DEPLOYMENT OF VIRTUAL MACHINES FOR TIERED APPLICATIONS IN CLOUD SYSTEMS WITH OPTIMIZED RESOURCE ALLOCATION BASED ON AVAILABILITY SLAS

by

PRANEETH SAKHAMURI

Bachelor of Technology

GITAM University

Visakhapatnam, AP, INDIA 2014

A thesis presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Applied Science

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2017

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

ABSTRACT

DEPLOYMENT OF VIRTUAL MACHINES FOR TIERED APPLICATIONS IN CLOUD SYSTEMS WITH OPTIMIZED RESOURCE ALLOCATION BASED ON AVAILABILITY SLAS

Praneeth Sakhamuri, M.A.Sc.
Electrical and Computer Engineering
Ryerson University, 2017

Deploying and managing high availability-tiered application in the cloud is challenging, as it requires providing necessary virtual machines to make the application available. An application is available if it works and responds in a timely manner for varying workloads. Application service providers need to allocate specified number of working virtual machine copies for each server with at least a given minimum computing power, to meet the response time requirement. Otherwise, we may end up with response time failures. This thesis formulates an optimization problem that determines the number and type of virtual machines needed for each server to minimize the cost and at the same time guarantees the availability SLA (Service-Level Agreement) for different workloads. The results demonstrate that a diverse approach is more cost-effective than running on a single type of virtual machine, and buying only the cheapest virtual machines for an application is not always economical.

ACKNOWLEDGEMENTS

It is my privilege to thank the people who either directly or indirectly helped me in making this thesis a success.

This thesis would not have been possible without my advisor, professor Dr. Olivia Das whose guidance and advice helped me in completing the thesis. She played a major role in assisting and guiding me through all these years. I would also like to thank the members of the defence committee, Dr. Bala Venkatesh, Dr. Alagan Anpalagan and Dr. Truman Yang, for spending their valuable time and energy reviewing my thesis and providing their humble responses which have been insightful.

I would also like to thank the anonymous reviewers who provided their valuable feedback on my paper, which has been a part of this research work. Furthermore, I would like thank Ryerson University for providing a great atmosphere and resources that helped to complete the thesis, and the financial support they have provided in the form of graduate awards, assistantship and fellowship during my graduate work.

I would finally like to acknowledge each and every member of my family for their tremendous support and hardships they have endured to help me complete my thesis and my friends who always stayed by my side when I was down. My parents' support in helping me pursue my graduate studies and creating a positive atmosphere around me helped me focus more on my research. My brother's sense of humor helped me cheer up in stressful situations and made me focus on the problem at-hand.

This thesis is dedicated to my amma and papa.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

VM	Virtual Machine
QoS	Quality of Service
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
ASP	Application Service Provider

Chapter 1

INTRODUCTION

1.1 Introduction

Cloud computing has been increasingly advancing as one of the major fields with its applications in various areas of business and personal use like storage, developing virtual circuits, running various applications and programs with its vast networking, operational and storage capabilities. Initially, through cloud storage data was made available anyplace around the globe. Storing information in the cloud and accessing it from any other place has made data management much easier and safe. Nowadays, cloud services have increased from storing data to accessing various servers virtually and running various applications through internet rather than connecting a server to the system or actually downloading the application into the system. This way of computing usually provides a better way to access a shared pool of resources for a better cost and on-demand.

Cloud services deployed through cloud computing are described by service models, which may be a combination of Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a

Service (SaaS). All these services are differentiated based on the various tasks offered by the cloud. These services mostly follow a hierarchy and may be interdependent [12]. **IaaS** model is the lowest-level in the hierarchy and provides necessary infrastructure to the clients. It is followed by the **PaaS** model where necessary application platform is delivered to the client and is dependent on IaaS for its resources. **SaaS** model is the final layout available to the client and gives the client a complete access over all its applications. The figure below shows a clear description of the various service models.

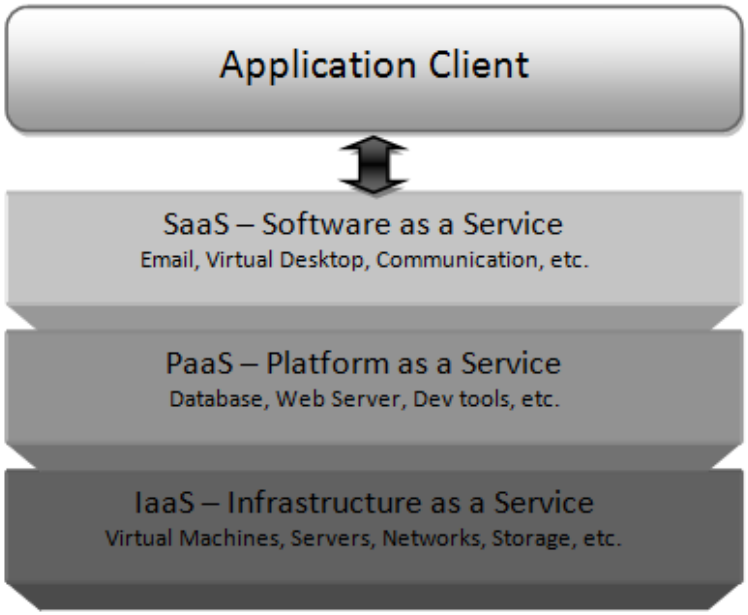

Figure 1. Cloud Service Models

When the service reaches the SaaS model, the application service provider (ASP) has to perform necessary checks on the system such that the system has better Quality of Service (QoS) to provide a better experience to the client.

