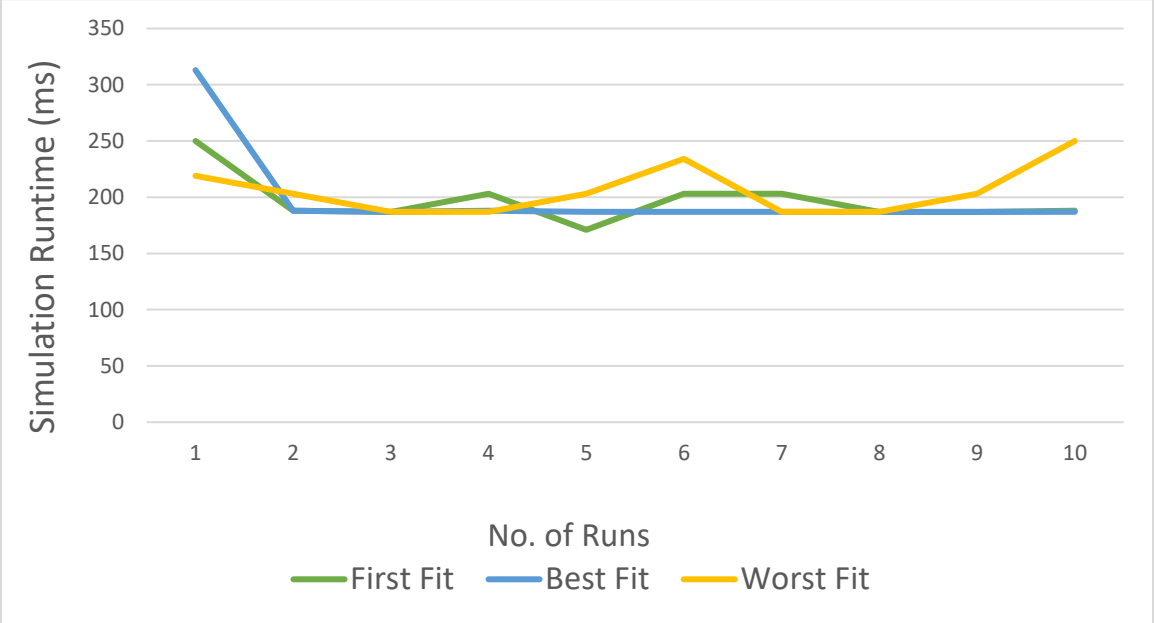

Figure 7. Comparison of simulation runtime

The graph in Figure 7 presents a comparison between the simulation runtimes of all three algorithms.

Average Simulation Runtime for First Fit Consensus algorithm = 250 + 188 + 187 + 203 + 171 + 203 + 203 + 187 + 187 + 188 / 10 = 191.7 ms

Average Simulation Runtime for Best Fit Consensus algorithm = 313 + 188 + 187 + 188 + 187 + 187 + 187 + 187 + 187 + 187 / 10 = 218.5 ms

Average Simulation Runtime for Worst Fit Consensus algorithm = 219 + 203 + 187 + 187 + 203 +234 + 187 + 187 + 203 + 250 / 10 = 206 ms

From the above data, it is evident that the algorithms in the increasing order of time consumption are: First Fit Consensus, Worst Fit Consensus and Best Fit Consensus. The First Fit Consensus algorithm takes the least amount of time while the Best Fit Consensus algorithm takes the most amount of time.

