Hybrid Resource Scaling for Dynamic Workload in Cloud Computing Environment

By: Megersa Daraje

A Thesis submitted to the

Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

Adama

September, 2018

Hybrid Resource Scaling for Dynamic Workload in Cloud Computing Environment

By: Megersa Daraje

Advisor: Dr. Ravindra Babu

A Thesis submitted to the

Department of Computer Science and Engineering

School of Electrical Engineering and Computing

Presented in partial fulfillment of the requirement for the Degree of Masters of Science in Computer Science and Engineering

Office of Graduate Studies

Adama Science and Technology University

Adama

September, 2018

Declaration

I hereby declare that this MSc Thesis is my original work and has not been presented for a degree in any other university, and all sources of material used for this thesis have been duly acknowledged.

Name: _____

Signature: _____

This MSc Thesis has been submitted for examination with my approval as thesis advisor.

Name: _____

Signature: _____

Date of submission: _____

Approval of Board of Examiners

We, the undersigned, members of the Board of Examiners of the final open defense by_____have read and evaluated his/her thesis entitled "_____" and examined the candidate. This is, therefore, to certify that thesis has been accepted in partial fulfillment of the requirement of the Degree of ……...

_____	_____	_____
Supervisor/Advisor	Signature	Date
_____	_____	_____
Chairperson	Signature	Date
_____	_____	_____
Internal Examiner	Signature	Date
_____	_____	_____
External Examiner	Signature	Date

Acknowledgement

First of all, thanks to my Almighty God for the help to come up at this stage.

The second thanks would go to Dr. Mesfin Abebe the coordinator of PG study in the department of computing for his basic suggestion starting up to end. I am also thanks my advisor, Dr. Ravindra Babu for his guidance and constructive suggestions that are the major elements for the completion of this thesis.

My special thanks to Desta Zerihun (MSc), Endale Aragu (MSc), and Antene Alemu Lecturer at Adama Science and Technology University for his valuable advice, basic and concrete suggestions and comments on this thesis.

I am also indebted to give other special thanks staff members of Cloud Computing and Distributed System SIG; they provided me the knowledge and supported me not only during my Masters study but also BSc study.

My heart felt also goes to my friends Mekuria Gemechu, Mubarek Dubiso and all my family's in general for their supportive in all things.

At the last but not the least, I never forget to acknowledge my lovely **Derartu Jabessa** for her support and encouragement for the accomplishment of this thesis and my lovely mother **Abelu Kenea** for her endless support from I was a Child up to now. Just I want say "**I love you more than I can say**".

Contents

List of Figures

List of Tables

List of Acronyms

API	Application Programming Interface
App	Application
AWS	Amazon Web Services
CC	Cloud Computing
CPU	Central Processing Unit
CVM	Cost of Virtual Machine
EC2	Elastic Compute Cloud
GUI	Graphical User Interface
HLL	High Level Language
HW	Hardware
IT	Information Technology
I/O	Input/output
ISA	Instruction Set Architecture
MIs	Millions of Instructions
PDA	Personal digital assistant
RAM	Random access memory
SLA	Service Level Agreement
SW	Software
TCVM	Total Cost of Virtual machines
VM	Virtual Machine

Abstract

Cloud Computing service providers have sufficient amount of resources on their resource pool to be provided for the cloud users. This resource should be scaled up and scaled down to satisfy cloud users demand. Cloud service provider use two methods to scaling the resources. These methods are; horizontal scaling and vertical scaling method. Horizontal scaling scheme for scale up takes a minute to configure an extra virtual machines and low resource utilization. But the positive side of horizontal scaling is, it has availability. The second option are vertical scaling method, scaling resources are added on running virtual machines and it takes second for the configuration.

The goal of this study is to propose the hybrid of both resource scaling approach to maximize resource utilization and satisfy users demand in cloud environment by performing vertical scaling and horizontal scaling approach respectively. We use CloudSim tool kit to implement the proposed approach. To make resource scaling we are following the threshold value of the resources.

Keywords: - Cloud Computing, Vertical Scaling, Virtualization, Auto scaling, Scalability, Horizontal scaling.

CHAPTER ONE

1. Introduction

1.1. Background

Computing based on the Internet using shared resources is called cloud computing. Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort [1]. The underlying concept of cloud computing is the separation of applications from the operating systems and the hardware on which they run [2].

Cloud computing deliver applications via the internet, which is accessible from web browsers, desktop and mobile apps, while the software and data are stored on servers at a remote location.

In the past, many of us worried about losing our documents, photos, and files if something bad happened to our computers, like a virus or a hardware malfunction. Today, our data is migrating beyond the boundaries of our personal computers and all our data would still safely reside on the web, accessible from any Internet-connected computer, anywhere in the world because of cloud computing.

Many different companies and independent individuals have used and given definitions for cloud computing corresponding to their products, strategies, and visions. A general working principle of cloud computing as cloud computing is "a model for accepting suitable, on-demand network access to a shared pool configurable computing resources (i.e. applications, storage, servers, and networks) that can be quickly provisioned and with least service provider contact." According to the above definition, there are five main cloud characteristics, four classes of deployment models and three types of service models. Five main characteristics of cloud computing are - resource pool, on-demand self-service, scalability and elasticity, broad-network access and pay-per-use. Based on the type of cloud computing model, cloud computing can be divided into four: - private cloud, public cloud, hybrid cloud and community cloud models [3].

There are different cloud providers, which give their service with fixed cost within an hour for what the user uses the resource on the cloud by renting CPU cycle. Today, the cloud computing paradigm has become one of the most interesting technologies with the virtualized concept. This is a technique to create an abstract layer of system resource and hides the difficulty of hardware and software working environment, typically, as virtual machines with processing unit networking

connectivity, storage, and bandwidth. Simply it is a computer within a computer, implemented in software. Virtualization commonly implemented with hypervisor technology that enables several operating systems to simultaneously run on a computer system. The cloud determines how those virtualized resources are delivered, allocated, and presented. In the cloud, different applications and information is a dump. It has many benefits over traditional data center. As cloud defined in [3] attributes for delivering computing services are:

- The security and maintenance services are offered by cloud providers,
- It uses the Internet to offer elastic services which are dynamically getting a resource on different workload,
- It provides a massive infrastructure to support elastic services,
- It is cost-efficient due to resource multiplexing.

The cloud enables its users to reach the best resource utilization by scaling vertically and horizontally according to their structure. Vertical scaling is adding resource on running virtual machine when there is resource shortage on running virtual machine. Horizontal scaling is creating a new virtual machine when there is resource shortage by copying application instance into new virtual machines, it takes its own time to configure virtual machine [5].

Cloud is beneficial for users, because cloud computing is virtualized technologies, that has better service and collaboration, with the wide availability of high bandwidth networks and high capacity of storage. This makes a cloud to work in the web browser as a thin client that allows users to run any application. The resources might be computational power, has more storage, inter-networked, and run huge applications. Therefore, the IT resource is elastic. Elasticity (Auto-scaling) is the ability of the system dynamically adjusts its available resources to handle the variable workload changes over a time [6].

In the recent years, cloud computing has begun to be very useful computing model in which some services are available for the users such as, virtualized and scalable resources. There are vast of cloud infrastructure used by the internet users like Amazon EC2 [7], Microsoft Azure [8], IBM Blue [9], Google App Engine [10], and etc. Many clients and organizations are operating their business basic applications, deploy their applications and keeping sensitive data, infrastructure and service on data centers of cloud so that they access it through the internet. Cloud data centers prepare facilities that encourage large data processing.

The performance of cloud data center influences service quality available [11]. The performance of the cloud data center is improved by rising resource utilization. Cloud server shares their resources, for example, memory, CPU centers, network bandwidth and storage among these client items and web applications. Data center virtualization is the key innovations that can be utilized to increase resource usage. It is the most important technologies utilized in the cloud computing that provides scalable and on-demand computational resources to these cloud application. Server virtualization technology makes conceptual layers above physical infrastructure that is available on servers. Cloud applications are provisioned with the arrangement of Virtual Machine (VM) sharing physical resource. Statistically dividing the physical resources into Virtual machine according to top request of uses may prompt to poor usage of resources. Over-provisioning of the resources results that lower profit while under-provisioning does not satisfy consumer [12].

Clearly, the solution is scaling the resources based on workload request with no administration interference. Cloud server tries to fulfill clients' request and provide services for users by sharing virtualized structure. As the works loaded on cloud applications are not static (dynamic) in nature, the resource allocated to VM might be over-provisioned or under provisioned. At the point when the assigned resource for VM can't fulfill client request, extra resources are assigned by Virtual Machine (VM)-based scaling system. Scaling is the capability of an application to make ideal usage of resource at various workload levels by keeping over-provisioning, under-use and under-provisioning [13]. An expansive part of cloud applications (called occupants) may make its workload to increase sharply at some time or days or week. To manage such changeable load patterns, there is a need for dynamic scaling or flexibility with no any service interruption. According to the users' requirement, VMs can be created, configured, deployed and managed properly. At the point when many VMs are requested, new systems and tools for controlling VM become essential so as to automate the steps in cloud services. The open issue for cloud clients is creating and configuring a huge number of Virtual Machines (VMs) for particular reasons [14].

The main aim of this work is to propose hybrid resource scaling approach with respect to responsiveness, availability and optimal utilization of cloud resources.

1.2 Motivation

The allocation of the resource particularly affects the performance of work, availability, and responsiveness. Since, the workload on the cloud are dynamic, the resource allocated should be scale up and scale down according to the users need. The flexibility of the Virtual machines (VMs) is widely used to satisfy the requests of the cloud users. These scaling method can be performed by either increasing the number of VMs called horizontal scaling or increasing the size of resources in the single machine called vertical scaling [12]. Horizontal scaling methods creates another Virtual Machines (VMs) and distributed the coming work load work load among them to satisfy cloud users. But, vertical scaling method is resizing the available resources without adding an extra machine to allocate resources as well as to satisfy cloud users. This two types of scaling methods have their limitation like availability, cost, physical limitation, power consumption, affects users need, and etc. Depending and considering these limitations of both scaling methods, designing and developing hybrid scaling method by taking the advantages both scaling methods to satisfy cloud users and to maximize resource utilizations was proposed.

1.3. Statement of the problem

The workloads on Cloud may be dynamic in nature for this reason the resource assigned for Virtual Machine (VM) can be scaled down or scaled up to fulfill users request. There are few research works identified with the scalability of resources have been published lately.

Vertical scaling method has the advantage of cost whereas horizontal scaling has advantage of availability. In horizontal scaling method, workloads are handled by creating an extra Virtual Machines (VMs). Creating an extra Virtual Machines (VMs) are adding another VMs when the requested resource becomes increase. As creating VMs consumes network bandwidth, increases IO traffic, and costly. But, in the vertical scaling method, workloads are handled by resizing the resources within single machine with respect to the capacity of the virtual machine. Vertical scaling method also have limitations like, limited in size, limited in power and etc. Hence there is a need for further research that avoids the limitations of both scaling methods. The proposed method took the advantage of these scaling methods. Creating an extra resource are only when free resources are not available in the vertical scaling

1.4. The significance of the Study

Cloud computing is a computing paradigm where a large pool of systems is connected in private or public networks to provide dynamically scalable infrastructure for its application, data and file storage. With the advent of this technology, the cost of computation, application hosting content storage and delivery is reduced significantly. Cloud computing is a practical approach to experience direct cost benefits. Most providers give their service with fixed bundle instance and hourly billing system with horizontal scaling scheme. Even if some automatic scaling scheme, they still use horizontal scaling scheme but make with dynamically; that is why they call with automatic scaling. This system of delivery by cloud providers has its own effect on both customer and provider. The research is significant to all cloud computing users especially to the service providers and clients.

1.5. Objectives

1.5.1. General objective

The objective of this thesis is to design and implement a hybrid resource scaling approach for dynamic workloads in a cloud environment.

1.5.2. Specific objective

- Proposing architectural, algorithmic, design and implementation solutions of dynamic workloads in a cloud environment.
- To satisfy users demand whenever computing resource need arises.
- Proposing mathematical model of total cost of virtual machine created in case of horizontal scaling.
- Development of the prototype for proposed solutions using cloud simulation tool.
- Performance evaluation of the proposed solution.

1.6. Research Questions

The research is built upon the research question:

1. How hybrid cloud resource scaling techniques satisfy users demand?
2. What are the existing cloud resource scaling techniques performed?
3. How are the existing cloud resource scaling techniques performed?

1.7. Delimitation/Scope

1.7.1. Scope of the Study

In existing scaling methods effects various parameters such as performance, availability, cost, physical limitation and power limitations. In resource virtualization, dynamic scaling for cloud application is the key challenge for cloud providers. This work provides efficient scaling mechanism that postpones VM migration as long as possible. The proposing solution is a hybrid of vertical and horizontal techniques that consider current workload, allocated resources of application and available resources in the cloud.

1.7.2. Limitation of the Study

The study was encountered the following limitation throughout the research process. Due to the tight schedule, the study cannot accommodate the concept of SLA. The other constraint of the study is, the experiment and implementation of the proposed approach are performed only on cloudSim toolkit.

1.8. Research Process Procedures

In order to attain the above objectives and to answer the research questions, the following procedures and techniques is used.

- First, we identified the area and the problem to be solved.
- Second, we identified the research question to be answered.
- Third, in order to do this, we reviewed different articles, journals, books and sites.
- Fourth, we identified the required parameters to design and design the proposed solution.
- Fifth, we prepared the experimental environment using Cloudsim toolkit to conduct the experiments.
- Sixth, we documented the result that we got from the cloudSim toolkit.
- Finally, we analyzed the result that we got from the experiment. Also, we compared the performance of the existing algorithm by calculating the running time of the algorithm based on the input and future work.

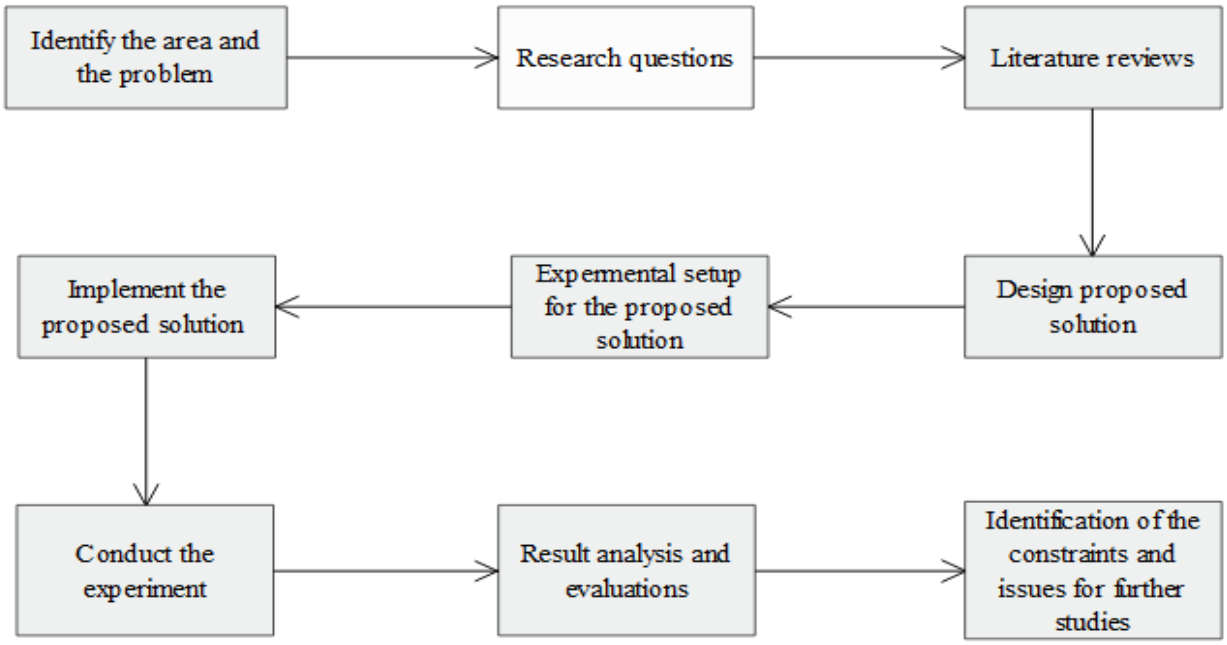

Figure 1. 1. Research process framework.

1.9. Organizations of Thesis

The structure of the thesis is as presented bellow.

- Chapter one describes the overall of research area, motivation, statement of the problem, objectives, beneficiaries of the study, and followed procedures to achieve the goals.
- Chapter two describes the literature review and related works.
- Chapter three describes the methodology used to achieve the goal of the research.
- Chapter four describes the proposed solution. In this part we design the algorithm. How the solution works also explained in detail.
- In Chapter five, the experimental setup and implementation are described. In this section, we try to see in detail how to set the experiment and what kind of hardware and software are used to implement the proposed solution and how to implement the proposed solution in a cloud environment.
- In Chapter six, result and evaluation are discussed. In this part, we analyzed the result gotten from the experiment performed and evaluating our algorithm with the existing approach.
- Chapter seven describes the conclusion and future work of the study.

CHAPTER TWO

2. Literature Review

A literature review is reading or finding the most related research articles for more understand, analyze, evaluate and compare to the research area. The main purpose of this literature review is to summarize and read the concept, information, and workflow of cloud resource scalability.

2.1. Overview of Cloud Computing

Experts in the cloud computing industry and providers their own definition for the term cloud computing. Indirectly the term used everywhere when we used the internet but it is not known by the consumers. Nowadays there is not yet general agreement for what the term means. Evaluating some of the existing definitions helps to clarify the term and what it involves. Below some of the definitions quoted from different companies and experts:

"Cloud computing enables convenient network access to a shared pool of configurable computing resources. This can be provisioned and launched with the minimum management effort or service provider interaction. This can be defined in plain terms is the capacity to reach convenient infrastructure for end users to utilize parts of bulk resources and the resources can be acquired quickly and easily. Due to the nature of infrastructure, application and soft wares being saved 'in the cloud which enables access from any fixed and movable devices like mobile, which can link up to the internet" [2][3].

Cloud computing is defined as a concept, where system users save the information on remote servers which is controlled and managed the applications which stored inside the server and accessed from remote locations.