There are different types of resources in cloud computing based on QoS with different functions and attributes. Some of the key attributes are service time, service cost, availability and reliability with which the service is offered. There are also different types of attributes based on the system requirement.

For example, storage can be considered an attribute when using storage services where storage capacity is a main concern. Managing these attributes accordingly helps in providing the user with a better system experience.

This can be achieved in two ways: High performance computing and high availability computing. As the names suggest, high performance computing is used to make sure the system performance is at its best. High availability computing is used to make sure the system's availability is higher or the system can be reliable [7].

1.2 What is Availability?

The term reliability means, the ability of a system to perform a required function, under given environmental and operational conditions and for a stated period of time[13]. Reliability can also be defined as the probability of a system in which no failure occurs in a give time period (0,*t*). Instantaneous availability or point availability denoted as *A(t),* can be defined as the probability that the system is working at an instant *t*, regardless of the number of times the system might have failed and been repaired in the time interval (0,*t*)[30].

At an initial instant, the availability is 1 but later gradually decreases until it tends to a constant limiting value known as steady-state availability. In a non-repairable system, instantaneous availability A(t) is equivalent to reliability R(t). However, if the system is repairable, availability can be expressed as:

$$A(t) = R(t) + \int_0^t R(t-x)m(x)\,dx \tag{1.1}$$

At any time t, availability is always greater than or equal to reliability. As it is already stated that *R(t)* represents reliability of a system in a time period of (0,t), *m(x) dx* is the

probability that a renewal occurs at an instant *x*. *R(t - x)* represents the probability that the system works in the time interval (0, t-x).

In general, steady-state availability of s component can be defined as

$$A = \frac{MTTF}{MTTF + MTTR} \tag{1.2}$$

When it comes to formulating steady-state availability, it does not apply for general system with internal redundancy, which means it only works for a system with a single UP state and a single DOWN state[30]. The steady-state availability equation is also given as:

$$\lim_{t \to \infty} A(t) = \frac{\mu}{\lambda + \mu} \tag{1.3}$$

The over-all steady-state availability of a system can be calculated by modeling the system as an interconnection of different components in series or parallel.

1.3 Research Problem

Cloud computing is the result of the adaptation of existing technologies and their evolution into a platform that allows clients to take advantage of all the technologies with low cost and task centered focus rather than worrying about different parameters and obstacles. The main ideology behind cloud computing is virtualization. It differentiates physical devices to virtual devices that are used in cloud computing to perform different tasks.

There are several researches that discuss the use of cloud computing as a platform to reduce system complexity and cost by opting virtual systems instead of physical systems. Further studies have also explored the allocation of VMs from different levels of cloud based on performance. In [4], Anand et. al. have given a solution to reduce VM migration based on performance SLAs which reduced the

overall migration of resources to a minimum by using integer programming. Authors in [15] consider the minimization of power and migration cost through VM placement and have proposed an algorithm that is based on dynamic programming and convex optimization method.

Availability constraints have been explored by Menascé et. al. in [10] which propose a near optimal solution to allocation of VMs based on high availability in the IaaS layer which has also produced an increase in revenue. A dynamic approach is proposed to evaluate reliability in clouds in [13] based on fault tree analysis which helps in studying system reliability and also portrays the impact of reliability on the cloud system.

Availability is one of the key features of cloud computing. Imagine a system where the servers go down in the middle of a project. This would affect the user experience and in turn leads to loss of customer usage. Therefore, maintaining high availability is very important in cloud computing.

One of the most important traits of cloud computing is the ability to allocate virtual machines to perform several tasks remotely from another location. These VMs are mostly allocated based on different properties such as system utilization, machine performance, system availability, system response time, system cost, memory and so on. Most of these properties are declared initially at the IaaS level and some are developed during run time. In this thesis, only availability and cost are considered from the ASP perspective to make the system work in an optimal condition such that the total system has high availability with a relatively low cost.

There has been development of various techniques based on deployment of virtual machines in cloud on the SaaS layer based on high performance computing. This thesis covers allocation of VMs in the SaaS layer where the Application Service Providers (ASPs) process operations of the system and optimize the number of VMs to be allocated to obtain a better result using high availability computing.

Initially, to run any computing application clients need to purchase necessary servers and components for the tasks to work for which, in the long run leads to loss of resources and also high cost. Using cloud computing however helps to virtually allocate machines through the network to produce necessary setup for a required task producing a consistent output. Making these resources available for a pay-per-use basis, monthly and yearly terms depending on the usage of resources reduces cost imparted on the client.

The main concept of cloud computing arises from the idea of replacing physical machines with virtually allocated machines from another location through an established network. The introduction of VMs into the system facilitates easy allocation and de-allocation of physical machines serving the system to meet necessary Quality of Service (QoS) objectives.