2. Total VM Allocation

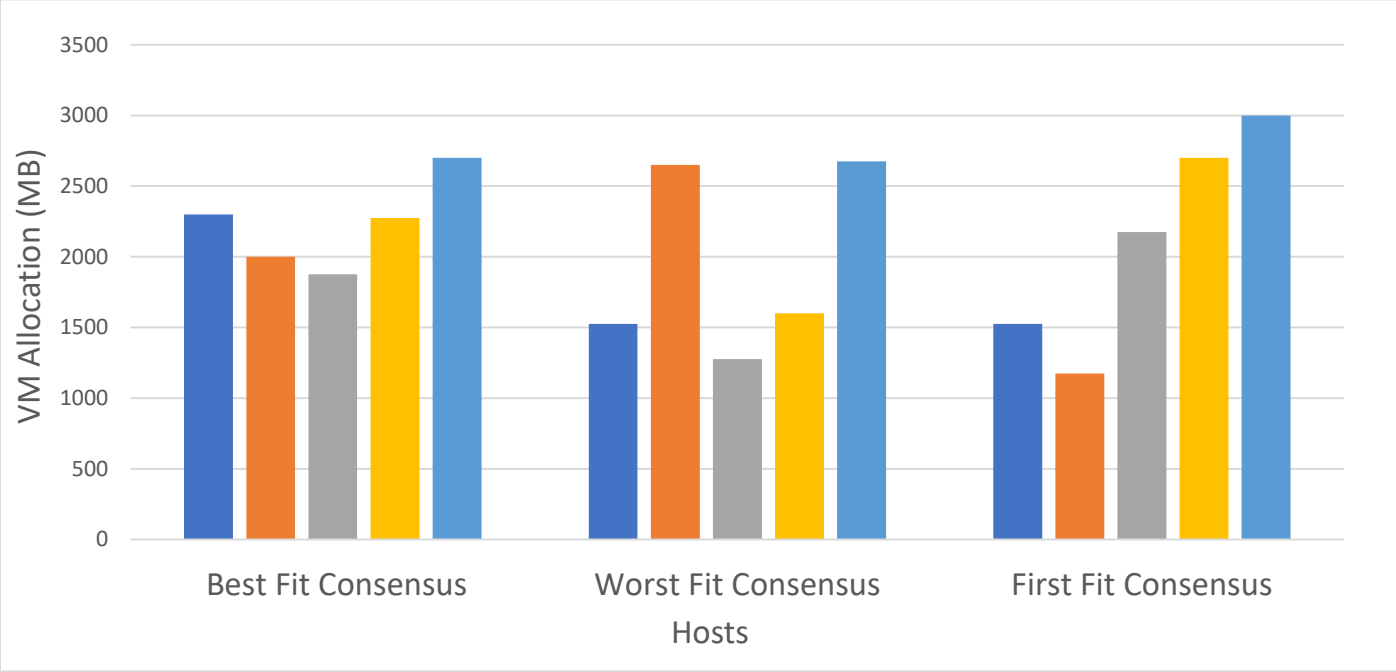

Figure 8. Comparison of VM Allocation for Load Balancing

The graph in Figure 8 presents the comparison between the total VM allocation for each consensus algorithm.

For the First Fit Consensus algorithm, the difference between the minimum and maximum size of hosts after VM allocation is 1475 MB. For the Best Fit Consensus algorithm, the difference between the minimum and maximum size of hosts after VM allocation is 400 MB. For the Worst Fit Consensus algorithm, the difference between the minimum and maximum size of hosts after VM allocation is 1150 MB.

From this data it is evident that the load is most balanced with the Best Fit Consensus algorithm and least balanced with the First Fit Consensus algorithm. The algorithms in the decreasing order of load balancing are: Best Fit Consensus, Worst Fit Consensus, First Fit Consensus.

Between the three algorithms, the Best Fit Consensus algorithm achieves the most balanced distribution of load. However, it also exhibits the maximum simulation runtime. Therefore, this algorithm can be useful in the case where service delivery while maintaining a balanced load is the primary concern in the data center. The First Fit algorithm exhibits a comparatively lesser degree of load balancing than the other two algorithms. However, since it completes its execution with the least simulation runtime, it can be preferred in the case where service delivery while prioritizing time criticality is the primary concern in the data center. The Worst Fit Consensus

algorithm exhibits an average performance in terms of both load balancing as well as simulation runtime.

VI. CONCLUSION AND FUTURE WORK

With the exponential rise in the number of consumers of cloud services, there has been a proportional increase in the demand for more data processing. While cloud service providers look for increased profits, consumers look for features such as faster and reliable service delivery. This puts a lot of load on cloud data centers. This gives rise to the possibility of over provisioning or underutilization of resources. Therefore, load balancing within the cloud data center becomes a vital concern.

This work studied and examined several approaches related to load balancing strategies in the context of distributed and cloud systems. It proposed three load balancing strategies based on consensus. These load balancing algorithms essentially depend on the VM allocation scheme. In the first fit approach, the VMs are allocated to the hosts in the data center in the first come first serve manner. In the best fit approach, the VMs are allocated to the host that can hold the maximum VM size. The worst fit approach aims at allocating the maximum number of VMs on a given host. The simulation of architecture, development and testing of the algorithms has been done in Java, using CloudSim. The simulation results revealed Best Fit Consensus algorithm to be the best in terms of load balancing, with a time penalty. The First Fit Consensus algorithm achieved the least amount of load balancing but took comparatively less time to run. The Worst Fit Consensus algorithm showcased an average performance in terms of both time and load balancing. The load balancing has been achieved in the context of RAM availability and requirements of the hosts and VMs, respectively.

This project has provided a novel understanding of the ways consensus approaches can work with distributed resources. However, there are many different areas where future work is possible. The concept of consensus load balancing used in this project can be extended to an inter-data center implementation. These data centers could be geographically distributed. Also, the load balancing has been achieved with respect to RAM requirements. It would be interesting to test the algorithms in the context of CPU processing power and network bandwidth. In addition to this, the three algorithms could be tested and evaluated for their power consumption, to consider a more power efficient load balancing approach. The algorithms could be extended and tested in different systems like Radio Access Networks, Wireless Area Networks, etc.

References

[1] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, NIST Std., Jan 2011.

[2] S. Sonkar and M. Kharat, "A Review on Resource Allocation and VM Scheduling Techniques and a Model for Efficient Resource Management in Cloud Computing Environment," in *ICTBIG*, Indore, India, Nov. 18-19, 2016, pp. 1-7, doi: 10.1109/ICTBIG.2016.7892646.