2.2. Cloud Computing Architecture

Cloud computing architecture contains many cloud parts, which are loosely coupled. Cloud computing architecture is broadly divided into two parts: [15]

- Front End
- Back End

Each of the ends is connected through a network, typically the Internet. The following diagram demonstrates the graphical perspective of cloud computing architecture:

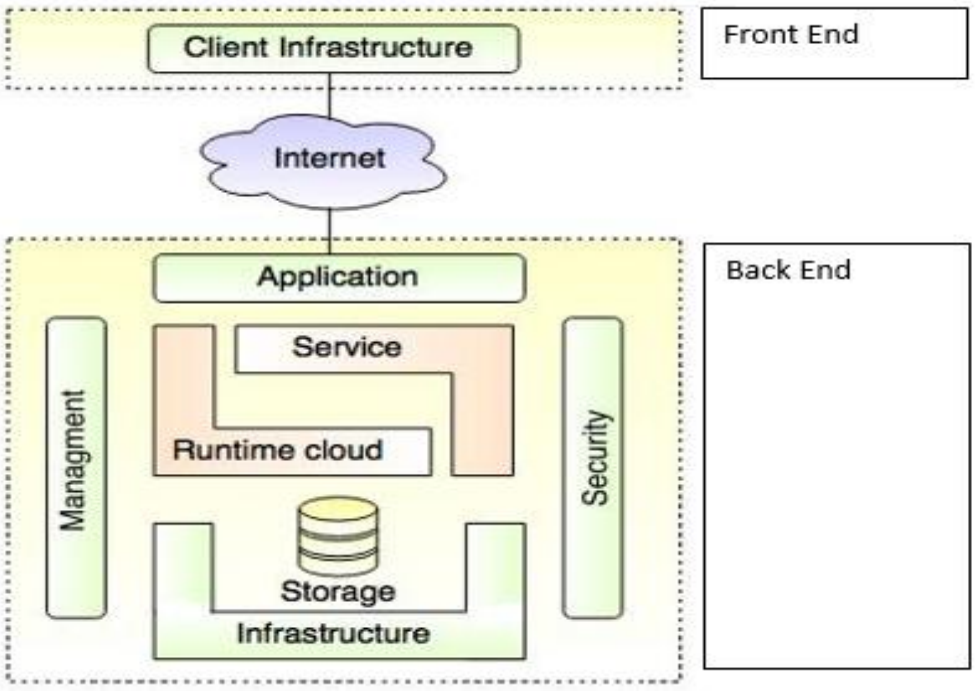

Figure 2. 1 Cloud Computing architecture [15].

2.2.1. Front End

The front end refers to the customer part of the cloud computing framework. It comprises of interfaces and applications that are required to get the cloud computing platforms, for example Web Browser.

2.2.2. Back End

The back end refers to the cloud itself. It comprises of the all required resources to give cloud computing services. It contains huge information storage, security mechanism, virtual machines, servers, services deployment models, etc.

2.3. Cloud Computing Infrastructure

Cloud computing framework contains servers, network, storage devices, cloud management software, platform virtualization, and deployment software [15].

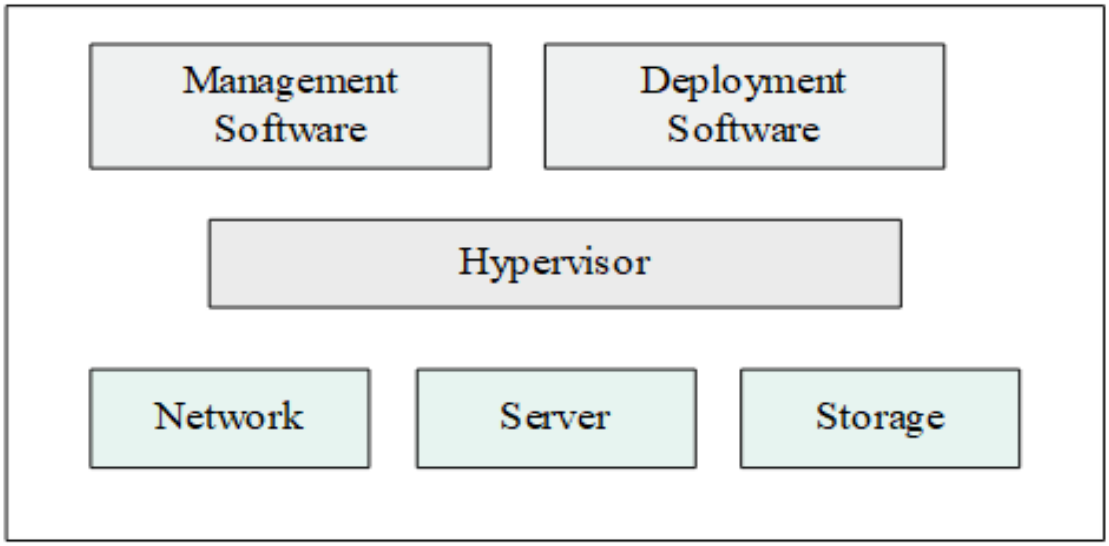

Figure 2. 2: Cloud Computing infrastructure [15].

2.3.1. Hypervisor

The hypervisor is a framework or low level that acts as a virtual machine Manager. It permits to share a single physical machine of cloud resources between several tenants.

2.3.2. Management Software

This management software supports to maintain and configure the framework or infrastructure.

2.3.3. Deployment Software

The main duty of deployment software is to support to deploy and integrate the application on the cloud computing.

2.3.4. Network

It is the key element of cloud infrastructure. It allows connecting cloud services over the Internet. It is additionally possible to deliver network as a utility over the Internet, which implies, the customer can modify the network route and protocol.

2.3.5. Server

The server processes the resource sharing and offers different services, for example, resource allocation and resource de-allocation, monitoring the resources, providing security and etc.

2.3.6. Storage

Cloud computing keeps various copies of storage. If one of the storage resources fails, then it can be extracted from another one, this makes cloud computing reliable.

2.4. Cloud Computing Deployment Models

All cloud computing models should have certain essential characteristics. There are a wide variety of service models offered by the cloud and four deployment models. Currently, there are different providers have gained with the external market. There are different types of deployment models based on requirements and usage.

2.4.1. Private Cloud

A private cloud is belonging to a single organization for its different business units. It is privately managed by an organization for its own use. The organization gets benefit from cloud infrastructures flexibility and user-friendliness while guaranteeing the privacy of sensitive data. A private cloud takes advantage of the public clouds, such as being elastic and service. The infrastructure may exist on or off-premises and maintained by the organization or third parties in the private cloud [1]. We can also list the examples that gives private clouds are eBay and Amazon.

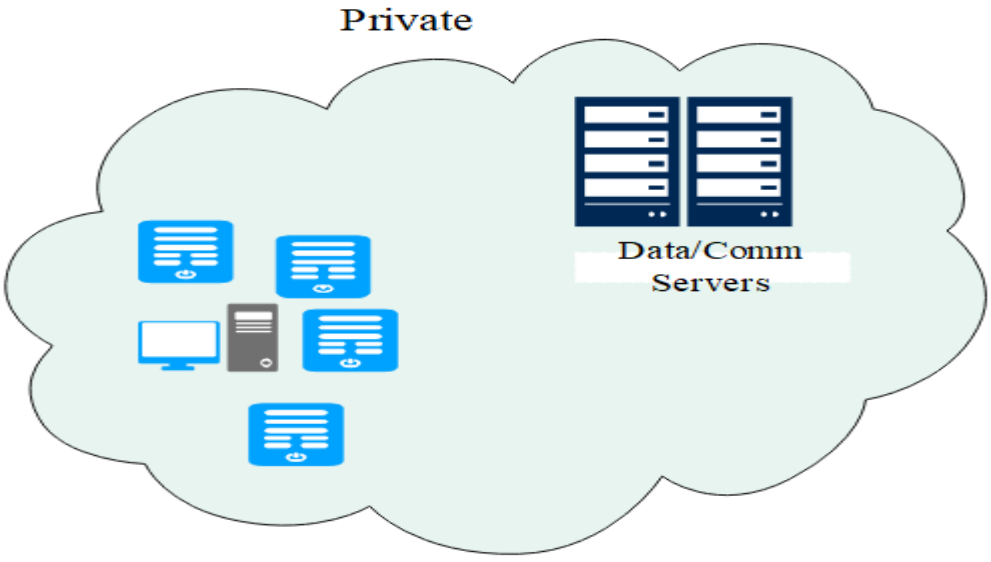

Figure 2. 3 : Private cloud [1].

2.4.2. Public Cloud

A public cloud is a freely available cloud environment possessed by a cloud provider. In the public cloud model, the cloud service provider is responsible to provide the cloud infrastructure and maintain the public cloud for a user's. This means cloud infrastructure is available to the general public or a large industry group and it is owned by organization marketing cloud services. The security and privacy of users' data are managed by the service provider, as the data is hosted on the provider's cloud [1]. Generally, Public cloud provides an elastic, cost-effective means to deploy solutions for users. Examples of public Cloud service includes Microsoft's Windows Azure Platform, Amazon's AWS and Google's AppEngine.

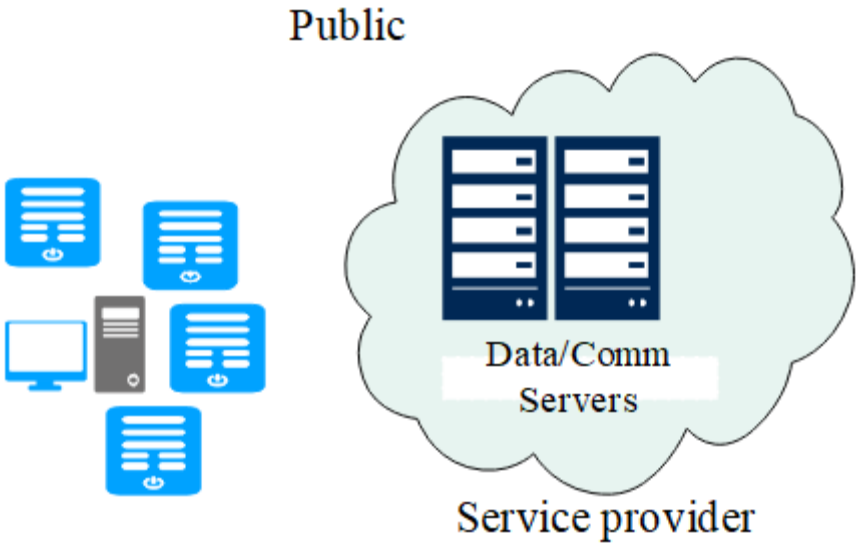

Figure 2. 4 : Public cloud [16].

2.4.3. Hybrid Cloud

Hybrid cloud is a cloud environment comprised of two different cloud deployment models. In the hybrid cloud model, the cloud infrastructure offers an extension for the private cloud to pool local resources with the resources of the public cloud [17].

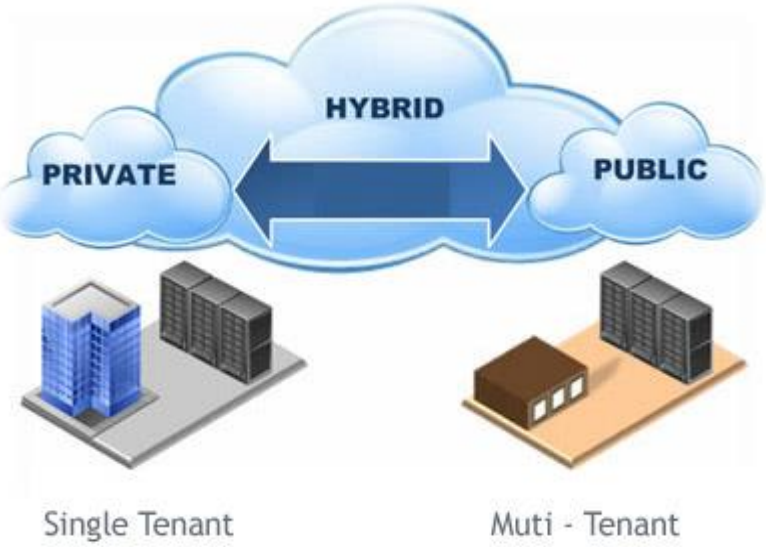

Figure 2. 5: Hybrid cloud [17].

2.4.4. Community Cloud

A community cloud is the same as a public cloud, but the difference is the access limited to a specific community of cloud users in this cloud deployment model. This cloud deployment model is owned either by a third party cloud provider or by the member of the community with limited access. The responsibility for describing and developing the community cloud is handled by the member cloud users of the community.

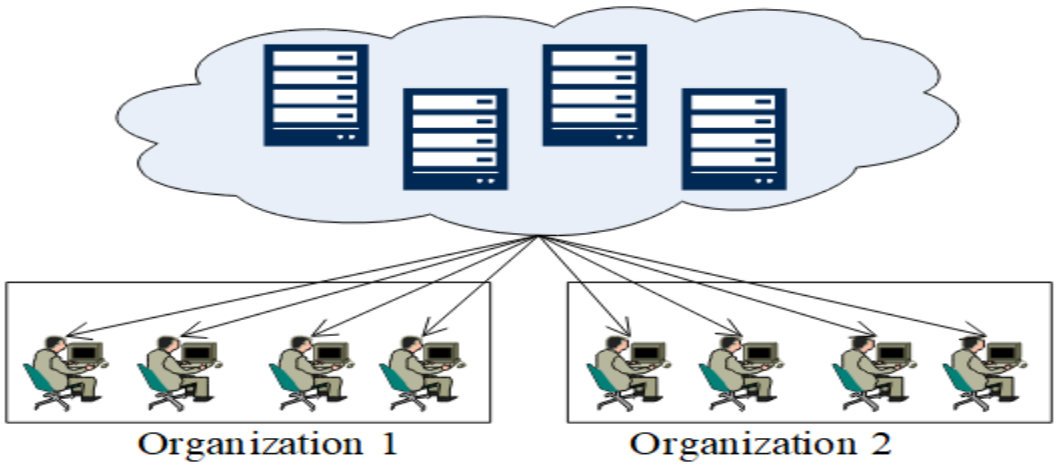

Figure 2. 6: Community cloud [16].

2.5. Cloud Computing Service Models

Delivering the computing resources such as operating system, hardware, software as a service rather than a product is called a cloud computing services.

Depending on the definitions of National Institute of Standards and Technology, the cloud service model is divided into three types. These are:

- Software-as-a-service (SaaS)
- Platform-as-a-service (PaaS) and
- Infrastructure-as-a-Service (IaaS).

2.5.1 Software-as-a-Service (SaaS)

Anybody can use the application through a thin users interface such as web-based e-mail. Controlling the infrastructure, operating system (OS), network, storage and even the service they use and gain are not authorized for the consumers. A user will only be able to configure the application. This model eliminates most of the security threats caused by users. Software as a Service is one of the most common and popular cloud-based service models. Many companies or software providers use the SaaS model to deliver their services to the users. The capability gave to the user is to use the supplier's application running on cloud infrastructure [1]. Salesforce, Google and Microsoft Office 365 are a typical example of Software-as-a-service (SaaS)**.**

2.5.2. Platform-as-a-Service (PaaS)

The ability delivered to the users is to deploy on the cloud infrastructure using programming language, services, libraries, and tools encourage by the provider**.** Users won't have access to the underlying network, operating system, and storage. PaaS supports the cost-efficient deployment of applications without the complexity of buying and managing the underlying hardware and software layers. Even though the consumers have no control over the infrastructure, they have full control of the deployed application. Amazon SimpleDB/S3, Microsoft Azure and Google App are examples of Infrastructure-as-a-Service (PaaS) [1].

2.5.3 Infrastructure-as-a-service (IaaS)

The broadly useful of an IaaS environment is to give cloud consumers with a high level of responsibility and control over its utilization and configuration. Infrastructure-as-a-Service (IaaS) specifies to offerings of necessary computer resource Infrastructure-as-a-Service (IaaS) providers that are full control of virtual resources. IaaS gives the user control over the whole infrastructure such as operating system, network, storage etc. GoGrid, Amazon EC2, and FlexiScale is an example of IaaS [1]. The below figure [17] shows the layered architecture of IaaS, PaaS, SaaS, and applications.

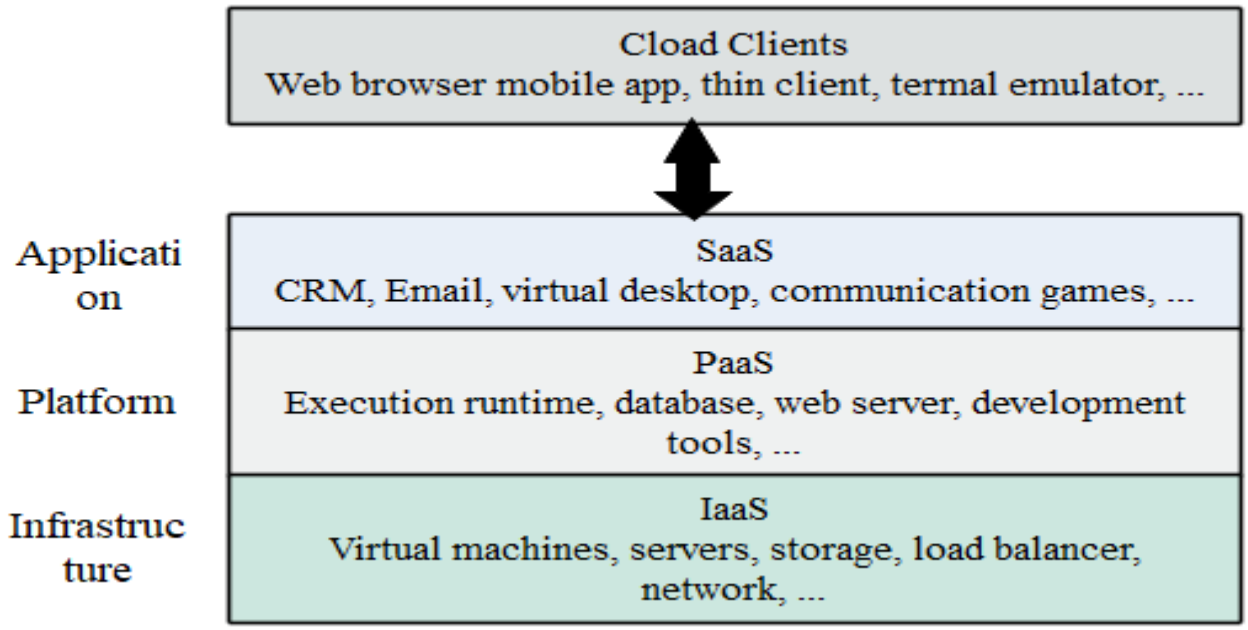

Figure 2. 7: layered architecture of IaaS, PaaS, SaaS, and applications [17].

Cloud Applications: This layer includes applications that are directly available to end-users. We define end-users as the active entity that utilizes the SaaS applications over the Internet. These applications may be supplied by the Cloud provider (SaaS providers) and accessed by end-users either via a subscription model or a pay-per-use basis. Alternatively, in this layer, users deploy their own applications. In the former case, there are applications such as Salesforce.com that supply business process models on clouds (namely, customer relationship management software)

and social networks. In the latter, there are e-Science and e-Research applications, and Content-delivery networks.

User-Level Middleware: This layer includes the software frameworks such as Web 2.0 Interfaces (Ajax, IBM Workplace) that help developers in creating rich, cost-effective user-interfaces for browser-based applications. The layer also provides those programming environments and composition tools that ease the creation, deployment, and execution of applications in clouds. Finally, in this layer, several frameworks that support multi-layer applications development, such as Spring and Hibernate, can be deployed to support applications running in the upper level.

***Core Middleware*:** This layer implements the platform level services that provide a runtime environment for hosting and managing User-Level application services. Core services at this layer include Dynamic SLA Management, Accounting, Billing, Execution monitoring and management, and Pricing (are all the services to be capitalized?). The well-known examples of services operating at this layer are Amazon EC2, Google App Engine, and Aneka. Critical functionalities that need to be realized at this layer include messaging, service discovery, and load balancing. These functionalities are usually implemented by Cloud providers and offered to application developers at an additional premium. For instance, Amazon offers a load balancer and a monitoring service (Cloud watch) for the Amazon EC2 developers/consumers. Similarly, developers building applications on Microsoft Azure clouds can use the .NET Service Bus for implementing message passing mechanism.