1.4 Research Overview

This thesis studies the effect of cost and availability of the virtual machines to acquire virtual machines that can be allocated to different software servers based on their availability constraints. An optimized model is developed that is run on an ILOG CP solver using AMPL. The steady state availability of a three – tier system with each tier using different types of VMs in different situations are studied, and the resulting solution that produces most cost-efficient output with minimum number of VMs dispatched to solve the problem is selected.

A hypothetical three – tiered cloud application is analyzed in this thesis. The application consists of three software servers: Web server, App server and Database server. Multiple copies of each server are assumed to be running in each tier respectively. An optimization problem is formulated where the goal was to minimize cost subject to SLA constraints. Availability is used as a constraint and a model is generated to test the system in different situations. Two different cases are considered depending on the

number of working copies in each tier. Case – 1 is considered as a simple system with only copy working for each server. Case – 2 describes multiple working copies in each tier.

Along the thesis, different conditions are described for both the cases and the outputs displayed help the ASPs make their decisions on acquiring VMs. This thesis also shows that allocation of cheapest VMs to reduce system cost is not always the best method to reduce the system cost and it is best preferred to run the servers on different types of VM instances rather than a fixed type of VMs.

1.5 Related Works

For the last couple of years, the growth of cloud computing has demanded a lot of research on the problems faced by cloud providers based on resource allocation. Although there have been many works on resource allocation, very little has been done from the application service providers (ASPs) perspective.

Authors in [8] provide a survey on a cloud system which uses QoS modeling and several other early QoS management systems. In [6], a research survey has been done on management of clouds based on resource allocation and different challenges faced by the cloud providers to achieve it. The main problem faced in resource allocation is to decide the optimal way to allocate virtual machines to physical machines. There have been many different methods discussed to formulate this problem by linking it to response time, availability, cost and power constraints. The most common objective is by increasing the revenue of the cloud provider. Authors in [25,29] present different solutions to this NP hard problem based on bin-packaging formulation. However, scaling problems of bin-packing have called for heuristic solutions [10, 15, 16, 17, 22, 34].

Authors in [10] discuss about an optimal allocation of cloud computing resources by proposing a near optimal (NOPT) algorithm based on hill-climbing approach and the results are compared with best

fit strategy. Through this comparison it is observed that the NOPT approach gives 45% better revenue compared to the best fit strategy and simultaneously maintaining an availability value very close to 1.

In [24], the authors have presented algorithms for minimizing the infrastructure cost based on response time constraints. This requires converting the customer SLA requirements to infrastructure level parameters. In [39], the authors propose evolutionary algorithms to minimize resource usage which improves execution time. Authors in [38] consider minimizing resources used by clustering components such that resource and communication requirements are not violated. In [40], authors discussed on a resource management system to allocate data dynamically based on user demand. It mainly focuses on minimizing cost by optimizing number of servers used.

Menascé et al. have presented in their recent work [2] two heuristic techniques that were used by ASPs to determine the number of VMs to be leased for minimizing system cost subject to response time constraints. In [32], the authors present an automated smartscale scaling framework which uses a combination of vertical and horizontal scaling. It ensures the application converges to the desired level. A heuristic algorithm has been proposed in [25] based on force-directed search where the upper bound profit is calculated and is compared with the proposed resource consolidation technique.

In [3], the authors proposed a nash bargaining approach that discusses about a cost effective and dynamic VM resource allocation method for handling media services in the cloud platform. The main focus is on challenging the issue which is to reduce the overall cost of running servers while making sure that the resources are being utilized at their maximum potential and the system completes the given job by the deadline. Several experiments have been conducted in the paper and the results show that the bargaining algorithm improves the resource utilization over time, with lesser VM migration overhead and active servers.

Mao and Humphrey defined an auto-scaling mechanism in [26] that guarantees execution of all the requests in given time. The main goal is to allocate resources that are only needed. It presents an

approach where the elements of the cloud, each of different costs and sizes, user specified performance metrics for the jobs are specified and the aim is to make sure all the jobs are done within the pre-fixed deadlines for a minimum cost. This is done by dynamic allocation/deallocation of VMs and scheduling tasks on cost-effective instances on various workload patterns which show increased savings from 9.8%-40.4% when compared to other approaches.

None of the above papers discussed consider availability constraint from the ASP perspective. Although availability is considered as a constraint, they were mainly focused on the cloud provider. The papers that have worked on the application provider considered response time or performance as their constraints. As a part of the thesis, [28] has been developed and it helps in bridging the gap created in the ASP level. It provides an optimization model to determine the number of VMs needed of each type for each server that helps minimize the cost and at the same time guarantees the availability SLA. It uses an ILOGCP solver to solve the model and helps the ASPs in taking decisions as to selecting the necessary amount and types of VMs.

1.6 Contributions

Following are different contributions of this thesis:

- The model that is developed to meet the optimal requirements for the system reduces the number of virtual machines that are allocated to the servers which assists in minimizing allocation of unexploited VMs.
- Allocation of less number of VMs results in minimization of overall cost of the system.
- The optimized model also focuses on reducing the cost of the system by comparing the cost of each type of VM and its availability together to comprehend the kind of VM that is better suited for the job.