[3] M. Ullrich *et al.*, "Towards Efficient Resource Management in Cloud Computing: A Survey," presented at the 2016 IEEE 4th Int. Conf. FiCloud, Vienna, Austria, Aug. 22-24, 2016, pp. 170-177, doi: 10.1109/FiCloud.2016.32.

[4] S. Kadioglu *et al.*, "Heterogeneous Resource Allocation in Cloud Management," presented at the 2016 IEEE 15th Int. Symposium on NCA, Cambridge, USA, Oct. 31-Nov. 2, 2016, pp. 35-38, doi: 10.1109/NCA.2016.7778589.

[5] S. Aslam and M. Shah, "Load Balancing Algorithms in Cloud Computing: A Survey of Modern Techniques," presented at the 2015 NSEC, Rawalpindi, Pakistan, Dec. 17, 2015, pp. 30-35, doi: 10.1109/NSEC.2015.7396341.

[6] A. Gupta and R. Garg, "Load Balancing Based Task Scheduling with ACO in Cloud Computing," presented at the 2017 *ICCA*, Doha, UAE, Sept. 6-7, 2017, pp. 174-179, doi: 10.1109/COMAPP.2017.8079781.

[7] S. Khara and U. Thakkar, "A Novel Approach for Enhancing Selection of Load Balancing Algorithms Dynamically in Cloud Computing," presented at the 2017 Comptelix, Jaipur, India, Jul. 1-2, 2017, pp. 44-48, doi: 10.1109/COMPTELIX.2017.8003935.

[8] A. Ailijiang *et al.*, "Consensus in the Cloud: Paxos Systems Demystified," presented at the 2016 ICCCN, HI, USA, Aug. 1-4, 2016, pp. 1-10, doi: 10.1109/ICCCN.2016.7568499.

[9] L. Lamport, "Paxos Made Simple," Nov. 2011.

[10] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm (Extended Version)." Online available: https://web.stanford.edu/~ouster/cgi-bin/papers/raft-extended.pdf .

[11] P. Marandi *et al*., "The Performance of Paxos in the Cloud," presented at 2014 IEEE 33rd International Symposium on Reliable Distributed Systems, Nara, Japan, Oct. 6-9, 2014, pp. 41-50, doi: 10.1109/SRDS.2014.15

[12] A. Loon and M. Mahyuddin, "Network Server Load Balancing using Consensus-based Control Algorithm," presented at the 2016 IEEE IEACon, Malaysia, Nov. 20-22, 2016, pp. 291-296, doi: 10.1109/IEACON.2016.8067394.

[13] P. Mudialba, "A Study on the fundamental properties, features and

usage of cloud simulators," presented at the 2016 PlatCon, Jeju, South Korea, Feb. 15-16, 2016, pp. 1-5, doi: 10.1109/PlatCon.2016.7456782.

[14] A. Maarouf *et al*., "Comparative Study of Simulators for Cloud Computing," presented at the 2015 Int. Conf. CloudTech, Marrakech, Morocco, Jun. 2-4, 2015, pp. 1-8, doi: 10.1109/CloudTech.2015.7336989.

[15] R. Calheiros *et al.*, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23-50, Jan. 2011, doi: 10.1002/spe.995.

[16] S. Santra, K. Mali, "A New Approach to Survey on Load Balancing in VM in Cloud Computing: using CloudSim," presented at the 2015 Int. Conf. IC4, Indore, India, Sept. 10-12, 2015, pp. 1-5, doi: 10.1109/IC4.2015.7375671.

[17] J. Wang *et al*., "A Heuristics-based VM Allocation Mechanism for Cloud Data Centers," presented at the 2017 IEEE ISCAS, Baltimore, USA, May 28-31, 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050470.

[18] A. Chatterjee *et al*., "Job Scheduling in Cloud Datacenters Using Enhanced Particle Swarm Optimization," presented at the 2017 2nd I2CT, Mumbai, India, April 7-9, 2017, pp. 895-900, doi: 10.1109/I2CT.2017.8226258.

[19] V. Velde, B. Rama, "Simulation of Optimized Load Balancing and User Job Scheduling Using CloudSim," presented at the 2017 2nd IEEE Int. Conf. on RTEICT, Bangalore, India, May 19-20, 2017, pp. 1379-1384, doi: 10.1109/RTEICT.2017.8256824.

[20] E. Rani, H. Kaur, "Study on fundamental usage of CloudSim simulator and algorithms of resource allocation in cloud computing," presented at the2017 8th Int. Conf. on ICCCNT, Delhi, India, 3-5 July 2017, pp. 1-7, doi: 10.1109/ICCCNT.2017.8203998.

[21] Online Available: https://www.researchgate.net/figure/Architecture-of-CloudSim-18_fig2_319392288, accessed: May, 16, 2018.