System Level: The computing power in Cloud environments is supplied by a collection of data centers that are typically installed with hundreds to thousands of hosts [12]. At the System Level layer there exist massive physical resources (storage servers and application servers) that power the data centers. These servers are transparently managed by the higher level virtualization [10] services and toolkits that allow sharing of their capacity among virtual instances of servers.

2.6. Essential Characteristics of Cloud Computing

A basic condition that a cloud provider must fulfill is the ability to deliver computing resources whenever customers need them. From the customer side, the available computing resources are nearly infinite. This means the customers are not limited to the set of servers located at one site and it is the responsibility of the cloud computing provider o have sufficient resources to satisfy

the requirement of all their customers. The known essential characteristics of the cloud computing environment are discussed below.

2.6.1 Rapid Elasticity

Rapid elasticity is well-defined as the ability to scale resources both up and down as needed. The provision and release of the resource, instantly and elastically, are preferably done in a programmed way in order to follow a consumer to quick scale out and in which considered as rapid elasticity [18].

2.6.2. Measured Service

Measured service refers, cloud provider is checking and pointing all aspects of the cloud service. Usage of resource is controlled and optimized by cloud systems automatically. It has been done by leveraging a metering capacity at some level of abstraction appropriate to the type of service for example, storage, processing, bandwidth and active user account.

2.6.3. On-Demand Self Service

The on-demand and self-service aspect means that a consumer of cloud computing can unilaterally deliver computing abilities such as server time and network storage as needed dynamically without requiring human involvement with each service provider. This enables the customer to avoid making unnecessary upfront investment of servers and computing resources.

2.6.4. Broad Network Access

This means that the cloud provider's skills are accessible over the network and can be accessed through standard mechanism that stimulate use by varied thin-client or thick-client platforms for example, mobile phones, laptops and PDAs.

2.6.5. Resource Pooling

With the resource pooling, the provider's computing resources are shared to serve multiple consumers using a multi-tenant model with dissimilar physical and virtual resources dynamically assigned and reassigned according to user demand.

2.7. Benefits of Cloud Computing

More than the semantic dispute around classification, we believe that in order to maximize the benefits that cloud computing delivers a solution needs to demonstrate these particular

characteristics of the cloud computing. This were discussed under the section of essential characteristics of cloud computing [19].

Cloud computing benefit public sector organizations of all types and sizes by:

2.7.1. Reducing Costs

Know most of users' applications are being moved to CC to use resources wisely by minimized cost. Costs can certainly be reduced. For instance, you don't have to invest in servers at your site, because somebody else is doing that for you. And, of course there is management. User finds only the hardware required for each project. This potentially reduces the risk for the institutions which want build and elastic system, thus providing greater flexibility since the user is only paying for the required infrastructure while reserving the option to increase up services as needed in the future.

2.7.2. Capacity, Scalability and Speed

There are three additional key benefits of the cloud computing depending on its ability to deploy, scalability and speed. A cloud subscriber no longer has to buy more computing capacity than what is needed at any given time since the cloud is scalable. Cloud are designed to distribute as much computing power as any user needs. The cloud resources are projected to simplify the developer's dependence on any specific hardware. Just like the utility of elasticity, the cloud can shrink according to the varying needs of individual subscribers [19]

2.7.3. Availability, Geography and Mobility

Cloud computing is ubiquitous. Cloud technology endeavor access, through the internet to more data stored in the cloud at any time and from any location means, anything at anytime and anywhere.

2.7.4. Backup and Recovery

The procedure of back up and recovering data is simplified since it resided on the cloud and not on physical device. Several providers of cloud offer reliable and flexible backup or recovery solutions. In some cases, the cloud itself is used solely for the sake of backup repository of the data located in local computers. Public cloud computing (CC) is built on a backbone of data security which is combined with geographic redundancy for maximum availability [19].

2.7.5. Increased Storage Capacity

Cloud computing hold and store more data compared to a local computer and in a way it offers almost unlimited storage capacity. It eradicates concerns about running out of storage space and at the same time it spares businesses the need to modify their computer hardware, more to reducing the overall IT costs.

2.8. Virtualization in Cloud

Virtualization is a technology which is different VM is share the same computing hardware machines. VM is mainly enhance resource sharing and utilization with application flexibility. Hardware and software resources are virtualized with different functional layers. This idea is to separate HW and SW for better efficiency. For example, paging in virtual memory was implemented in Manchester university [20]. Similarly, this all resources are (CPU, memory, storage, network channel, etc.) are available within servers through distributed or single data center.

Server may be normal or virtualized using cloud technology. On server there are different load balancing activities when client integrated with sending data. Traditionally servers are maintained by administrator with combine of operating systems hardware storage and different applications. Based on service they serve we named for example serving email server, for serving file operation is called as file server. In traditional server if the storage is reached maximum storage capacity, the administrator responsible to add another server. Today's mostly as the technology is forward to cloud computing the server become more virtualized. These kind of server hide server's software resources away from the hardware on virtual server there are different operating systems storage and applications. Virtual server has different hosts [21]. The administrator is responsible for providing different services accordingly the users request we can make server virtualized either hardware or software to create the virtual entities in cloud computing environment that entities are actually, are not physically present.

Virtualization work with one or many computing resources by combining or dividing resources to end users that are the similar operating environment. There are three different fundamental abstractions which is memory, interpreter and communication link in computing system [22]. The operating system is responsible to manage computer resources for implementing these three

abstractions that is virtualization made simple to their use by isolating users from one another and support replications and scalability for the systems.

Many cloud computing advantages are takeover through virtualization for what serve to providers and consumers that are accessibly, reliability, security, hardware in dependability, isolation, and hiding.

There are different types of virtualization levels [23], these are

- Hardware
- Operating system
- Library support
- Instruction set architecture

2.8.1. Hardware Virtualization

Modern operating systems permit multiple processes to run simultaneously. Processors execute instructions with two different mode user and kernel, privileged instructions are run on kernel mode. Others are run on user mode. Hardware virtualization is done on top of bare hardware by creating a virtual hardware environment for a VM. Virtualization technique helps to map the virtual resources with physical resources.

2.8.2. Operating System Level Virtualization

OS level virtualization is that the virtualized layer, above the OS producers a partition per virtual machines demand which a copy of the operating environment on the physical machine. OS level virtualization creates different containers on single physical server. The containers act as a real server. This kind of virtualizations is mostly used in creating virtual hosting environments to allocate hardware computing resources within different users. The VM share the hardware and operating system on the physical machines.

2.8.3. Library Support Level

Applications written mostly with HLL often call library modules. It also compiled in to object code [24]. Applications are used APIs exported by user-level libraries rather than using lengthy system calls by the OS. User-level library implementations are designed to hide the OS related details to keep it simpler for normal programmers since most systems provide an APIS, such as an

interface becomes another candidate for virtualization. By controlling communications link between applications and rest of systems by virtualization library interfaces through API.

2.8.4. Instruction Set Architecture Level

This virtualization is done at ISA level implemented by emulating in software [23]. In this ISA level approach, it is possible to run binary code written for various processors on any kind of HW machines. Instruction which is issued by guest VM is executed by emulator by translating native instructions and then executing them on the available hardware. Instructions like processor such as, add, sub, div, and I/O are used for the devices. Emulation is done through code interpretation which interprets the source instructions to target instructions one by one. When an emulator works by translating instructions from the guest to the host accomplishing the same task through instructions available on the host, without any stringent.

2.9. Related Work

Many types of research have been done in the area of cloud computing in recent years. But some papers covered some concepts about cloud resource scaling. Some papers also covered some concepts of horizontal and vertical scaling resources of cloud data centers. The gaps of the thesis; their limitation; the methodology used and result got in previous work was revealed in the next paragraphs.

In Chien-Yu Liu, Meng-Ru Shie, Yi-Fang Lee, Yu-Chun Lin, Kuan-Chou Lai [11] an overall perspective on vertical/horizontal resource scaling approach mechanism for federated clouds with the help of experiment was provided. The researcher motivation is a service provider that share software and hardware sometimes not fulfill users' demand. Due to an increase in users' demands to use cloud data centers, the single virtual machine cannot satisfy the users demands. Therefore, their mechanism will allocate new resources in the federated clouds by vertical/horizontal scaling fashions. performance for NAS (Network Attached Storage) parallel benchmarks, for example, Conjugate Gradient (CG), Parallel (EP), Integer Sort (IS) was evaluated.

The paper had two phases. The first phase analyzes the execution pattern of applications and the cost/efficiency of resources in the federated cloud. In the second phase, their proposed approach allocates resources according to the cost/efficiency and the execution pattern of applications for improving system performance. Based on requested load services varies. Therefore, resource auto-

scaling varies the required resources adding and removing the virtual machine by using horizontal elasticity. The result of the study indicates Horizontal/Vertical resource scaling has the advantage to allocate resources to satisfy user's demand by cloud service providers in the federated cloud. When user's demand increase, cloud service provider has to get more resources to avoid SLAs. Their experimental result also shows that horizontal and vertical resource scaling can improve the system throughput.

In Wenting Wang, Haopeng Chen, Xi Chen [12], the authors discussed that availability-aware policy by performing both horizontal and vertical scaling to explore where and how computing resource can be allocated. The aim of scaling of the resource in the cloud computing to provide enough resource to users. In the cloud the elastic mechanism of cloud computing providers requires to scale up or down based on workload intensity. The researchers approve their objectives by simulation results. In this work, the researchers addressed two problems. These are - 1. modeling cloud availability and generality of the cloud; 2. Placement of the resource by dynamic scaling in a cost-efficient and availability-aware way integrated with horizontal and vertical scaling principles. This availability improvement mechanism using these two scaling are maximized when scaling up and when an application needs to be resized down, the existing availability maintained. The proposed system is provided only for single applications in the cloud. The obtained result validates the result.

In Stella Gatziu Grivas, Marc Schaaf [13], proposed two approaches to scale up cloud environments. These two concepts are scale cloud environments based on the request and on demand. The researcher's motivation is cloud service providers sometimes over-provisioned (having too many resources), under-utilized (not using resources effectively and under-provisioned (having too little resources) have an influence on service providers as well as on users. The other motivation of the researchers is in no-cloud (traditional) environments under-utilization and under-provisioning was avoided hardly. This problem was leads to cloud service providers not utilized as much as possible and cloud users not use as much as they need. Cloud computing resources are utilized hourly, daily, weekly and monthly by variable workload and well suited for applications. The contribution of the paper was to present the result of experiment on automatic scaling resource mechanism of AWS elastic beanstalk. The experimental result showed that the

cloud application can automatically scale up and scale down when the load increases and decreases respectively and resource provisioning took few minutes.

In Tao Li, Jingyu Wang, Wei Li, Tong Xu, Qi Qi [25], it is necessary to cloud computing that control elasticity and dynamically adjust resources on demand by increasing or decreasing the number or size of virtual machine. In this paper the authors tried to develop algorithmic based auto-scaling to control automatic scaling of resources by changing the no of virtual machine or size based on load prediction. With the fast advancement of Information Technology what's more, the internet and the quick increasing in mass information processing, Cloud computing became as sharing resource model. Virtual machine, for example, VMware and Xen guarantee that the cloud computing allocates resources elastically and flexibly with low cost. In cloud environment automatic scaling difficulty has two main issues. One is the point at which the scaling begin, on the other hand is the means by which to scale. The researchers also gave solution for such problem by load prediction approach. The experiment result shows their proposed system.

In M. Kriushanth, L. Arockiam and G. Justy Mirobi [14], the authors proposed cloud computing overview and accentuate the auto scaling. Cloud computing is current technology to give resources from large data centers and it is available as a service to its users. It has become important IT to begin a business or to use the resources without any capital venture. To get services from cloud users charge cloud services and service provider also charge the users by using on demand or request service approach. Therefore, in order to give the good services, service providers have to enhance the scalability factor. The researchers presented concept of scalability, scalability types, auto scaling and features of auto scaling.

In Nilabja Roy, Abhoshek Dubey and Anirudda Gokhale [26], the authors suggested different approaches and mechanisms to overcome the general lack of powerful techniques to workload predicting and best resource allocation. In spite of perceived advantages of auto-scaling, understanding the maximum capacity of auto scaling is hard because of various difficulties coming from the need to exactly estimate resource use despite critical fluctuation in customer workload arrangements. This helps researchers to makes three contributions for the research. These contributions are: - first, it talks about the difficulties associated with auto scaling in the cloud. Second, it develops algorithm for workload estimating that is used in auto scaling resource. In addition, the researchers also evaluating their approach and shows its advantages for both cloud

providers and users. The limitation of the paper is their approach is in the context of minimum number of machine used.

In Wathit Chaloemwat and Sukumal Kitisin [27], an overall resource scaling in cloud computing and highlight with the help of simulation was provided. The contribution of the paper differentiating the advantage of using algorithm in cloud auto scaling. The researchers propose horizontal auto scaling with process relocation (migration) systems for cloud services with algorithm. Also the researchers implemented the proposed system on java based simulation and tested it in two unique situations. These scenarios are: - tested without algorithm and migration of process. Second, one tested with the skewness algorithm and migration of process. The analyzed result could be tested using cloud simulator. The results of the simulation show that the process migration and skewness algorithm can help smooth out the variance of the quantity of virtual machines deleted and spawned which can help reduce the overhead.

In Zhen Xiao, Senior Member, IEEE, Weijia Song, Qi Chen [28], The authors are presented the system that use virtualization technology to allocate datacenter resources dynamically within the applications demand. They are used skewness and virtualization techniques to support Green computing. The main aim of this work was that the capacity of PM should be satisfied for all virtual machines resources need running on it, the physical machines number should be decreased as all need virtual machines satisfied, in Green computing the energy is decreased by turned off the idle machines. To measure the skewness, they were used based on hot spot and cold spot. Hot spot is used if any resource utilization is above hot threshold directing the server is overloaded and the virtual machine migrate to away. Cold spot resource utilization is below the threshold directing the virtual machines are idle and dynamically the power of the servers turned off. They mainly work for horizontal scaling method to energy saving and scale up and scale down with virtual machines. We are listed the drawback of the paper by the table.

In Jiayin Li, Meikang Qiu, Jian-Wei Niu, Yu Chen, Zhong Ming [29], the authors presented adaptive resource allocation algorithm for cloud system by pre-emptible tasks. The proposed algorithm for cloud systems was used to adjust resources based on updating the actual task executions. They were used to algorithms to scheduling the tasks; these are, Adaptive list scheduling (ALS) and Adaptive min-min scheduling (AMMS) algorithms. The paper also presented static task scheduling for static resource allocation that generated offline. These scheme

also use horizontal scaling method for each task to assign to virtual machines. In this study it gives priority for the task already run in. but not for new task became when it adapts the execution environment it assigns the resources to the already running task.

Table 2. 1: Related papers.

Reference	Objective	Pros	Limitations
[30]	➢ Developing elastic scaling using vertical resource scaling mechanism.	➢ Plugin CPU dynamically not statically, improve CPU usage efficiency.	➢ The controller scale only one resource that is only CPU, used only vertical scaling that have limited power of CPU. ➢ Priorities VM machine is assigned by provider without know about customer. ➢ When more resources are required by non-priorities VMs there no way were they take.
[31]	➢ Developing hybrid auto scaling.	➢ Develop hybrid auto scaling to handle burst workload.	➢ The developed mechanism is 1stly perform horizontal scaling that is traditional based scaling. ➢ They are not stated any way how to handle, if workload bursts are greater than the capacity of the single machine (vertical scaling).
[28]	➢ Presenting the system that use virtualization technology to	➢ They saved energy by turning off the	➢ This method is mainly used horizontal scaling method to save energy.

	allocate data center resources dynamically within the applications demand.	idle machines using cold spot resource utilization method.	
[29]	➢ Adaptive resource allocation algorithm for cloud system by pre-emptible tasks.		➢ These paper also used horizontal scaling method for each task to assign to virtual machines.

CHAPTER THREE

3. Research Methodology

3.1. Overview

In this section, the flow of the research methodology; examining the current system in order to get deep understanding of our development method, defining the design of the proposed system method, describing proposed system testing method, and compare and contrast available development tools in order to select appropriate tools for the proposed system were described.

3.2. Cloud Resource Scaling Methods

Currently, two types of cloud resource scaling techniques were introduced to scale resources to satisfy users demand. Some researchers [30] [28] have been done developing cloud resource scaling techniques in different ways. They are performed by two different mechanisms. These mechanisms are vertical scaling and horizontal scaling mechanisms.

3.3. Development Tools

Development tools discuss the tools that provided to develop the proposed approach. There are different types of tools used for this research to design and implement the proposed system. These are software, UML modeling, and other relevant tools. These are described below:

3.4. Design Tool

Edraw Max tool is a type of design tool which enables the software designer to reliably create and publish many kinds of diagrams to represent an idea. This design tool provides many features such as flowcharts, workflows, software, UML diagrams, Business diagrams, database and ERD and no other diagram software gives you all these features [24].

3.5. Implementation Method

Cloud Scaling mechanisms are implemented by different development software based on their platform. Currently, CloudSim is the type of open source development software that is used to develop cloud scaling systems. According to [32], CloudSim has various features including modeling and simulating large scale data centers, modeling and simulating virtualized server hosts, modeling and simulating energy-aware computational resources, modeling and simulating federated clouds, inserting simulation elements dynamically, and user defined policies for

allocation of hosts to virtual machines. Hence, CloudSim is selected to implement the proposed system.

3.6. Data Collection Method

Websites and literature reviews are the main sources of data for this research in order to collect relevant user friendly design matrices. Possible information's using the recent information from the websites and CloudSim based user scalable design guidelines were selected [33].

3.7. Proposed System Design Method

The system design has been designed for analyze the input based on the designed system and finally give the output. Specifically, the designed system is to evaluate elements based on the given guidelines; identify detected upper thresholds, and give some critiques regarding to that.

3.8. Proposed System Implementation Method

The Hybrid scaling resources for dynamic workloads on cloud environment based on proposed solution architecture designed that provided in the proposed solution section. The designed system has been implemented using CloudSim integrated with Eclipse.

3.9. Proposed System Evaluation

This proposed system usability evaluation is run on Eclipse integrated with CloudSim. The testing can be performed by developer during development. First, the developer should add some input (workload) and run the code. Finally, the system generates the result (output).

After the researcher has applied the above mentioned steps, the performance of the expected result was evaluated. In fact, the evaluation is important and mandatory to analysis our result very well. In this study the researcher test and evaluated the performance of the developed system.

CHAPTER FOUR

4. Designing Proposed Solution

4.4. Proposed Solution

This chapter proposed an architectural solution for Dynamic workload using hybrid resource scaling approach. Research questions were defined to solve the problem mentioned inside the statement of problem in this work. System architecture is proposed based on the literature discussed previously in related works in order to achieve this work.

4.4.1. Proposed Solution Architecture

Hybrid resource scaling approach that perform both vertical and horizontal resource scaling have been proposed. Firstly, the users demand can be performed on vertical resource scaling and as demand increases, if the resource in vertical is not sufficient horizontal resource scaling will be performed.