1.7 Thesis Outline

This thesis layout is arranged as follows: Chapter 2 discusses the background study on Cloud computing, cloud computing in SaaS layer, system availability and allocation of virtual machines in the cloud. Chapter 3 discusses the system model, availability of the system and distribution of virtual machines to different layers. In Chapter 4, different cases involved in distribution of virtual machines have been clearly elaborated. The framework and problem analysis are discussed in Chapter 5 and the results obtained have been displayed. Chapter 6 presents the future work and concludes the thesis.

Chapter 2

BACKGROUND

Cloud computing is a popular term that is being used in various contexts in almost every industry for a simple data transfer to performing multiple high – level operations by the help of virtual resources for a comparatively cheaper cost on a pay-per-use basis. This chapter discusses about the structure of the cloud, allocation of resources and the effect of system availability on the cloud.

2.1 Cloud Computing

Cloud computing is an internet-based service that delivers necessary facilities to the user. Its operations can vary from simple mathematical calculations to running business operations on its agile architecture [19].

The key features of cloud computing include:

- Service oriented architecture
- Greater flexibility
- Low cost
- Offer services on-demand

Cloud computing removes the need to spend money on expensive hardware that have limited operating capabilities and also works only for a certain period of time.

2.1.1 History of Cloud Computing

Although the implementation of cloud computing started in the early 2000's, the initial formulation of the concept was by John McCarthy in the year 1960. Since its formulation, it has been evolving gradually both in its operations and the fields of usage. According to IBM[20], it began in 1950 through mainframe computing where multiple users used to access the mainframe computers through dumb terminals. In around 1970, the concept of virtual machines (VMs) has surfaced and led to the creation of virtualization software like VMware which made it possible to execute several operating systems simultaneously in an isolated environment. From the 1990s, virtualized private network connections were offered by different telecommunication companies. Since then, there has been huge research and development on Infrastructure as a service (IaaS) and Platform as a service (PaaS) but there has been little research in Software as a service (SaaS). It began as a full-time sharing solution on different platforms like Multics, Cambridge CTSS, and the early UNIX ports.

2.1.2 Characteristics of Cloud Computing

The use of cloud computing has become prominent due to its key characteristics. The following are the characteristics of cloud computing [35]:

Resource Sharing – Users can access a pool of multiple resources with a variety of physical and virtual resources with different configurations dynamically based the user's needs.

Internet Access – Users can access the services provided by the cloud at any location through internet. The network provides access to the resources through the protocols supporting various client platforms.

Reduced Cost – Resources in the cloud can be accessed by the users on a pay-per-use basis. This reduces the user's need to buy all the necessary hardware. Instead, the user can just access the required resources virtually and pay for them based on the deployment of those resources.

Scalability – One of the main advantages of cloud computing is on demand scaling. Resources are evenly – distributed and thus allowing resource utilization to spread evenly among the servers available.

Transparency – The resources utilized by the user can be measured, monitored and reported by the cloud provider providing transparency between the consumer and provider.

2.1.3 Classification of Cloud Computing

A basic cloud computing model can be classified into three different layers [31]:

1. Infrastructure as a Service
2. Platform as a Service

3. Software as a Service

- ***Infrastructure as a Service (IaaS)***

IaaS manages the hardware resources of the cloud which includes servers, routers, cables and switches. It is the lowest layer in the cloud. It offers a pool of resources through a virtual datacenter that connects multiple virtual machines to a single physical server. Some of the leading vendors that rely on IaaS layer are: Amazon's product EC2, Amazon S3, Flexiscale and Rackspace Cloud Servers.

- ***Platform as a Service (PaaS)***

The second layer in the cloud is Platform. It deploys the environment for the resources which includes APIs, frameworks and databases. It provides the infrastructure required through internet, the environment best suited based on the architecture generated by the first layer. The main vendors that use this layer are Microsoft Windows Azure and Google App Engine.

- ***Software as a Service (SaaS)***

The final layer in cloud computing is Software. It delivers necessary software and data that are hosted in the internet. It can be accessed through a thin client by the users. The users pay for the service on a monthly basis based on usage.

2.1.4 Types of Clouds

Cloud has emerged as a convergence of multiple computing trends. Based on the physical location and service distribution clouds can be classified into four different types which are mentioned below [34].

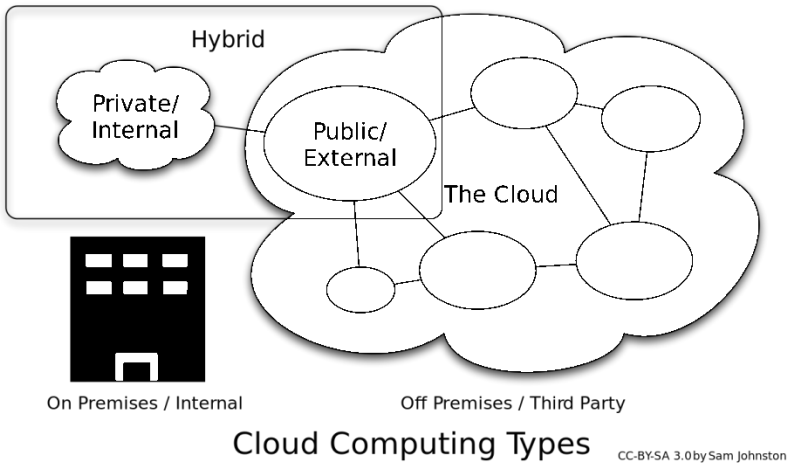

Figure 2. Types of Cloud computing

- ***Private Cloud***

Private cloud is restricted to an organization. The services provided by a private cloud are only available for the people in that particular organization or third party vendors. Private clouds can exist inside the premises or outside but the services cannot be accessed by the public.

- ***Public Cloud***

Some organizations offer their cloud services on pay-per-use basis to general public. Various businesses adopt public clouds to save the hardware or software cost. Public clouds are mostly

used for development, deployment and management of enterprise applications for a reasonable cost. The major drawback for public clouds is its open access to anyone paying for the service as it exposes the system to imminent threats compromising security. Therefore a proper validation is required to access the public cloud.

- ***Community Cloud***

Community cloud is similar to public cloud except that it can be accessed only by a particular community of users. Its infrastructure can be present on premises or located somewhere else at a third party organization. It can be accessed by the users of the same community who have similar concerns like privacy requirements, policy, and security concerns [34].

- ***Hybrid Cloud***

A hybrid cloud is an integration of two or more clouds: private, public and community cloud. It helps the users to access the secure applications of private cloud, while also allowing shared data access and cost benefit of the public cloud.

2.2 Research in SaaS Layer

There has been a wide range of researches on the IaaS and PaaS layers. But when it comes to SaaS, very limited research has been done. The cloud infrastructure is leased by the SaaS providers by instantiating VMs that are compatible and that are much suited for the consumer. The user pays for this service on a pay-per-use basis based on the service provided. This states that the SaaS provider has to be

able to determine the total number of VMs required for the service to run perfectly, assuring that the Quality of Service (QoS) is not compromised and the total amount spent on the VMs is minimum.

In [1, 2], Aldhalaan and Menascé have proposed two techniques that best satisfy the consumer demands. In this paper, ScaleUpDown Algorithm and FillSlotsFirst Algorithm are the two algorithms presented that can be used by the application service providers to determine the type and quantity of VMs to be leased in order to satisfy the customer demands. It considers minimizing the cost and response time constraints of the VMs. L. Wu, S. K. Garg and R. Buyya[24] have also worked on the SaaS layer and proposed a different algorithm that also minimizes response time constraints while maintaining a minimum cost of infrastructure. Their algorithm involves translating customer SLA requirements into infrastructure level parameters.

[26] produced algorithms that reduce resource usage and improves execution time. Evolutionary algorithm approach is used to face the problem.

2.3 Quality-of-Service

Cloud computing allows the access of different computing resources like servers, working platforms, networks, storage spaces and applications. All these resources that are accessed through the cloud network are called services[14]. These services are easily managed and are provided to the users on demand. This means the user pays the provider for these services on a pay-per-use basis. These services are offered depending on certain established agreements between the provider and the client called Service-Level Agreements (SLAs). These SLAs specify certain values of system availability, response time and other QoS parameters (or metrics) that the user and provider agree upon. These parameters are therefore monitored continuously and the users are notified if there is any service disruption in case of QoS degradation or when the services become unavailable, or to make sure the cloud provider does not deviate from the QoS statements mentioned in the SLA.

Quality-of-Service (QoS) is the amount of reliability, availability or performance that is provided by the infrastructure or platform that hosts it. The users refer to QoS as a parameter to select a cloud. There have been several researches on QoS management to make sure the cloud resources produce a high performance system to satisfy the user[27]. However, when it comes to availability, there hasn't been much research to study its effects on the cloud system.

2.3.1 SLAs in Cloud

Cloud computing does not offer proper control over services provided and this makes the customer to take necessary precautions to counter loss of QoS. Therefore, SLAs have become a part of cloud computing and most customers select a cloud provider based on their SLA proposals. Service Level Agreements (SLAs) are the binding agreements signed between the customer and cloud service provider to specify the level of service to be delivered as well the steps to how measuring, reporting and handling of SLA violation should be done [5]. SLA mainly focuses on the dependability of the system and specifies the rate for a particular time period, like a month or a year. For example, Amazon EC2 SLA mentions its system dependability on an annual basis and calls it as "Annual Uptime Percentage". The company states its Annual Uptime Percentage to be 99.99% and offers a 10% service credits if it deviates from the value. Similarly, different cloud providers like Microsoft, Google, Rackspace offer different percentage values and also provide different amount of credits to the customer when the servers go below the value. These QoS values presented in the SLA proposals are specified in Service Level Specification (SLS).