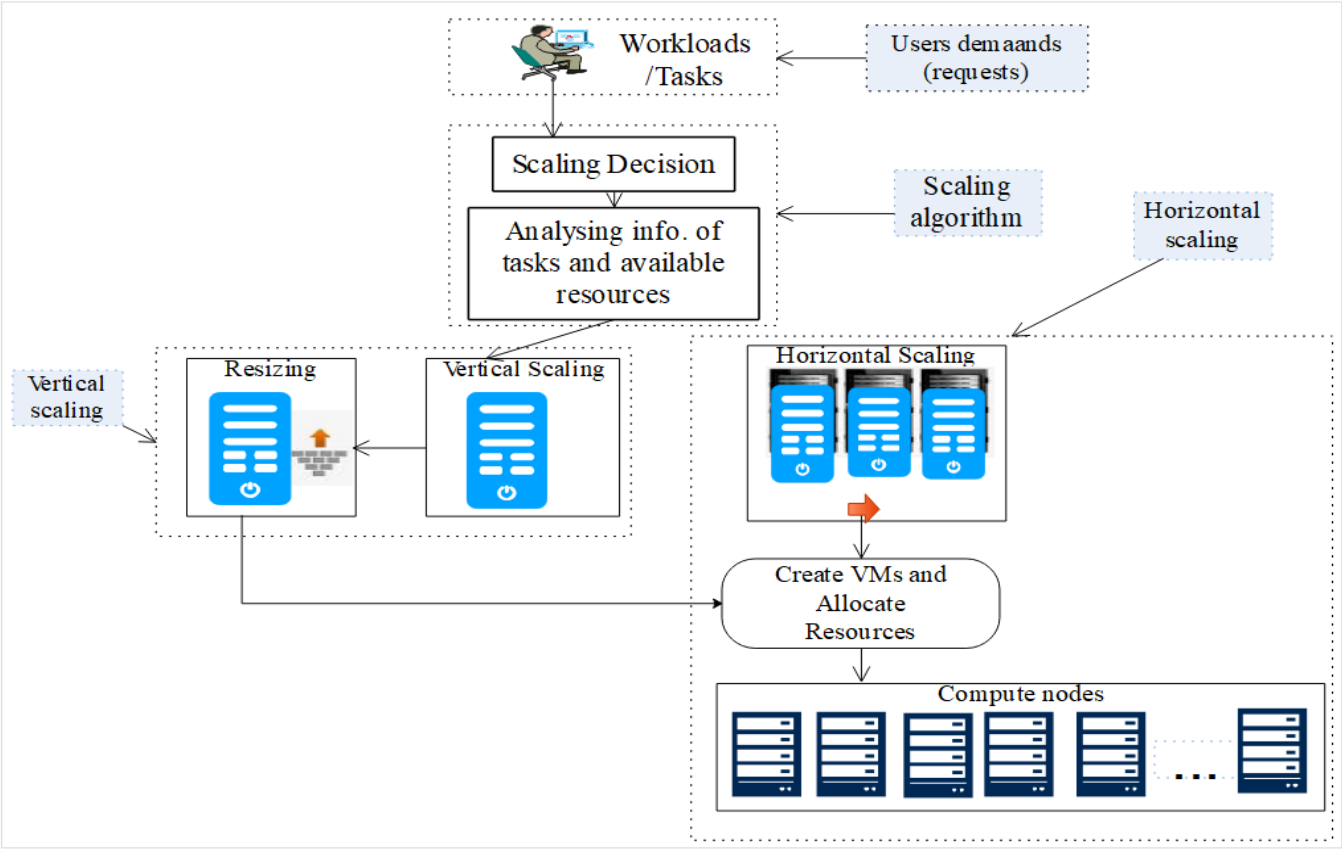

Figure 4. 1: Overall Architecture of proposed solution.

Workload: In this work we took workload as cloudlet or a cloud task.

Scaling Algorithm: This entity analyzes resources allocated to VMs and initiates the appropriate scaling to satisfy users demand. If additional resources required for VM are available in the host server itself then the vertical scaling is initiated. If there are insufficient free resources in the host server, it resizes itself up to its size and allocate. If the required resources are greater than the capacity, then horizontal scaling initiated.

4.4.2. Pseudo Code of Proposed Approach

This section describes the pseudo code algorithm of proposed solution.

Resource scaling up algorithm

Input ← Workload or User_demand (Resource_to_be_allocated)

Output ← Resource_allocated (CPU, RAM)

Begin

for i← 0 to n do

Workload or user_demend_to_be_allocated

If single_machine (resource_available && resource_available > user_demand) ← true

//Vertical scaling

User_demand(allocate_resource)

If single machine (host_has_capacity_to_scale_up) ← true

Resize_up_to_be_allocated (CPU, RAM)

Else

//Horizontal Scaling

Create_extra_machines_to_be_allocated (allocate_resources)

End if

4.4.3. Hybrid Resource Scaling Pseudo-Code Perform Based on the following Operations:

Input (Workload): this describes the user demand to be allocated resource for their applications and etc. since the implementation of the proposed system was not on real cloud, cloudlets were taken as input for our system.

Single Machine: after users send the demand to be allocated resource based on the algorithm, if there is enough resource to be allocate users demand before resizing the single host users can allocate its demand resources. But, if resource available in the single host is not enough to be allocated users demand, resizing the host as much as its capacity to allocate the resources to the users. This *technique is called performing vertical scaling method*.

Extra Machines: this describes, when the user's demands are greater than resource available in the single host and also, greater than the resized host it creates another machines based on users' demand and allocate resources to users.

Scaling Metrics: the systems capture these key metrics, such as CPU utilization, memory usage, and queue lengths. Decision making logics evaluates these predefined thresholds, and decides whether to scale up or scale down.

4.4.4. Hybrid Allocating Resources Flow Chart

The flow chart of allocating a resources using hybrid method diagram with respect to availability, responsiveness and etc. were discussed in diagram shown below.

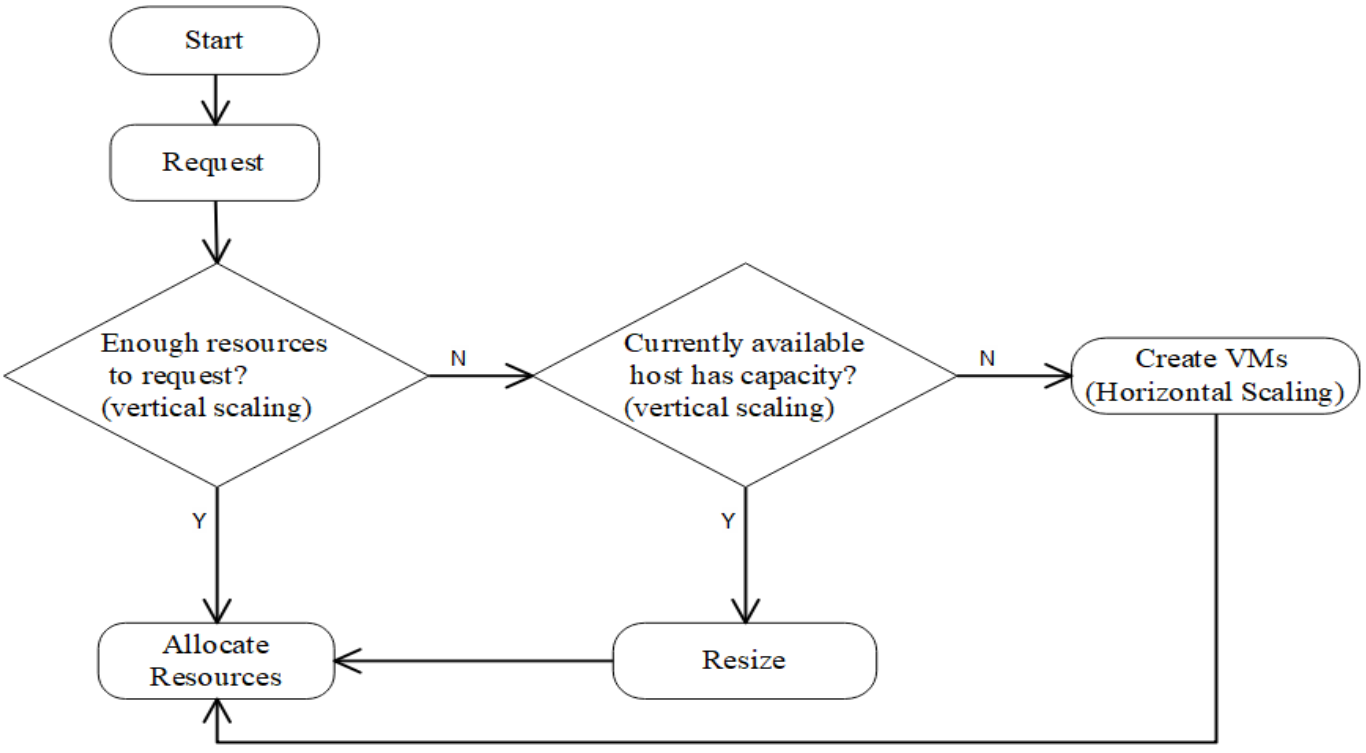

Figure 4. 2: Hybrid Allocating Resources flow chart.

Currently Available Host has Capacity: **-** these describes, if the currently available host has capacity to scale up the resource will scale up.

Currently Available Host has not Capacity: - these describes, if the VM currently available has not capacity or reached its upper thresholds, scale up host (horizontal scaling) that is available in the datacenter.

4.4.5. Hybrid Scaling Down/Releasing Resources

Steps to scale down resources are described below.

Step1: User send request to scale down/ release.

Step2: If resource allocated is in vertical scaling subdivision, scale down/release resources.

Step3: If resource allocated is in horizontal scaling, scale down/release resources.

Step5: Update the resources.

Resource scale down Algorithm

Input: User_request (to_be_scaleDown/release_resources)

Output: resource_scaleDown/release ()

Begin:

User_request (to_be_ScaleDown/release)

If (User_request < Currently_AllocatedResources) ← true

//Vertical scaling

If (Single_Machine) ← true

ScaleDown/release (AllocatedResources)

Else

//Horizontal scaling

ScaleDown/release (AllocatedResources)

End if

Flow chart of hybrid resource scaling down / releasing is shown below.

UR- *User Request.*

AR- *Allocated Resources*

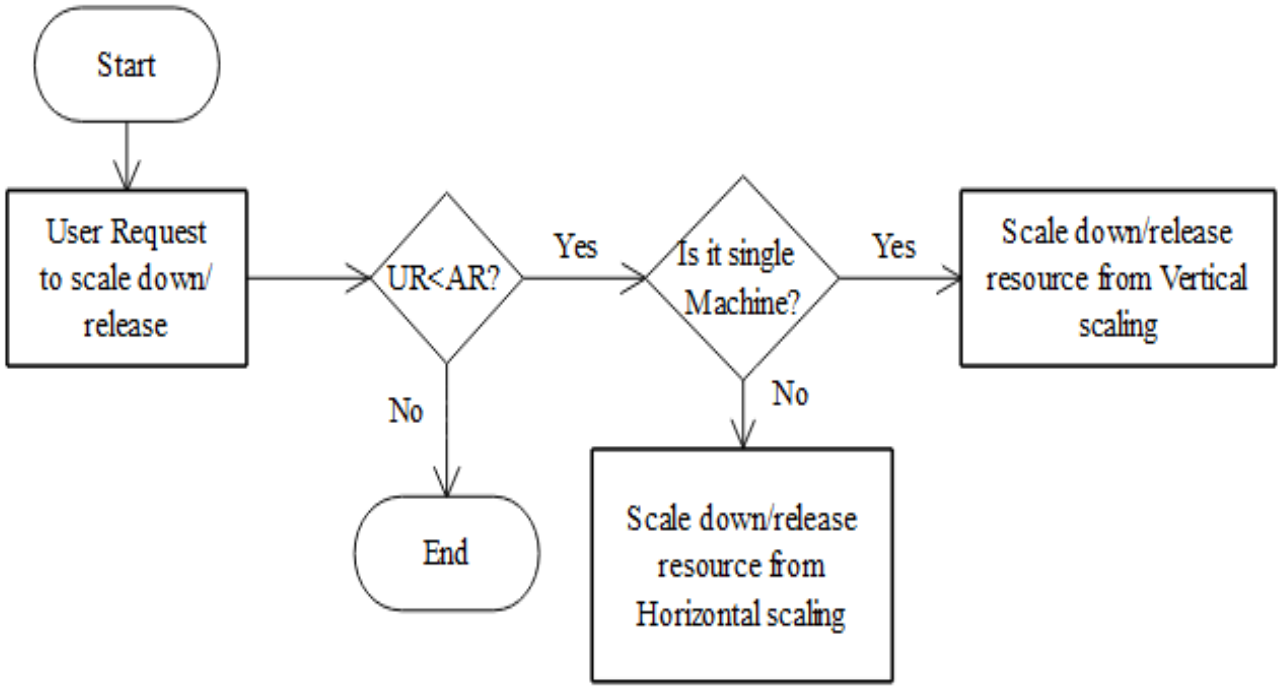

Figure 4. 3: Hybrid Resource Scaling down flow chart

4.4.6. Hybrid Scaling follows Path.

VM that has a Vertical VM Scaling Object set monitors its own resource usage using an Event listener to check if an under loaded (upper threshold) or over loaded (lower threshold) condition is met; If any of these conditions is met, the VM uses the vertical VM scaling to send a scaling request to its Datacenter broker; the Datacenter broker forwards the requests to the Data center where the VM is hosted; then, the Datacenter delegates the task to its VM allocation policy; next, the VM allocation policy checks if there is resource availability and scale the VM. After checking if there a resource availability, if there is no resource it's going to perform Horizontal scaling.

Since cloudlets (workloads) are created and submitted to the broker in run time, the number of arrived cloudlets can be too much to existing VM, it required the creation of new VMs to distribute load. Horizontal scaling implementation performs scaling up by creating compute nodes as needed.

Detail description of scaling up and scaling down:

Scaling up: this is performed if users demand is increased.

Scaling down: this is performed if users demand becomes decreased. For example:

Let we take 'n' as allocated resources for the users and,

'm' as users' demand.

If n > m scale down.

4.5. Mathematical Equation of Virtual Machine Cost

The mathematical model of the total cost of virtual machine created in the case of horizontal scaling when an extra virtual machines is created to satisfy users demand was developed. In this mathematical modeling, each virtual machine may have considered as they have their own cost or the cost of each virtual machine is different. So, to get the total number of virtual machine cost the cost of each virtual machines should have to added.

Before calculating the total cost of virtual machine, cost of single virtual machine is calculated. Because the total cost of extra virtual machine created is the summation of cost of each virtual machine. Therefore: -

Cost of each virtual machine can be calculated by:

$CVMn = \sum_{m=1}^{no.R} Rm = (R1 + R2 + R3 + \cdots + Rm)$ ----------(1)

Where:

- n is virtual machine created and n = 1,2,3, 4,…, n
- m is single resource
- CVMn is cost of single virtual machine created
- no.R is number of Resource available in single machine
- R is resource

After the calculation of cost of each virtual machine, the total cost of virtual machine created in case of horizontal scaling was calculated.

Total cost of each virtual machine can be calculated by:

$$\text{TCVMcreated} = \sum_{n=1}^{no.VMcreated} \text{CVMn} = (\text{CVM1} + \text{CVM2} + \text{CVM3} + \cdots + \text{CVMn}) \text{ ------(2)}$$

Where:

- *TCVMcreated*- total cost of virtual machine created
- *n*- virtual machine
- *CVMn*- cost of single virtual machine
- *no.VMcreated*- number of virtual machine created.

The proposed method is performed by taking the advantage of both vertical and horizontal scaling method to provide an effective resource scaling. Both scaling methods and the architectures are separately discussed in section 4.6 and 4.7.

4.6. Vertical Scaling

Vertical scaling is one of the scaling resource mechanism, that perform its work if the current capacity of resources like RAM and CPU available in a single machine is not suitable for work loaded, it can resize itself depending on the capacity of the machine. It also called method of adding or removing some resources to o from virtual machine.

4.7. Horizontal Scaling

Horizontal scaling also one the scaling resource mechanism that can be done by adding VM or compute node to satisfy users request.

In order to scale resources up or down, the controller considers two resource usage upper threshold and down threshold respectively. In the first case, the controller controls resource usage below already defined upper threshold and allocate additional resources to a VM based on its capacity, when the workloads exceed the upper threshold. The second one is used to avoid where the resource is under-utilized.

4.8. Modeling Resource Allocation Process

There are three components to model allocating resources for users. The first model definition is the model component which undertakes activities like get input. The second model component is

a model deciding in which scaling decision algorithms are applied to decide to perform either vertical scaling or horizontal scaling to allocate appropriate resources to the users. The third model component is a model that evaluate the allocated resources to the users.

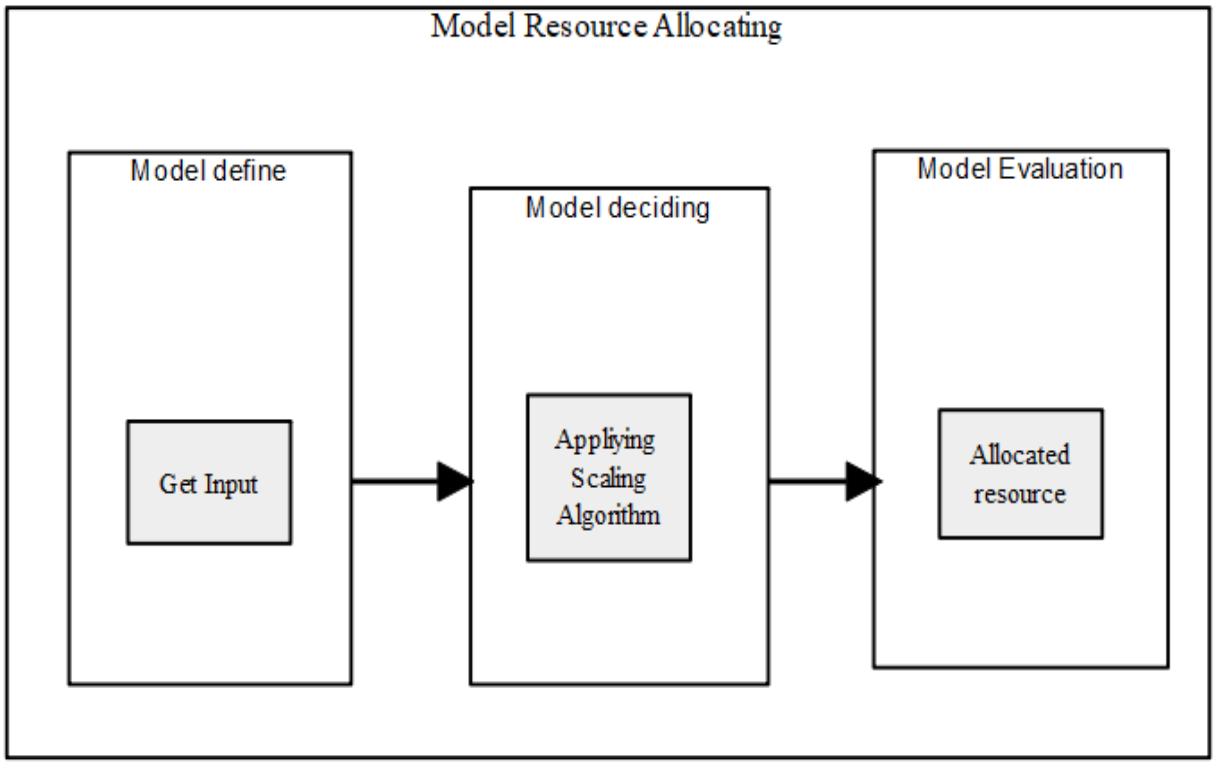

Figure 4. 4: Model of resource allocation.

CHAPTER FIVE

5. Experimental setup and Implementation

5.1 Experimental Setup

This section discusses the experimental environment used to conduct this research. Conducting the experimental research for this proposed approach or for cloud computing using real world infrastructure is time-consuming due to their complexity. Because of this, CloudSim was used. In this section we discussed the reason why cloudsim was selected as the best and popular cloud simulation tools and we compared the simulation tools to choose the best one. Also, the hardware and software requirement used in this study were discussed.

5.1.1 Software Requirements

The software used to conduct this research is below.

- Operating System: - Microsoft windows 10
- IDE: - Eclipse Oxygen
- Programming Language: - Java
- Simulation Tool: - CloudSim

5.1.2 Hardware Requirements

- The hardware requirement we used to conduct this research is listed below.
- Processer: Intel(R) Core(TM) i3-3120M CPU @ 2.50GHz 2.50GHz.
- RAM: Installed memory(RAM) 4.00GB.
- System Type: x64 based processor.
- Hard Disk: 640 Gb.

5.2. Simulation Tool

There are a lot of tools available for the simulation in cloud environment and the most popular simulation were tried to described. To select the best fit simulation, we have gone through some identified simulation tools. We compared these simulation tools. But due to computability, flexibility and poor feature factor, we rejected and select CloudSim for the simulation.

5.2.1. Comparison of Simulation Tool

5.2.1.1 CloudSim

CloudSim [34] provides a generalized and extensible simulation framework that enables seamless modelling and simulation of application performance and it is toolkit which is developed for the simulation of Cloud Computing environment in clouds Laboratory at CSE Department of Melbourne University, Australia. It supports the modeling and simulation of various cloud computing components, including power management, performance, data centers, computing nodes, resource provisioning and virtual machine provisioning. CloudSim admits simulation of service modeling PaaS, SaaS and IaaS. The architecture of cloudSim is shown below.

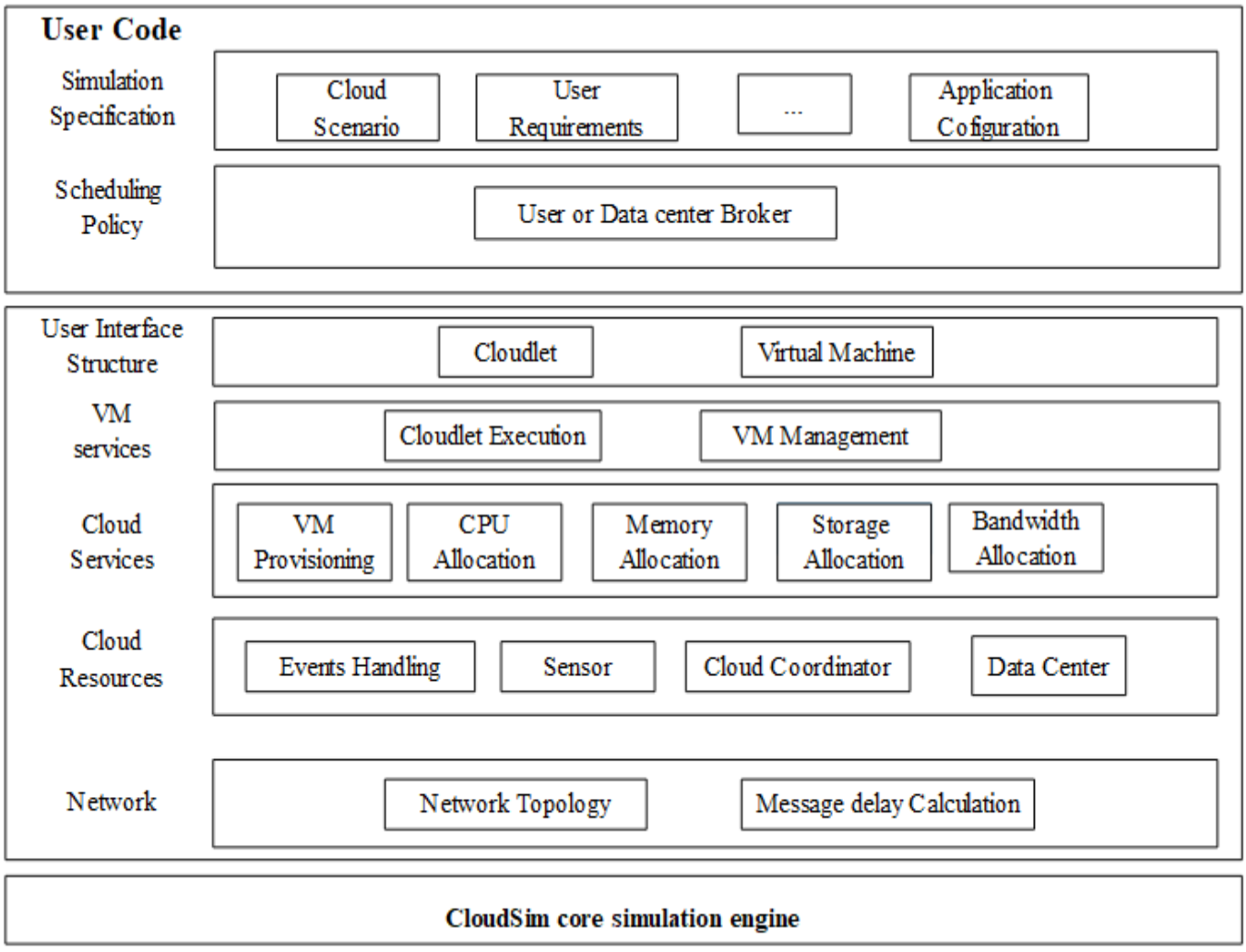

Figure 5. 1: Architecture of CloudSim [35].

5.2.1.2. CloudReport

CloudReport [36] is another tool that built upon CloudSim framework, but it improves many features of CloudSim such as user-friendly graphical user interface, running multiple simulation at the same time, and display simulation result in chart and table. It provides different features for the researcher to play the role or service providers and users. It also provides information about VM allocation, energy consumption, and user defined characteristics. It allows service providers to evaluate their cloud environment before leasing the services to users. It enables the user to set the number of virtual machines each user owns, broker responsible for allocating these virtual machines and resource consumption algorithms, make the number of computational nodes and their resource configuration, which comprises processing capacity, the amount of RAM, available bandwidth, resource utilization and execution time. The architecture of CloudReport is shown below.

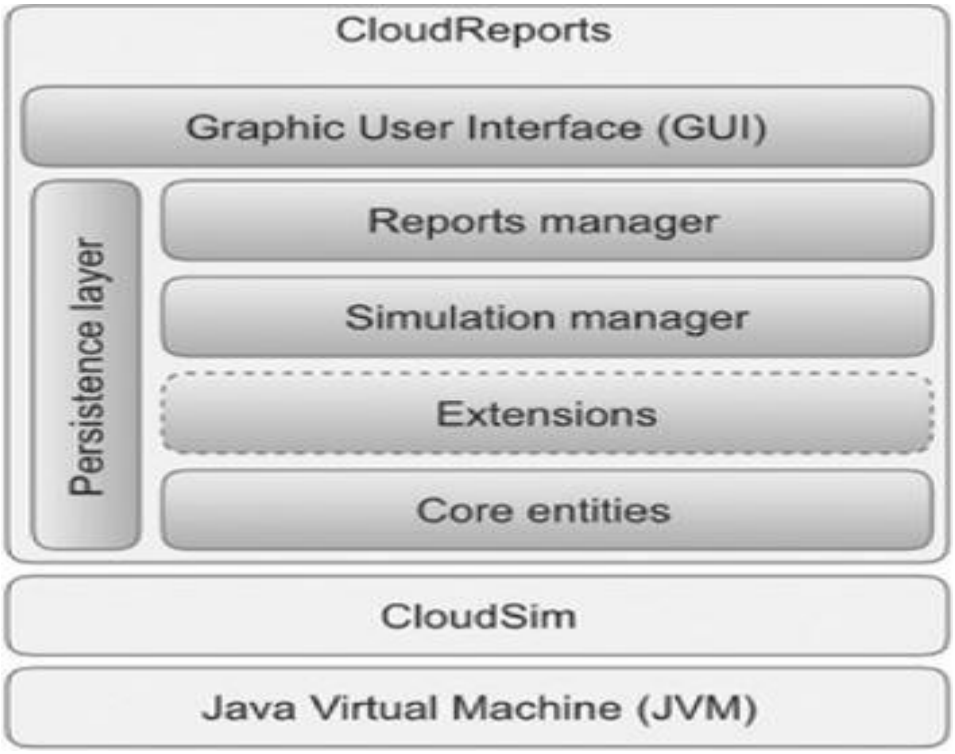

Figure 5. 2: Architecture of CloudReport [36]

5.2.1.3. CloudAnalyst

CloudAnalyst [36] is graphical user interface simulation tool which is built on top of CloudSim toolkit. It is developed at the GRIDS laboratory at the Melbourne University, Australia and designed for some specific goals. The simulator can be applied to examining the behavior of large scaled internet application in a cloud environment. It separates the simulation experimentation exercise from a programming exercise. This simulation supports the evaluation of social network tools, based on their graphical distribution of users and data centers. I addition to this, it also configures all parameters of simulation at a high level performance and accuracy which displays the result in the form of tables and charts. The main features of this simulation tool is easy to use, graphical user interface, repeatability of experiments, graphical output and easy to extension. The architecture of CloudAnalyst is shown below.

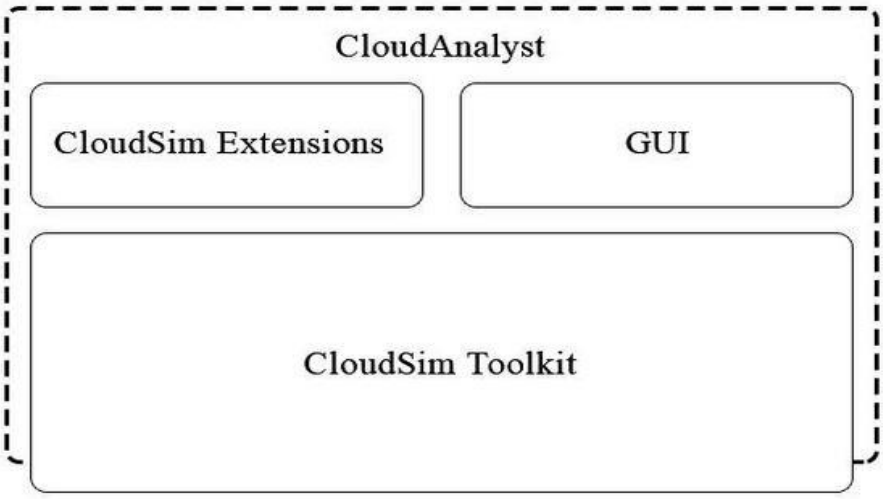

Figure 5. 3: Architecture of CloudAnalyst [35].

5.2.1.4. GreenCloud

GreenCluod [34] is improvement of CloudSim and considered as an extension of the network simulator Ns2. It is designed to prove the green cloud computing. The interest with GreenCloud development to interact and measure cloud performance is the lack of detailed simulators, and having no provisioning system for analyzing the energy efficiency of the clouds. It provides a detailed fine-grained modeling of the energy consumed by the data center IT equipment such as network, computing servers, and communication links. It is used to develop novel solutions in monitoring, resource allocation, workload scheduling as well as optimization of communication protocols and network infrastructures. The architecture of GreenCloud is sown below.

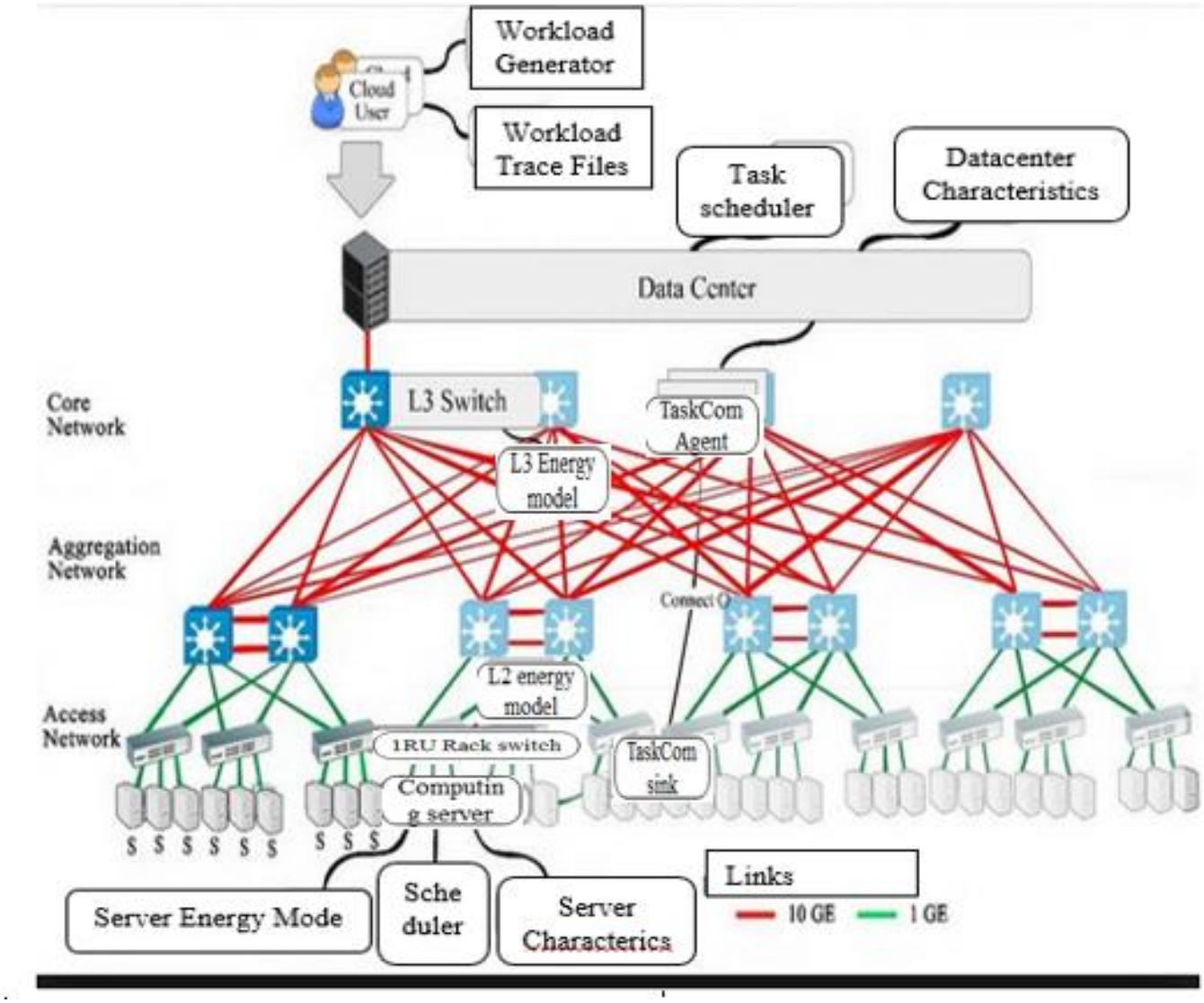

Figure 5. 4: Architecture of GreenCloud [34].

5.2.1.5. CloudExp

CloudExp [36] is modeling and simulation environment which introduced a specialized mobile cloud computing experimental framework and integrates several new features and important components to cloudSim. This simulator allows researchers to study the communication cost between customers and clouds. In this simulator, cluodlets are created using simple drag and drop. In addition to this, it conducts various mobility scenarios for mobile devices. In CloudExp, the users need to configure the workplace properties and then decide the number of users, a number of cloudlet in that region, type of cloudlet (fixed or mobile), and other properties of cloudlet. When

the simulation is complete, ClouExp results are saved as Excel sheets to simplify the analysis of the result.

5.2.1.6. iCanCloud

iCanCloud [37] is a simulator tool capable of conducting large-scale experiments. This simulator is developed over SIMCAN that is used to analyze high performance I/O architectures and uses the popular OMNET++ framework for simulating computer networks. It also provides a flexible, scalable, and easy to use tools. Virtual machine is the building block of the cloud computing system in this simulator. In this simulator VM can be built by depending on its four main components and configure them. These components are: CPU, memory, storage, and network. It also provides, graphical user interface for its users to configuring their simulations. The architecture of iCanCloud is shown below.

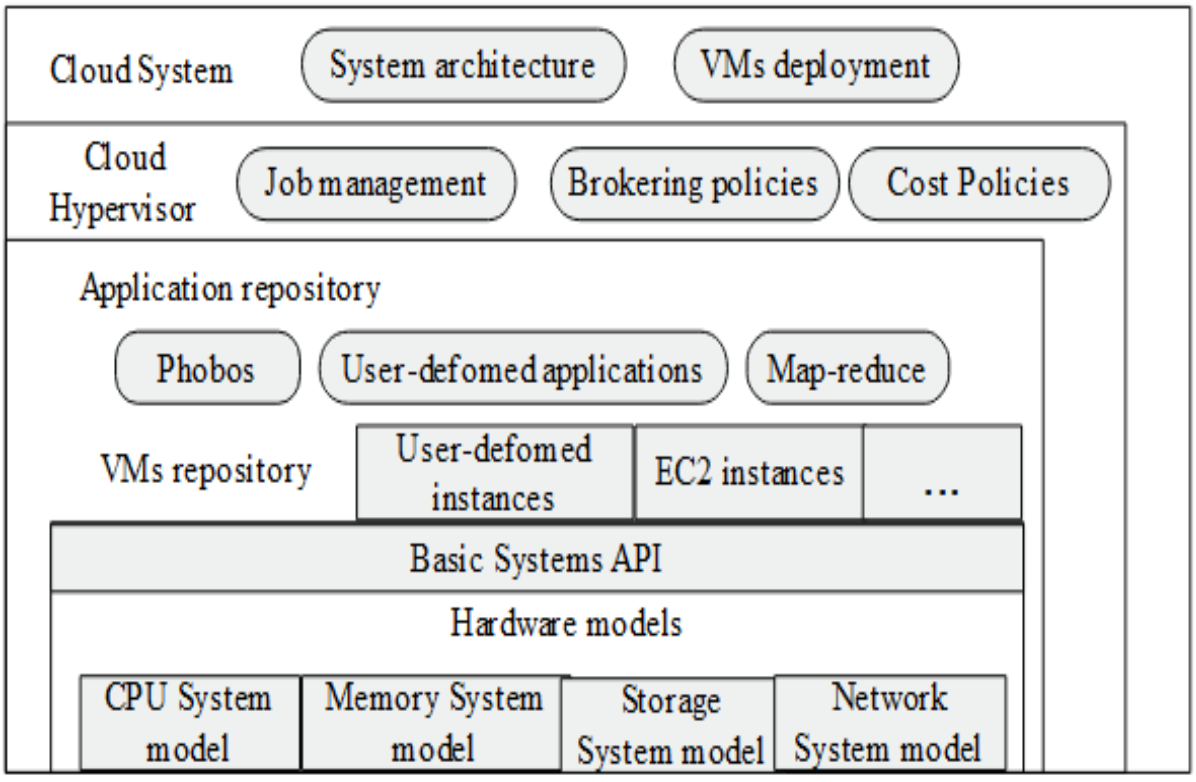

Figure 5. 5: Architecture of iCanCloud [34].