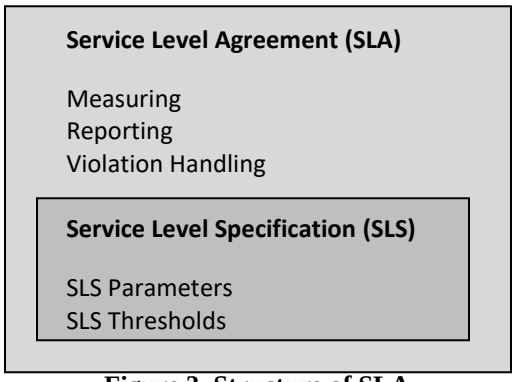

Figure 3. Structure of SLA

When working on a cloud, it is observed that offering only availability parameters in SLA is not enough to ensure better service delivery to the customer. For example, if a system has an availability of 99.99% and if its performance levels are too low or if the run time is too high it is not considered an ideal choice by the customer. The cloud infrastructure has to keep changing based on the user demands and has to automatically allocate resources based on the SLA requirements. At the same time it should also detect violations and act accordingly to avoid paying credits to the customer.

Although the system overcomes different SLA requirements, it still face a couple of challenges such as:

- Allocation of resources based on SLA requirements
- Measurement and monitoring of system for violations
- Acting accordingly based on the observed violations

If any violations are observed for availability, the system might have to add more resources in standby to handle the failures. For performance violations, if the system is getting overloaded the best case would be allocating VMs to other physical machines to reduce system overload.

2.4 Availability based Quality-of-Service

Availability of a system is the probability that the system can complete a required task in a given period. To achieve high availability means the system should not fail at any point. Cloud providers use different cloud resources to supply necessary services to customers. Various conventional studies made on availability in the past are not particularly suited for the more dynamic and distributed field of cloud computing. Originally, QoS does not consider availability as a resource in cloud computing. An availability oriented QoS model produced in [36] that helps monitor availability of the cloud.

There are different attributes to cloud computing resources. The most common attributes would be service time, cost of service, service credit rating and reliability of service offered by the cloud.

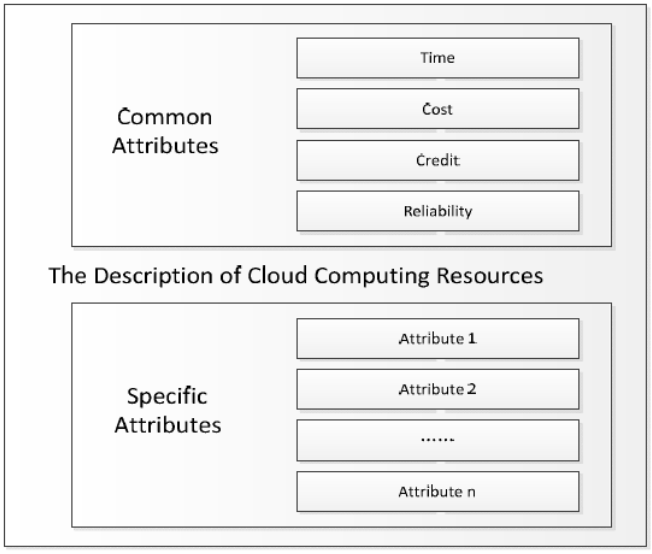

Figure 4. Resource Attributes Model[36]

To determine the quality of service of a cloud system, several modeling techniques[8] can be used such as queuing models, Petri nets, reliability block diagrams and so on. All these models are classified based on performance, dependability, black-box services, and simulation.

2.4.1 Availability of a component

In a cloud, availability refers to the amount of time the system is up and running without any drop in its services. If the system goes off when there is any task running on the cloud it affects the user's experience and the customer would be inclined to opt for another provider. Availability of a system is a factor of reliability. In other words, if the reliability increases, availability increases [18].

Availability can be represented as a ratio of expected system uptime to the sum of expected uptime and downtime (or) it is the measure of readiness of the system.

$$A = \frac{UpTime}{UpTime + DownTime} \tag{2.1}$$

(or)

$$A = \frac{MTTF}{MTTF + MTTR} \tag{2.2}$$

where,

MTTF is the mean time to failure of the system

MTTR is the mean time to repair

Many methods and analysis models are developed to calculate availability such as Fault tree Analysis, Reliability block diagrams, Markov Chains, Petri Nets and so on.

2.4.2 System Availability

Availability of a system can be evaluated by modeling the entire system as a group of series and parallel components. A system is said to be in series if one failed component in the system results to the entire system failing. A system is said to be in parallel if when using a failed component the operations of that particular component are being taken over by the other component that is working (or the system fails only when all the components fail) [18]. Consider a 2-component system where A_x is the availability of component x and A_y is the availability of component y.

The combined availability for a 2-component system in series is:

$$A = A_x A_y \tag{2.3}$$

The combined availability for a 2-component system in parallel is:

$$A = 1 - (1 - A_x)(1 - A_y) \tag{2.4}$$

2.4.3 Availability in Cloud Computing

Maintaining system availability is one of the major issues in cloud computing [33]. If the user wants to access the system during a heavy load situation, the system cannot be available as there is a list of operations that are yet to be completed and no available resources for the system to become ready to be used by the user. To achieve system availability, the system and its

resources have to be robust and highly reliable so that they offer maximum failure resistance and in turn provide maximum availability. Also, the ASPs have to make sure that there is no hardware failure and all the systems are well maintained. Sometimes, user errors also lead to loss in availability.