5.2.1.7. NetworkCloudSim

This simulator is proposed by S.K. Garg and Rajkumari Buyya [38], extension of CloudSim. It uses network topology class which implements network layer in cloudSim, reads a BRITE file and generates a topological network. It allows for modeling of cloud data centers utilizing bandwidth sharing and latencies to enable scalable and fast simulations. Its structure supports designing of the real cloud data centers and mapping different strategies.

5.2.1.8. EMUSIM

EMUSIM [32] is built on the top of Automated Emulation Framework (AEF) for emulation and CloudSim for the simulation to anticipate services behavior on cloud environment to a higher standard. It merges simulation and emulation to extract information automatically from application behavior via emulation and uses this information to generate the corresponding simulation tool. The architecture of EMUSIM is shown below.

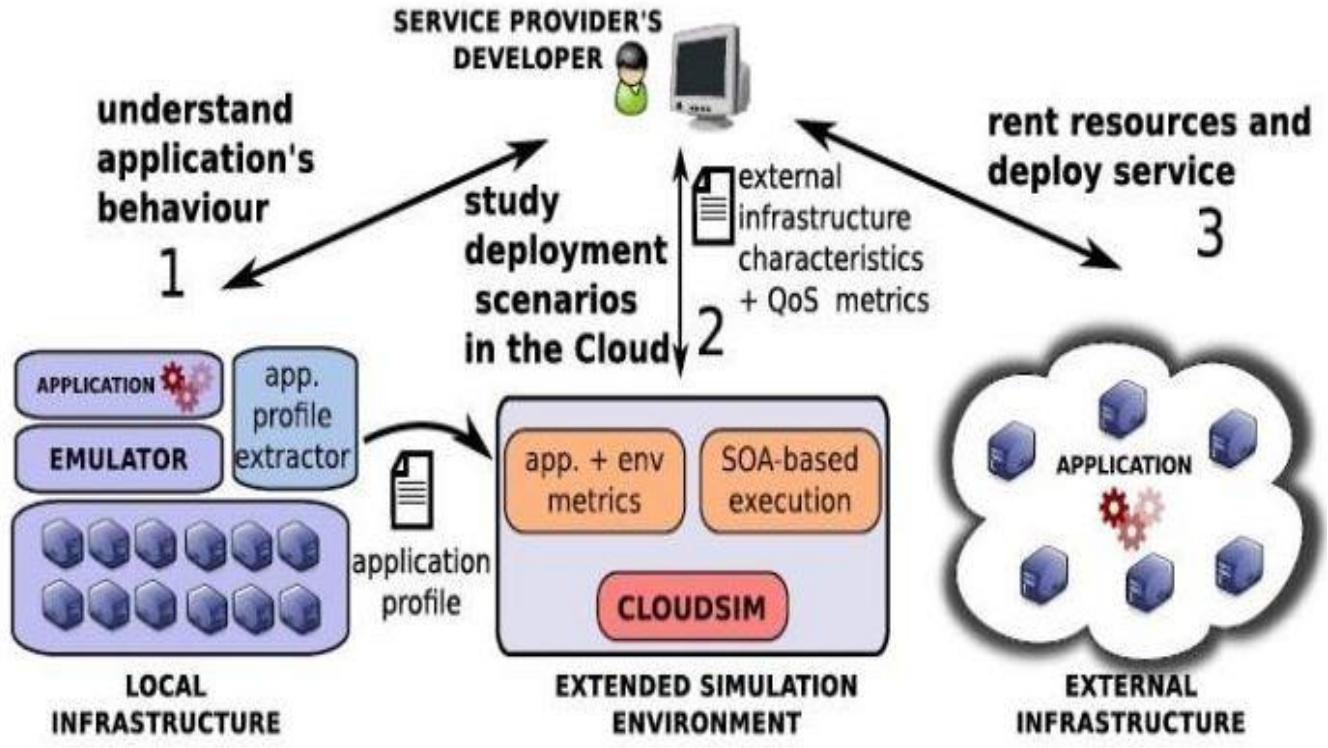

Figure 5. 6: Architecture of EMUSIM [34].

5.2.1.9. CDOSIM

CDOSim [34] is the Cloud Deployment Option (CDO) simulator that can simulate SLA violations, response times, and cost of the CDO. A CDO is a choices concerning test system which takes choice about the determination of a cloud supplier, particular runtime adjustment methodologies, segments organization of virtual machine and its examples setup. Component arrangement to virtual machine occasions incorporates the possibility of forming new parts of officially existing components. Virtual machine occurrences setup, refer to the instance type of virtual machine instances.

A comparison between different cloud simulators are shown below [34].

Table 4. 1: Comparison of cloud simulators.

Cloud Simulators	**GUI Support**	**Platform**	**Language**	**Support of TCP/IP**	**S/W or H/W**	**Availability**
CloudSim	Limited	GridSim	Java	none	S/w	Open Source
GreenCloud	Limited	Ns2	C++, Otcl	Full TCP/IP Protocol	S/W	Open Source
NetworkCloudSim	None	CloudSim	Java	None	S/W	Open source
EMUSIM	Limited	AEF, CloudSim	Java	None	S/W	Open Source
iCanCloud	GUI Supports	SIMCAN	Omnet, MPL, C++	None	S/W	Open Source
CDOSim	none	CloudSim	Java	No TCP/IP Support	S/W	NA
CloudAnalyst	Full	CloudSim	Java	limited	S/W	Open Source
CloudReport	Full	CloudSim	Java	limited	S/W	Open Source
CloudEXP	Full	CloudSim	Java	Full	S/W	No

CloudSim: Based on the above comparisons and descriptions CloudSim which is more appropriate with our requirements is selected. Also, the reason behind choosing CloudSim is that,

it allows cloud providers to test their services in a repeatable and controlled environment free of cost and to tune the performance bottlenecks before deploying on real world Clouds. By the simulation side, the simulation environment permits different kinds of resource allocation under different load distributions. CloudSim provides a good simulation framework for emerging cloud computing applications, so that researchers can investigate different design issues without getting concerned about low level details.

CloudSim [32] is a simulation tool that allows cloud developers to test the performance of their provisioning policies in a repeatable and controllable environment, free of cost. It helps tune the bottlenecks before real-world deployment. It is a simulator; hence, it doesn't run any actual software. It can be defined as 'running a model of an environment in a model of hardware', where technology-specific details are abstracted.

CloudSim is a library for the simulation of cloud scenarios. It provides essential classes for describing data centers, computational resources, virtual machines, applications, users, and policies for the management of various parts of the system such as scheduling and provisioning. Using these components, it is easy to evaluate new strategies governing the use of clouds, while considering policies, scheduling algorithms, load balancing policies, etc. It can also be used to assess the competence of strategies from various perspectives such as cost, application execution time, etc. It also supports the evaluation of Green IT policies. It can be used as a building block for a simulated cloud environment and can add new policies for scheduling, load balancing and new scenarios. It is flexible enough to be used as a library that allows you to add a desired scenario by writing a Java program.

The principal advantages of simulation are:

- Flexibility of defining configurations
- Ease of use and customization
- Cost benefits: First designing, developing, testing, and then redesigning, rebuilding, and retesting any application on the cloud can be expensive. Simulations take the building and rebuilding phase out of the loop by using the model already created in the design phase.
- CloudSim is a toolkit for modelling and simulating cloud environments and to assess resource provisioning algorithms.

5.3. CloudSim Classes

The class diagram design for the CloudSim is shown below [39]. Also details of classes which are working with CloudSim were shown below.

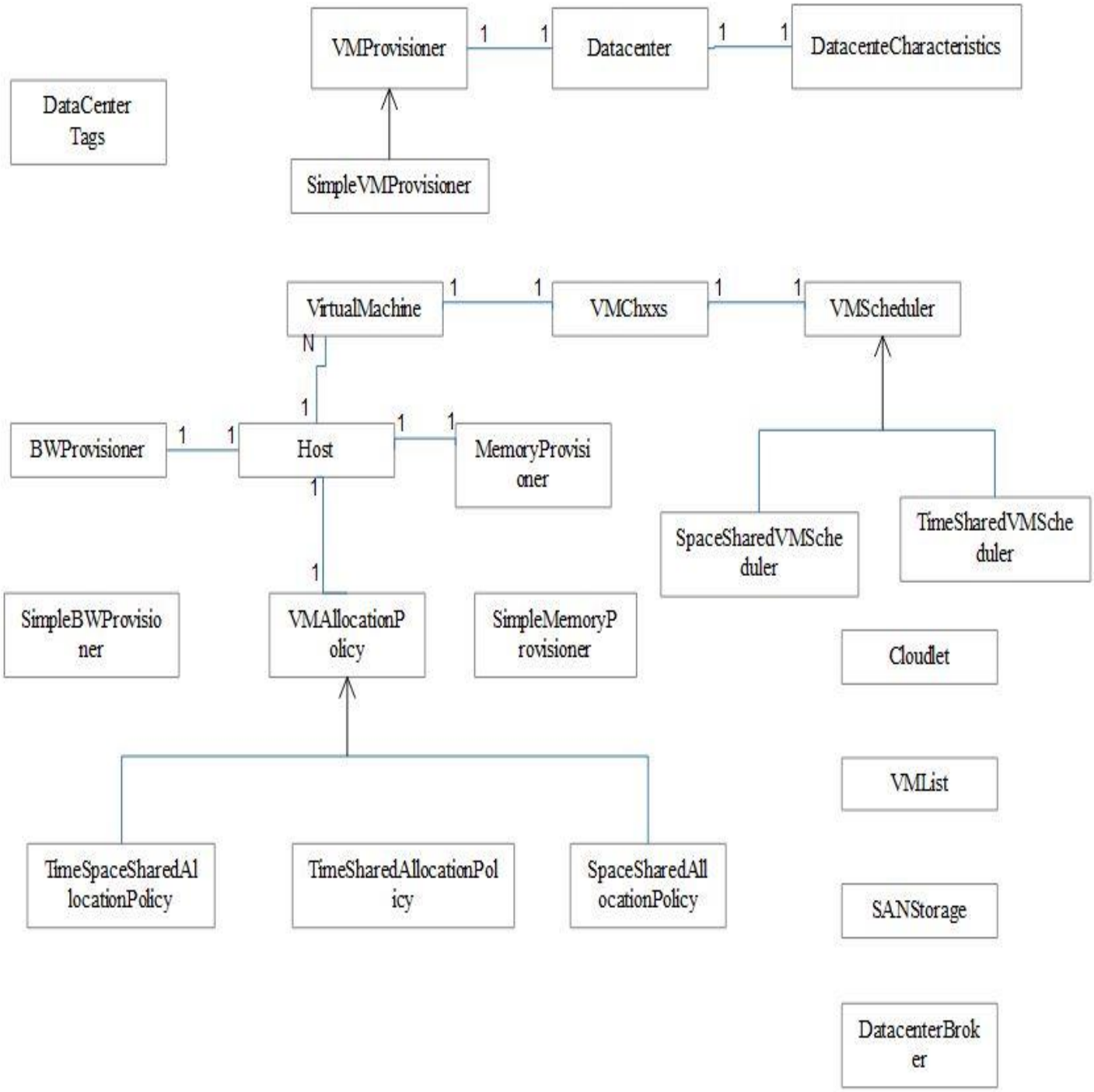

Figure 5. 7: Architecture of CloudSim classes [39].

1. **VM:**

This class models a VM, which is managed and hosted by a Cloud host component. Every VM components have access to a component which is extended from the abstract component called VM scheduling and these components can store the characteristics related to VM like memory, CPU, VM's internal provisioning policy storage etc.

2. **Host:**

This class models a physical resource such as a compute or storage server. It encapsulates important information such as the amount of memory and storage, a list and type of processing cores (to represent a multi-core machine), an allocation of policy for sharing the processing power among VMs, and policies for provisioning memory and bandwidth to the VMs.

3. **Datacenter:**

It is collection of computing hosts that can be either heterogeneous or homogenous as regard to their resource configuration such as memory, cores, storage and etc. Every datacenter instantiates a generalized resource provisioning component that implement a set of policies for allocating bandwidth, memory, and storage devices. Furthermore, every Datacenter component instantiates a generalized application provisioning component that implements a set of policies for allocating bandwidth, memory, and storage devices to hosts ad VMs.

4. **Datacenter Broker or Cloud Broker:**

Datacenter broker is a class which is responsible for an intermediate between user and service providers which depend on user's QoS requirements and deploys service tasks between clouds. The difference between the broker and the Cloud Coordinator is that the former represents the customer (i.e. decisions of these components are made in order to increase user-related performance metrics), whereas the latter acts on behalf of the data center, i.e. it tries to maximize the overall performance of the data center, without considering the needs of specific customers. The system developer can extend this class for conducting experiments with their custom developed application placement policies.

5. **Cloudlet:**

This class models the Cloud-based application services (such as content delivery, social networking, and business workflow). Cloudlet is also extended to represent the complexity of any applications in form of computational requirements.

6. Cloud Co-Coordinator:

This abstract class extends a Cloud-based data center to the federation. It is responsible for periodically monitoring the internal state of data center resources by the updateDataCenter() method by detecting queries sensors and based on that it undertakes dynamic load-shredding decisions.

7. Bandwidth Provisioner:

This is an abstract class that models the policy for provisioning of bandwidth to VMs. The main role of this component is to undertake the allocation of network bandwidths to a set of competing VMs that are deployed across the data center. Cloud system developers and researchers can extend this class with their own policies (priority, QoS) to reflect the needs of their applications. BandwidthProvionerSimple permits a VM to reserve as much as bandwidth as required. But, this is constrained by the total available bandwidth of the host.

8. Cloudlet Scheduler:

This abstract class is extended by the implementation of different policies that determine the share of processing power among Cloudlets in a VM. There two types of provisioning policies: Space-shared (CloudletSchedulerSpaceShared) and time-shared (CloudletSchedulerTimeShared).

9. Datacenter Characteristics:

This class contains configuration information of data center resources.

10. Network Topology:

This class contains the information for inducing network behavior (latencies) in the simulation and It stores the topology information, which is generated using the BRITE topology generator.

11. RAM Provisioner:

This is an abstract class that represents the provisioning policy for allocating primary memory (RAM) to the VMs. The execution and deployment of VM on a host is feasible only if the RAM Provisioner component approve that the host has the required amount of free storage.

12. SAN Storage:

This class models a storage area network that is commonly ambient in Cloud-based data centers for storing large chunks of data (such as Amazon S3, Azure blob storage). SAN Storage implements a simple interface that can be used to simulate storage and retrieval of any amount of data, subject to the availability of network bandwidth.

13. VM Scheduler:

This is an abstract class implemented by a Host component that models the policies (space-shared, time-shared) required for allocating processor cores to VMs. These policies are required for allocating the processing power to VMs.

14. VM Allocation Policy:

This abstract class represents a provisioning policy that a VM monitor utilizes for allocating VMs to hosts. The main purpose of this class is to select hosts that are available in datacenter that meets some conditions such as storage, memory and availability requirement for a VM deployment.

5.4. Simulation Methodology

The methods of the simulation processes are described in steps below.

Step1: Start and design the framework of the simulation.

Step2: We initialize VMs and its parameters.

Step3: We initialize Hosts and define the characteristics (features) of the Hosts.

Step4: Apply decision making algorithm for the selection of appropriate machine to allocate resources.

Step5: Select and allocate resources to the appropriate machine.

Step6: Evaluate the performance matrix.

Step7: Stop.

5.5. Common Simulation Configuration Activities

The implementation of the proposed Hybrid scaling resources is followed some common activities. These activities are described in the way.

1. **Datacenter Configuration: -** this class is a Cloud resource whose hosts are virtualized. For our simulation we have used one datacenter.
   - **Configure of Datacenter parameters**

     Datacenter (arch, os, vmm, hostlist, cost mem/bw/storage)

2. **Configure hosts parameters: -** this class describes the Physical Machine inside a Datacenter. It is used to executes actions related to management of virtual machines like creation and destruction. Inside host there are resources to virtualized like millions of instructions (mips) that run on virtual machines (VMs), storage and bandwidth.
   Host (mips, ram, storage, bandwidth)
3. **Configure VM with its parameters: -** Virtual machine (VM) is machine which is run inside host. VM is sharing host lists (Host resources) with other VMs available in that host. Also VMs are scaled up or scaled down with respect to tasks to shape itself with workload.
   MIPS, pesNumber (no. ofCPU), Ram(MB), BW(MB/s)
4. **Broker configuration:** brokers acting on behalf of users. It hides VM management, as VM creation, submit cloudlets (tasks) to the VMs and destruction of VMs when loads are decreased.
5. **Cloudlet configuration:** Since we were not using the real cloud datacenters, we have used CloudSim for simulating the proposed solution and take the cloudlets as a workload. Cloudlets has Length (MI), pesNumber, input Size, output Size that run on VMs.

5.6. Implementation

This section describes the implementation of proposed solution. The implementation begins by describing the basic steps to setup a cloud computing environment in cloudSim toolkit. Then we will explain the implementation of the proposed solution. The algorithm is implemented using java programming language in Eclipse IDE with the integration of cloudSim toolkit.

5.6. Steps for creating and running CloudSim

This shows steps for creating and running the simulated cloud infrastructure using CloudSim.

Steps for creating basic cloud Datacenter.

Step1: Initializing CloudSim package.

Step2: Creation of Datacenter.

Step3: Creation of broker.

Step4: Creation of VM.

Step5: Creation of cloudlets.

Step6: Start simulation.

Step7: Stop simulation.

Step8: Print the results.

5.7. Implementation Result

The result of the implementations has two cases. Both cases are performed based on the users input. First case is vertical resource scaling and the second is hybrid resource scaling. The input or users demand for the implementation is tasks (cloudlet). The task (cloudlet) is used as workloads in this simulation. The approach expected to check whether the available resources are enough for the input. Then perform the vertical or hybrid scaling resources.

User input code: this is users demand input or workload. We have already described in chapter that cloudlet was used as workload for the simulation.

Based on this input code user input his/ her request(demand). If the demand is less than or equal to the capacity of vertical scaling, vertical scaling is performed. If the demand is greater than the capacity of vertical scaling vertical scaling if performed up to its capacity and horizontal scaling will be performed.

As our experiment vertical resource scaling approach is performed up to the below condition is true. This means, the capacity of the vertical scaling is supported to scale itself with respect to below condition. These conditions are:

- Cloudlet (task) Length (mi) is less than or equal to 2000
- Cloudlet (task) File size is less than or equal to 2000.
- Cloudlet (task) No. of Pes is less than 14

```
Enter the length of tasks:
1900
Enter file size of tasks:
1900
Enter no of PES:
10
```

The result of the simulation after inputing tha above parameters. It performs vertical resourcs scaling approach. Because, vertical scaling is enough to allocate the requested demand and the proposed approach firstly performs vertical scaling approach..

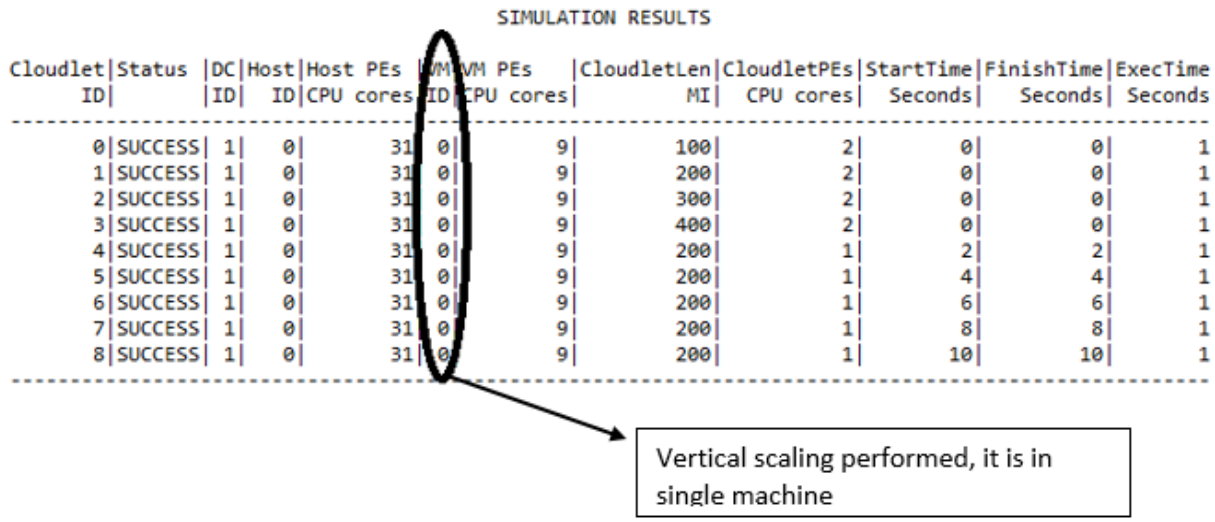

Figure 5.8: First simulation test

If the requests (demands) are greater than the upper threshold of the vertical scaling, next to vertical scaling it performs horizontal scaling by creating an extra virtual machine to handle user's requests. The condition that listed below was greater the capacity of vertical scaling. For this reason, it was performed the horizontal resource scaling next to vertical resource scaling.

- Cloudlet (task) Length (mi) is greater than or equal to 2000
- Cloudlet (task) File size is greater than or equal to 2000.
- Cloudlet (task) No. of Pes is greater than 14

```
Enter the length of tasks:
2500
Enter file size of tasks:
2500
Enter no of PES:
20
```

The result of the simulation after inputing tha above parameters. It performs both resourcs scaling approach. Because, vertical scaling is not enough to allocate the requested demand.

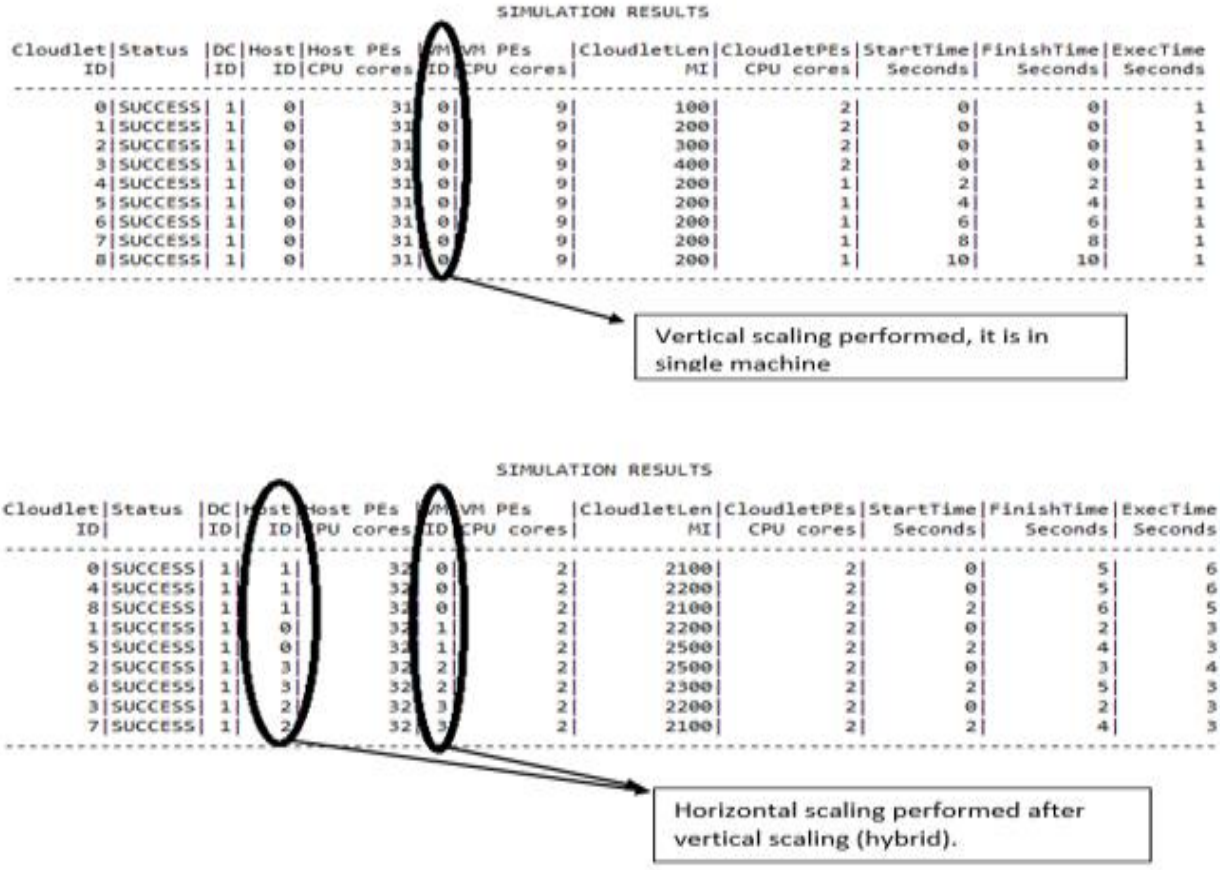

Figure 5.9: Second simulation test

CHAPTER SIX

6. Evaluation and Discussion

In this chapter, the evaluation of the proposed hybrid scaling resources method simulated on CloudSim by taking the cloudlets as user's requests (workloads) or tasks was described. The goal of the system evaluation is used to measure how much the goal of allocating resources, using hybrid resource scaling effectively is achieved.

6.1. Analysis of Proposed and Existing Algorithm

The analysis of these algorithms are related with the time it taken to allocate resources for the users based on their requests and using resources efficiently. By considering the number of requested to be allocated resources to the users and to be used resources efficiently the following algorithm is designed.

1. ***Resource scaling up.***

Input ← Workload or User_demand (Resource_to_be_allocated)

Output ← Resource_allocated (CPU, RAM)

Begin

1: *for i← 0 to n do*

2: *Workload or user_demend_to_be_allocated*

3: *If single_machine (resource_available && resource_available > user_demand) ← true*

4: ***//Vertical scaling***

5: *User_demand(allocate_resource)*

6: *If single machine (host_has_capacity_to_scale_up) ← true*

7: *Resize_up_to_be_allocated (CPU, RAM)*

8: *Else*

9: ***//Horizontal Scaling***

10: *Create_extra_machines_to_be_allocated (allocate_resources)*

11: *End if*

2. ***Resource scaling down/release***

Input: User_request (to_be_scaleDown/release_resources)

Output: resource_scaleDown/release ()

Begin:

12: User_request (to_be_ScaleDown/release)

13: If (User_rquest < Currently_AllocatedResources) ← true

*14: **//Vertical scaling***

15: If (Single_Machine) ← true

16: ScaleDown/release (AllocatedResources)

17: Else

18: //Horizontla scaling

19: ScaleDown/release (AllocatedResources)

20: End if

In case of related algorithm which has been used to allocate resources for users by using other approach are found [31]. Which makes this algorithm different from the proposed algorithm is, firstly it performs the horizontal scaling methods to allocate resources.

Input: $RSt = \{Rt,1 ,Rt,2,. . . , Rt,m\}$, $Tslo$, $isHScaler = true$, $Tvscaler = 1$, $Thscaler = 60$

Output: $RSt+1$, updated in each $Tvscaler$ period

Begin

//*Period of horizontal scaling is Thscaler, while Tvscaler is for periodically detecting and vertical scaling*

1: $detectRound = Thscaler/Tvscaler$

2: while true do

3: $W St \Leftarrow GetLastW orkloadState()$

4: if $isHScaler$ then

5: $W St+1 \Leftarrow P redictT hscaler(W St)$

6: $HorizontalAdjusting(W St+1 , RSt)$

7: $detectRound = detectRound - 1$

8: $Calmdown(Tvscaler)$

```
9:          / /check if it is tolerance to release some vertical resources
10:         Pt ⇐ GetLastP erformance(Tslo)
11:           W St ⇐ GetLastW orkloadState()
12:           if Pt < 1% then
13:                 V ScaleDown(W St, RSt)
14:                 detectRound = detectRound − 1
15:                 Calmdown(Tvscaler)
16:           end if
17:           isHScaler = false
18:         else
19:           Pt ⇐ GetLastP erformance(Tslo)
20:           if Pt < 1% then
21:                 V ScaleDown(W St, RSt)
22:         else
23:                 if Pt > 5% then
24:                 V ScaleUp(W St, RSt)
25:               end if
26:          end if
27:          detectRound = detectRount − 1
28:          if detectRound == 0 then
29:                 isHScaler = true
30:                 detectRound = Thscaler/Tvscaler
31:          end if
32:         Calmdown(Tvscaler)
33:       end if
34:     end while
      End
```

6.1.2. Comparison of Algorithms

Running time of these two algorithms for specific input depends on the number of inputs executed was evaluated. Running time of the algorithm is depending on different metrics such as:

1. Single vs multiprocessor x
2. Read/write speed to memory x
3. 32 vs 64 bit, and x
4. Input size → in case of the proposed study input size is number of request (demand) of users

In our case how the algorithms behave for various inputs, or how the time taken by the algorithms growth with in input to the algorithms. The time complexity of existing and proposed algorithms are provided below.

Table 6. 1: Running time evaluation algorithms

Previous algorithm			**Proposed algorithm**		
Line	Cost	No of times	line	cost	No of times
1	1	1	1	2	n+1
2	1	n+1	2	1	n
3	2	n	3	2	n
4	1	n	4	-	-
5	1	n	5	1	n
6	1	n	6	1	n
7	2	n	7	1	n
8	1	n	8	-	-
9	1	n	9	-	-
10	1	n	10	1	n
11	1	n	11	-	-
12	1	n	12	1	n
13	1	n	13	1	n
14	2	n	14	-	-
15	1	n	15	1	n

16	-	-	16	1	n
17	1	n	17	-	-
18	-	-	18	-	-
19	1	n	19	1	n
20	1	n	20	-	-
21	1	n			
22	-	-			
23	1	n			
24	1	n			
25	-	-			
26	-	-			
27	2	n			
28	1	n			
29	1	n			
30	2	n			
31	-	-			
32	1	n			
33	-	-			
34	-	-			
total		30n+2			14n+2

The graphical representation of the comparison of the existing and proposed solution based on the same input for both of them are shown below.

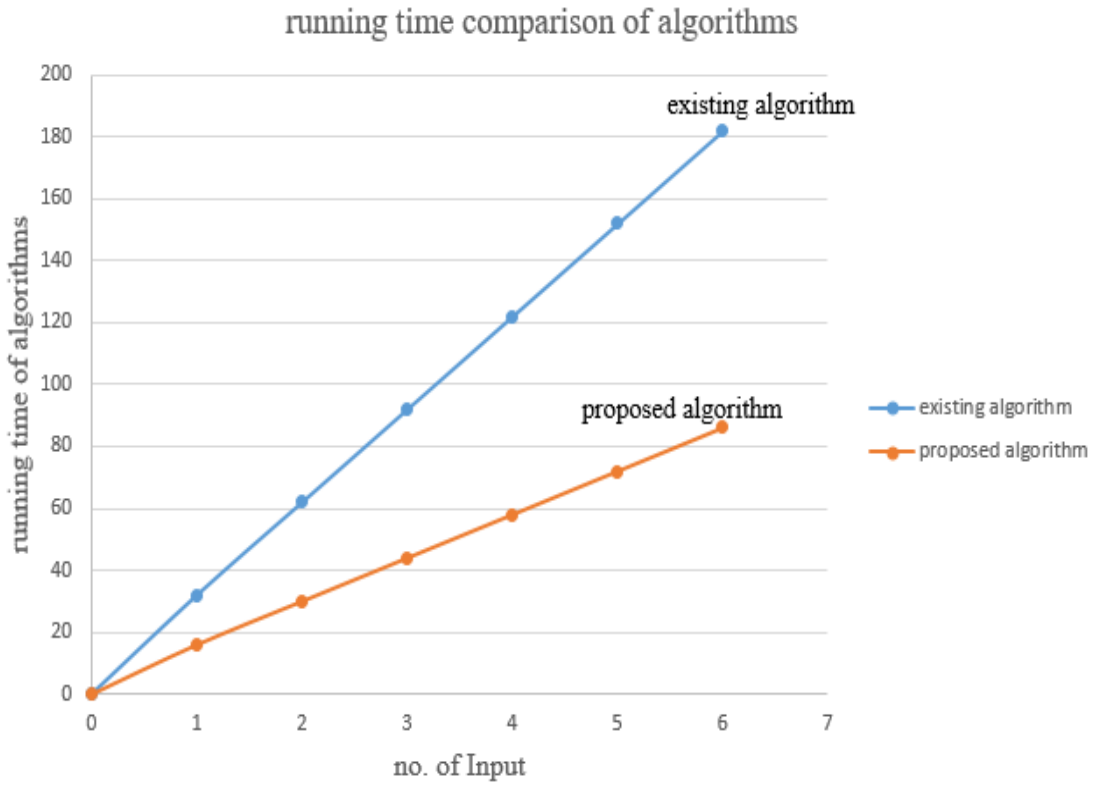

Figure 6. 1: Running time comparison.

6.2. Evaluation of Proposed Solution

6.2.1. Cost Based Evaluation: wen we are taking about cost in scaling resources there some conditions that are depends on cost such as:

- virtual machine creation: this describes virtual machine creation or migration if additional machines are needed to allocate resources.
- bandwidth: this describes bandwidth consumed to transfer request from one machine to another machine to allocate resources.

Evaluating the proposed method related with the cost, the proposed method is cost effective because firstly it performs vertical scaling or single virtual machine which does not need creation of an extra virtual machine and bandwidth to scale up and down. Because, vertical scaling not need

more machines, more network bandwidth, more resources, and etc. to scaling. But, horizontal scaling method needs more machines, more network bandwidth, and more resources to scale. The proposed method is going to horizontal scaling method only if there are no resources in vertical scaling.

6.2.2. Resource Utilization Based Evaluation: Resource utilizations describe the usage of resources such as: RAM, CPU, and etc. available in the virtual machine effectively. The proposed scaling method cannot be creating virtual machine (VM) and allocate resources before allocating all resources available in single machine up to the capacity of the that machine. Even if, the workloads become more, it resizes itself with respect to its capacity.

By considering resource utilizations, the proposed techniques go to horizontal scaling method if there are no resources in vertical scaling and vertical scaling has no capacity to maximize itself only. This describes before utilizing the resources available in the vertical scaling it doesn't go to horizontal scaling.

6.2.3. Delay of Time to Allocate Resource Based Evaluation: The proposed solution scaling method firstly performs vertical scaling, it resizes itself rather than creating extra virtual machines (VMs) to allocate resources. Because horizontal scaling needs time to creating extra virtual machines (VMs). When we see this by example, the cloud user wants 10 CPU but in vertical scaling available CPU is 8; but the machine can be maximizing itself up to 12 CPU (capacity of the single machine to resizes itself; in horizontal scaling there is only 8 CPU. So the for the user no need to create extra virtual machine (VM) to allocate resources; simply resizes the virtual machine (VM) and allocates for the users.

The another reason of firstly selecting vertical scaling technique to perform scaling in our proposed system is, vertical scaling has lower configuration latency in comparison to horizontal scaling.

CHAPTER SEVEN

7. Conclusion and Future Work

In this chapter we described the summary of the proposed solution and future research directions.

7.1. Conclusion

Today, there are two types of resource scaling techniques in cloud server. Each has their scaling mechanism. First, Horizontal resource scaling is performing the scaling mechanism by adding an extra server. Second, vertical resource scaling is performing the scaling mechanism by resizing itself.

In cloud server virtualization, effectively idle resource usage is the challenge of the cloud service providers. Several steps and procedures are followed to propose the proposed solution. In the beginning, it focuses on types of scaling techniques and collecting their qualitative matrices from documents and websites. Then proposed the hybrid scaling method for dynamic workloads. This proposed method present is an important resource scaling method that arrange the available resources and demand to perform an effective resource scaling methods. The proposed method is the hybrid of horizontal and vertical resource scaling approaches that considers available resources and current workloads to scale up and scale down.

Generally, the proposed method would result in satisfying users demand; maximize resource utilization, and improving the performance of workloads.

7.2. Recommendation

Resource scaling in cloud computing is important to satisfy users demand. Hence, we recommended to future researchers to design another architecture and algorithm or enhance the algorithm used in this study to increase the performance of the system.

7.3. Future work

This study was dedicated on hybrid scaling resources for dynamic workloads to develop best resource scaling method that satisfy users demands and maximize resource utilizations. The researcher recommended the following future works to be continued for any new researcher who have motivated.

1. The proposed solution techniques need to be implement and applicable in real cloud service providers.
2. The proposed algorithm is to be performed to measure the mathematical equation developed to calculate the total cost of virtual machines.

References

[1] Mell, P., and Grance, T., “The NIST definition of cloud computing. National Institute of Standards and Technology,” 2011.

[2] https://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&act=8&ved=0ahUKEwjJ36W_rXAhXKPhQKHbO2BN4QFggMAI&url=http%3A%2F%2Fwww.thbs.com%2Fdownlods%2FCloud-Computing-Overview.pdf&usg=AOvVaw2z-qaFk1WvxY5oiSU6nFJC.

[3] DC plummer, TJ Bittman, Austin, T., Clearley, D. and DM Smith, 12 05 2012. [online]. [Accessed January 12 2018].

[4] D.C. Marinnescu, Cloud Computing Theory and Practice Cloud Computing Theory and Practice, Network: Elsevier In, 3013, p. 132.