Availability of the system can be increased by minimizing human errors. Also upgrading a system when it is running other processes affects the availability. So such tasks have to be done when the system is in an idle state. Regular system maintenance and selecting robust machines also increase availability.

The main problem faced by the application service providers (ASPs) is to determine minimum number of VMs and servers to the system such that the SLAs are not violated. The major problem in cloud computing systems is, efficiently managing VM resources based on QoS requirement. This problem is discussed in [3] which proposes a nash-bargaining approach. A similar problem is discussed in [24]. This approach is done by studying the optimization problem that considers a system with minimum cost and assigning a fixed availability value to the system.

Chapter 3

DISTRIBUTION OF VIRTUAL MACHINES

3.1 Three – Tier Cloud System

A cloud computing system is an internet-based computing system that provides shared resourcing pool of processors and data to different devices on demand. A regular cloud computing application has three different layers: Web server, Application server and Database server.

3.1.1 Architecture

Consider the following layout for the system to be discussed. The application consists of three servers: Web Server, App Server and DB server. Each tier can have one or more copies running. Let E1, E2 and E3 be the tiers with multiple copies each having an individual virtual machine VM1, VM2 and VM3.

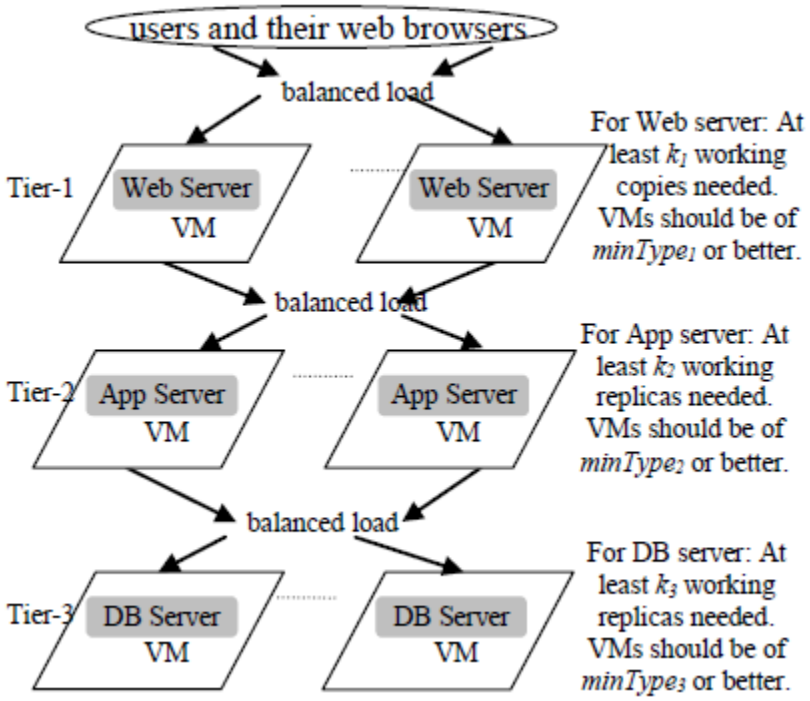

Figure 5. System Layout

For the above system to work, it is assumed that all the tiers E1, E2 and E3 have to work successfully. This approach shows that the top tiers depend on bottom tiers. So for tier - 1 to work both tier - 2 and tier - 3 need to be up. Also, all the copies in a tier and their respective VMs are assumed to be identical. Since the process uses a failure state logic, the layout actually gives the failure state of the system. Hence, if $\bar{A}_{sys}$ is the output unreliability of the system, A_{sys} is the final steady state availability acquired.

$$A_{sys} = 1 - \bar{A}_{sys} \quad (3.1)$$

This unavailability value is substituted to the optimization model for the final output conditions and the problem is solved. Before proceeding to solving the actual problem some assumptions are also included to provide a simpler solution to the model.

3.1.2 Assumptions

For the system to work, different assumptions are made to support the model to be discussed. These assumptions considered are as follows:

- The system works only if all the tiers in the system are working.
- Each tier has different number of layers or copies.
- Each tier is associated with its own virtual machine.
- All the layers in a particular tier are identical in their properties.
- All the layers of a virtual machine in a particular tier are identical in their properties.

3.1.3 Notations

The following are the list of different notations used in the paper.

- N : Total number of VM types provided by the cloud provider ($1 \leq j \leq N, j \in \mathbb{N}$).
- M : Number of tiers available in the system ($1 \leq i \leq M, i \in \mathbb{N}$).
- A_j : Steady-state availability of VM of type j.
- $\bar{A}_j$: Steady-state unavailability of VM of type j.

$$A_j = 1 - \bar{A}_j$$

- C_j : Cost of an individual VM of type j.