[5] https://dzone.com/articles/vertical-scaling-and-horizontal-scaling-in-aws.

[6] Dr. Basem Suleiman Web Application Engineering School of Computer Science and Engineering UNSW.

[7] Amazon web Services, https://amazon EC2.microsoft.com/enin/services/sql-database/.

[8] Microsoft Azure, https://azure.microsoft.com/enin/services/sql-database/.

[9] IBM Blue Mix, http://www.ibm.com/ cloudcomputing/bluemix/.

[10] Google App Engine, https://cloud.google.com/appengine/ Google App Engine, https://cloud.google.com/appengine/

[11] Liu, Chien-Yu, Meng-Ru Shie, Yi-Fang Lee, Yu-Chun Lin, and Kuan-Chou Lai.” Vertical/Horizontal Resource Scaling Mechanism for Federated Clouds.” *In Information Science and Applications (ICISA), 2014 International Conference* on, pp.1 -4. IEEE, 2014

[12] Wang, Wenting, Haopeng Chen, and Xi Chen.” An availability-aware virtual machine placement approach for dynamic scaling of cloud applications.” *In Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2012 9th International Conference* on, pp. 509-516. IEEE, 2012

[13] Bellenger, Dominique, Jens Bertram, Andy Budina, ArneKoschel, Benjamin Pfnder, Carsten Serowy, Irina Astrova, Stella Gatziu Grivas, and Marc Schaaf.” Scaling in cloud environments.” Recent Researches in Computer Science (2011)

[14] Kirschnick, Johannes, Jose M. Alcaraz Calero, Lawrence Wilcock, and Nigel Edwards.” Toward an architecture for the automated provisioning of cloud services.” Communications Magazine, IEEE 48, no. 12 (2010):124-131

[15] https://www.tutorialspoint.com/loud_computing/cloud_computing architecture.html (accessed aprl 23 2018).

[16]https://www.google.com/search?q=cloud+computing+deployment+models&tbm=isch&tbo=u&source=univ&sa=X&ved=2ahUKEwjrvqP-hsfdAhUQaBoKHVnMB4EQsAR6BAgEEAE.

[17] M.O’Neill, “Connecting to the cloud, Part 1: Leverage the cloud in applications. IBM.,” Retrieved May 26 2010. [Online]. Available: http:// *www.ibm.com/developersworks/webservices/library/x-cloudpt1/index.html.* [Accessed Retrieved March 8, 2018].

[18] Jackson, K. (n.d.)., May 2010. [Online]. Available: http://www.xmind.net/share/_embed/kvjacksn/cloud-computing-101/. [Accessed March 13, 2018].

[19] T.Merrill, Cloud Computing: Is your Company Weighting both Benefits and risks, Philadelphia, United States: ACE USA, April 2014.

[20] M. Rosenblen, “The reincarnation of virtual machines,”08 2004.

[21] R.M. Sharma, “the impact of virtualization in cloud computing, “international jornal of recent development in engineering and technology, vol.3, no. 1,07 2014.

[22] H. Saltzer and M.F. Kaashoek, principles of computer system design., network: Morgan Kaufamnn, 2009.

[23] R.M.Sharma, “The Impact of Virtualization in Cloud Computing,” *International Journal of Recent Development in Engineering and Technology*, vol. 3, no. 1, 07 2014.

[24] E. M. V8, "Professionally diagram and communicate with essential Edraw solution," Edraw soft, 2004 - 2016.

[25] Tao Li, Jingyu Wang, Wei Li, Tong Xu, Qi. “Load Prediction-based Automatic Scaling Cloud Computing.” 2016 *International Conference on Networking and Network Applications*.

[26] Nilabja Roy, Abhoshek Dubey and Anirudda Gokhale. “Efficient Auto scaling in the Cloud using Predictive Models for Workload Forecasting.”

[27] Wathit Chaloemwat and Sukumal Kitisin. “Horizontal Auto-Scaling and Process Migration Mechanism for Cloud Services with Skewness Algorithm.” 2016 13th *International Joint Conference on Computer Science and Software Engineering (JCSSE).*

[28] Zhen Xiao Weijia Song, and Qi Chen “Dynamic Resource Allocation Using Virtual Mchines for Cloud Computing Environment.” *IEEE Transactions on parallel and distributed systems,* vol. 24, no. 6, 06 2013.

[29] Jiayin Li, MeikangQui Jian-Wei Niu, Yu Chen, Zhong Ming, “Adaptive Resource Allocation for pre-empt able Jobs in Cloud systems,” *in 10th international conference on Intelligent System Design and Application,* 2011.

[30] Lenar Yasdanov, Christof Fetzer. “Vertical Scaling for Prioritized VMs Provisioning, 2012.”

[31] Song Wu, Binji Li, Xinhou Wang, Hai Jin. “Hybrid Scaler: Handling Bursting Workload for Multi-Tier Web Applications, 2016.”

[32] J. Gustedt E. Jeannot, and M. Quinson, “Expermental methodologies for large scale systems: Asurvey,” Vol. 19, no. 03. 2009. Pp. 399-418.

[33 Raj Kumar Rajiv Ranjon, Rodrigo N. Calherios. “Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim toolkit: Challenges and opportunities, 2009.”

[34] Utkal Sinha, Mayank Shekhar,” Comparison of various cloud simulation tools available in cloud computing 2015,” *International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 3, March 2015.*

[35] Parveen Kumar*, Anjandeep Kaur Rai "An Overview and Survey of Various Cloud Simulation Tools," *Journal of Global Research in Computer Science,* Volume 5, No. 1, January 2014.

[36] B. Wickremassingle, "CloudAnalayst: A CloudSim based Tool for Modeling and Analysis of large Scale Cloud Computing environments", MEDC project, 2009.

[37] Alberto Nunez- Jose L. Vasquez-Poletti-Agustim C, C. Caminero-Gebriel G. Castane- Jesus Carretero- Ignacio M. Llorente, "iCanCloud: Flexible and Scalable Cloud Infrastructure Simulator."

[38] R. B. Saurabh Kumar Garg, "NetworkCloudSim: Modelling Parallel applications in cloud simulations," *in Fourth IEEE International Conference on Utility and Cloud Computing*, 2011.

[39] R. N. Calheiros, R. Ranjan, A. Beloglazov, Cesar A. F. De Rose, Rajkumaar Buyya "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms". Cloud Computing and Distributed Systems (CLOUDS) Laboratory Department of Computer Science and Software Engineering the University of Melbourne, Australia.

APPENDIXES

Appendix 1: Datacenter Configuration

```
String arch = "x86";
             String os = "Linux";
             String vmm = "Xen";
             double time_zone = 9.0;
             double cost = 3.0;
             double costPerMem = 0.08;
             double costPerStorage = 0.003;
             double costPerBw = 0.0;
             LinkedList<Storage> storageList = new LinkedList<Storage>();

     DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
     arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage,costPerBw);
```

Appendix 2: Hosts Configuration

```
rivate static final int HOSTS = 1;
          private static final int HOST_PES = 32;

private Host createHost() {
              List<Pe> peList = new ArrayList<>(HOST_PES);
              for (int i = 1; i < HOST_PES; i++) {
                  peList.add(new PeSimple(1000, new PeProvisionerSimple()));

         }

              final long ram = 20000; //in Megabytes
              final long bw = 100000; //in Megabytes
              final long storage = 10000000; //in Megabites/s
              final int id = hostList.size();
              return new HostSimple(ram, bw, storage, peList)
```

Appendix 3: VMs Configuration

```
private List<Vm> createListOfScalableVms(final int numberOfVms) {
              List<Vm> newList = new ArrayList<>(numberOfVms);
              for (int i = 0; i < numberOfVms; i++) {
                  Vm vm = createVm();
                  vm.setPeVerticalScaling(createVerticalPeScaling());
                  newList.add(vm);
```

Appendix 4: Broker Configuration

```
createDatacenter();
        broker0 = new DatacenterBrokerSimple(simulation);

        vmList.addAll(createListOfScalableVms(VMS));

        createCloudletListWithDifferentDelays();
        broker0.submitVmList(vmList);
        broker0.submitCloudletList(cloudletList);
```

Appendix 5: Cloudlet Configuration

```
private Cloudlet createCloudlet(final long length, final int numberOfPes) {
              return createCloudlet(length, numberOfPes, 0);
          }
```

Appendix 6: Sample code of vertical Resource scaling

```
package mySimulatioHybrid;

public class verticalScaling extends Thread {

        private static final int SCHEDULING_INTERVAL = 1;

          private static final int HOSTS = 1;

          private static final int HOST_PES = 32;

          private static final int VMS = 1;

          private static final int VM_PES = 14;

          private final CloudSim simulation;

          private DatacenterBroker broker0;

          private List<Host> hostList;

          private List<Vm> vmList;

          private List<Cloudlet> cloudletList;

          private int createsVms;

          public static void main(String[] args) {
```

```
        new verticalScaling();
    }
    verticalScaling() {
        hostList = new ArrayList<>(HOSTS);
        vmList = new ArrayList<>(VMS);
        cloudletList = new ArrayList<>(CLOUDLETS);
        simulation = new CloudSim();
        simulation.addOnClockTickListener(this::onClockTickListener);
        createDatacenter();
        broker0 = new DatacenterBrokerSimple(simulation);
        vmList.addAll(createListOfScalableVms(VMS));
        createCloudletListsWithDifferentDelays();
        broker0.submitVmList(vmList);
        broker0.submitCloudletList(cloudletList);
        simulation.start();
        printSimulationResults();
    }
    private void onClockTickListener(EventInfo evt) {
        vmList.forEach(vm ->
                evt.getTime(), vm.getId(), vm.getCpuPercentUsage()*100.0,
vm.getNumberOfPes(),
            vm.getCloudletScheduler().getCloudletExecList().size(),
```

```
                    vm.getRam().getPercentUtilization()*100,
vm.getRam().getAllocatedResource())
                );
            }
            private void printSimulationResults() {
                final List<Cloudlet> finishedCloudlets = broker0.getCloudletFinishedList();
                final Comparator<Cloudlet> sortByVmId = comparingDouble(c ->
c.getVm().getId());
                final Comparator<Cloudlet> sortByStartTime =
comparingDouble(Cloudlet::getExecStartTime);
                finishedCloudlets.sort(sortByVmId.thenComparing(sortByStartTime));


                new CloudletsTableBuilder(finishedCloudlets).build();
            }
            private void createDatacenter() {
                for (int i = 0; i < HOSTS; i++) {
                    hostList.add(createHost());
                }
                Datacenter dc0 = new DatacenterSimple(simulation, hostList, new
VmAllocationPolicySimple());
                dc0.setSchedulingInterval(SCHEDULING_INTERVAL);
            }
            private Host createHost() {
                List<Pe> peList = new ArrayList<>(HOST_PES);
```

```
    for (int i = 1; i < HOST_PES; i++) {
        peList.add(new PeSimple(1000, new PeProvisionerSimple()));
    }
    final long ram = 20000; //in Megabytes
    final long bw = 100000; //in Megabytes
    final long storage = 10000000; //in Megabites/s
    final int id = hostList.size();
    return new HostSimple(ram, bw, storage, peList)
        .setRamProvisioner(new ResourceProvisionerSimple())
        .setBwProvisioner(new ResourceProvisionerSimple())
        .setVmScheduler(new VmSchedulerTimeShared());
}
private List<Vm> createListOfScalableVms(final int numberOfVms) {
    List<Vm> newList = new ArrayList<>(numberOfVms);
    for (int i = 0; i < numberOfVms; i++) {
        Vm vm = createVm();
        vm.setPeVerticalScaling(createVerticalPeScaling());
        newList.add(vm);
    }
    return newList;
}
private Vm createVm() {
    final int id = createsVms++;
```

```
        return new VmSimple(id, 1000, VM_PES)
            .setRam(VM_RAM).setBw(1000).setSize(10000)
            .setCloudletScheduler(new CloudletSchedulerTimeShared());
    }

    private VerticalVmScaling createVerticalPeScaling() {
        //The percentage in which the number of PEs has to be scaled
        final double scalingFactor = 0.1;
        VerticalVmScalingSimple verticalCpuScaling = new
VerticalVmScalingSimple(Processor.class, scalingFactor);
        verticalCpuScaling.setResourceScaling(vs->
2*vs.getScalingFactor()*vs.getAllocatedResource());
verticalCpuScaling.setLowerThresholdFunction(this::lowerCpuUtilizationThreshold);
verticalCpuScaling.setUpperThresholdFunction(this::upperCpuUtilizationThreshold);
        return verticalCpuScaling;
    }

    private double lowerCpuUtilizationThreshold(Vm vm) {
        return 0.4;
    }

    private double upperCpuUtilizationThreshold(Vm vm) {
        return 0.8;
    }

    private void createCloudletListsWithDifferentDelays() {
        final int initialCloudletsNumber = (int)(CLOUDLETS/2.5);
```

```
            final int remainingCloudletsNumber = CLOUDLETS-initialCloudletsNumber;
            //Creates a List of Cloudlets that will start running immediately when the simulation
starts
            for (int i = 0; i < initialCloudletsNumber; i++) {
                cloudletList.add(createCloudlet(CLOUDLETS_INITIAL_LENGTH+(i*100), 2));
            }
            for (int i = 1; i <= 5; i++) {
                cloudletList.add(createCloudlet(CLOUDLETS_INITIAL_LENGTH*2, 1,i*2));
            }
        }
        private Cloudlet createCloudlet(final long length, final int numberOfPes) {
            return createCloudlet(length, numberOfPes, 0);
        }
        private Cloudlet createCloudlet(final long length, final int numberOfPes, final double
delay) {
            final UtilizationModel utilizationCpu = new UtilizationModelFull();
            final UtilizationModel utilizationModelDynamic = new
UtilizationModelDynamic(1.0/CLOUDLETS);
```

Appendix 7: Sample code of Horizontal Resource scaling

```
package mySimulatioHybrid;

public class horizontalScaling extends Thread {

    private static final int SCHEDULING_INTERVAL = 5;

    private static final int HOSTS = 10;

    private static final int HOST_PES = 32;

    private static final int VMS = 4;

    private final CloudSim simulation;

    private DatacenterBroker broker0;

    private List<Host> hostList;

    private List<Vm> vmList;

    private List<Cloudlet> cloudletList;

    private static final long[] CLOUDLET_LENGTHS = Cloudlet ();{

    private ContinuousCloudletDistribution rand;

    private int createdCloudlets;

    private int createsVms

    public static void main(String[] args) {

        new horizontalScaling();

    }

    public horizontalScaling() {

        final long seed = 1;

        rand = new UniformDistr(0, CLOUDLET_LENGTHS.length, seed);

        hostList = new ArrayList<>(HOSTS);
```

```
    vmList = new ArrayList<>(VMS);
    cloudletList = new ArrayList<>(CLOUDLETS);
    simulation = new CloudSim();
    simulation.addOnClockTickListener(this::createNewCloudlets);
    createDatacenter();
    broker0 = new DatacenterBrokerSimple(simulation);
    broker0.setVmDestructionDelayFunction(vm -> 10.0);
    vmList.addAll(createListOfScalableVms(VMS));
    createCloudletList();
    broker0.submitVmList(vmList);
    broker0.submitCloudletList(cloudletList);
    simulation.start();
    printSimulationResults();
}
private void printSimulationResults() {
    List<Cloudlet> finishedCloudlets = broker0.getCloudletFinishedList();
    Comparator<Cloudlet> sortByVmId = comparingDouble(c -> c.getVm().getId());
    Comparator<Cloudlet> sortByStartTime = comparingDouble(c -> c.getExecStartTime());
    finishedCloudlets.sort(sortByVmId.thenComparing(sortByStartTime));
    new CloudletsTableBuilder(finishedCloudlets).build();
}
        List<Cloudlet> newCloudlets = new ArrayList<>(numberOfCloudlets);
        for (int i = 0; i < numberOfCloudlets; i++) {
```

```
            Cloudlet cloudlet = createCloudlet();
            cloudletList.add(cloudlet);
            newCloudlets.add(cloudlet);
        }
        broker0.submitCloudletList(newCloudlets);
    }
}
private void createDatacenter() {
    for (int i = 0; i < HOSTS; i++) {
        hostList.add(createHost());
    }
    Datacenter dc0 = new DatacenterSimple(simulation, hostList, new
VmAllocationPolicySimple());
    dc0.setSchedulingInterval(SCHEDULING_INTERVAL);
}
private Host createHost() {
    List<Pe> peList = new ArrayList<>(HOST_PES);
    for (int i = 0; i < HOST_PES; i++) {
        peList.add(new PeSimple(1000, new PeProvisionerSimple()));
    }
    final long ram = 2048; // in Megabytes
    final long storage = 1000000; // in Megabytes
    final long bw = 10000; //in Megabits/s
```

```
    return new HostSimple(ram, bw, storage, peList)
        .setRamProvisioner(new ResourceProvisionerSimple())
        .setBwProvisioner(new ResourceProvisionerSimple())
        .setVmScheduler(new VmSchedulerTimeShared());
}
private List<Vm> createListOfScalableVms(final int numberOfVms) {
    List<Vm> newList = new ArrayList<>(numberOfVms);
    for (int i = 0; i < numberOfVms; i++) {
        Vm vm = createVm();
        createHorizontalVmScaling(vm);
        newList.add(vm);
    }
    return newList;
}
private void createHorizontalVmScaling(Vm vm) {
    HorizontalVmScaling horizontalScaling = new HorizontalVmScalingSimple();
    horizontalScaling
        .setVmSupplier(this::createVm)
        .setOverloadPredicate(this::isVmOverloaded);
    vm.setHorizontalScaling(horizontalScaling);
}
private boolean isVmOverloaded(Vm vm) {
    return vm.getCpuPercentUsage()
```

```
  }

    private Vm createVm() {
        final int id = createsVms++;
        return new VmSimple(id, 1000, 2)
            .setRam(1000).setBw(1000).setSize(9000)
            .setCloudletScheduler(new CloudletSchedulerTimeShared());
    }

    private Cloudlet createCloudlet() {
        final int id = createdCloudlets++;
        final long length = CLOUDLET_LENGTHS[(int) rand.sample()];
        UtilizationModel utilization = new UtilizationModelFull();
    }
}
```

Appendix 8: Sample code of Hybrid Resource Scaling

```
import java.util.*;
public class HybridSimulation {

	public static void main(String[] args) {

 int CloudletLength,CloudletFileSize,numberOfPES;
 Scanner input=new Scanner(System.in);
 System.out.println("Enter the length of tasks:");
 CloudletLength=input.nextInt();
 System.out.println("Enter file size of tasks:");
 CloudletFileSize=input.nextInt();
 System.out.println("Enter no of PES:");
 numberOfPES=input.nextInt();
 if(CloudletLength>capacityOfLengthVerticalScaling ()||
              CloudletFileSize>capacityOfLengthVerticalScaling()||
              numberOfPES>capacityOfLengthVerticalScaling())
 {
        Thread t1 =new verticalScaling();
        t1.start();
        Thread t2=new horizontalScaling();
        try {
              t1.join();
       } catch (InterruptedException e) {
              // TODO Auto-generated catch block
              e.printStackTrace();
       }
        t2.start();
 }
 else
 {
        new verticalScaling();
 }
       }
```