- $X_j \geq 0$: Number of VMs of type j provided by the cloud provider to make the system work.
- $X_{(i,j)} \geq 0$: Number of VMs of type j provided by the cloud provider to tier i.
- k_i : Minimum number of working replicas needed for server i for the system to be up.
- $\mathbb{C}$: Total optimized amount to be paid to obtain the system with required steady-state availability.
- SSA_{req} : Steady-State Availability requirement of the system by the SLA.

3.2 Problem Description

Deployment and managing applications in a cloud is challenging as it requires determining and buying required number of VMs dynamically such that the application is available. The main problem faced by the ASPs is allocation of VMs in the SaaS layer considering availability SLAs.

3.2.1 VM Distribution

Allocation of VMs to run the system with a required SSA requires a calculated and optimized model to figure the number of VMs distributed across different layers. This model considers the required values of SSA and the condition of having a minimum amount of cost to be spent on the VMs. Although the overall optimized output is required to have better availability with seemingly affordable cost. This process of distribution of VMs is described in two cases.

Case – 1:

This case covers the condition of having the least of one VM running in each level to make the system to work.

Case – 2:

In this case, each level has a corresponding minimum number of VMs running to make the system to work. This case can be considered the parent case and case – 1 is the extension with minimum number of VMs for each level being unity.

3.2.2 Difference in Steady-State Availability between cases

As discussed above, the optimization has two different cases. Although a cumulative model has been obtained both cases have their identity of their own. A detailed classification of each case is described below:

Case – 1:

Consider a simple system that works when there is at least one replica from the available multiple layers of tasks running and each layer can run only one type of VM. The optimization model would be as follows:

minimize

$$\mathbb{C} = \sum_{j} (C_j * X_j) \tag{3.2}$$

s.t.

$$\prod_{j} \left(1 - \left(\bar{A_j}^{X_j}\right)\right) \geq SSA_{req} \tag{3.3}$$

where, C_j(j∈N) is the cost of VM of type j and X_j(j∈N) is the number of VM of type j provided for the system to work by the cloud provider. $\bar{A}_j$(j∈N) depicts the unavailability of the VM with type j.

The model is then run through the solver to obtain the number of VMs of each type to be dispatched by the cloud provider to get optimized steady-state availability for the entire system.

This model can be further extended to fit real-time scenarios using different type of VMs, each with different system availability, response time and cost of item etc. The following describes the optimization model for the system with a single working replica using any type of VMs.

minimize

$$\mathbb{C} = \sum_{i=1}^{M} \sum_{j=minType_i}^{N} (C_j * X_{(i,j)}) \tag{3.4}$$

s.t.

inequality 1:

$$\prod_{i}^{M}\left(1-\prod_{j=minType_i}^{N}\left(\bar{A_j}^{X_{(i,j)}}\right)\right)\geq SSA_{req} \tag{3.5}$$

inequality 2:

$$\sum_{j=minType_i}^{M} X_j \geq 1 \tag{3.6}$$

The system generates outputs that give different number of VMs for each tier and type required to provide a system with optimized availability and cost.

Case – 2:

This system discusses the concept of multiple working replicas. Working on a system that can only use a single replica working from a cluster is easy, but it also has a loss of resources. In this case, k number of replicas can be used or at least k replicas have to be working to make the system to complete the given task. The main advantage of this system is, it helps reduce processing time in case of large systems. The following is the optimized model for a k-out-of-n system.

minimize

$$\mathbb{C}=\sum_{i=1}^{M}\sum_{j=minType_i}^{N}\left(C_j * X_{(i,j)}\right) \tag{3.7}$$

s.t.

inequality 1:

$$\prod_{i=1}^{M}\left(\sum_{x_j=0}^{X_{(i,j)}}\prod_{j=minType_i}^{N}\binom{X_{(i,j)}}{x_j}A_j{}^{x_j}\bar{A}_j{}^{X_{(i,j)}-x_j}\right)\geq SSA_{req} \tag{3.8}$$

s.t,

$$\sum_{j=1}^{N}x_j \geq k_i \tag{3.9}$$

inequality 2:

$$\sum_{j=minType_i}^{N}X_{(i,j)} \geq k_i, \forall i \tag{3.10}$$

Equations (3.8) and (3.9) shows the probability that at least k_i replicas have to be working for server $i.x_j$ represents the number of working VMs of type j allocated for server i. The VM instances allocated to server i should be $minType_i$ or higher.

Equation (3.10) helps the system make sure that the total number of VMs of type j allocated to server i should always be greater or equal to k_i.

Chapter 4

ANALYISIS AND RESULTS

4.1 Problem Analysis

All the files are added to the database and the code is run through AMPL solver. Once the files are added to the database solver is opened and AMPL is selected by typing “ampl” in the command line. Then a new line with “ampl:” is opened where all the ampl codes can be run. The code is run by using include command.

After running the code, the solver takes a while to compare all the values that can be used to solve the problem and produces an optimized solution for the system. Different conditions are applied for the type of VMs to be used by adding necessary inequality conditions. These conditions are compared and graphical results are produced.

Case – 1:

As case – 1 discusses about using at least 1 VM working in all the tasks, different conditions are set using only 1 type of VM at a time. To set a particular type to be used, the other type inequalities are used. For example, if the client wants only type 1 VMs to be used, the inequalities 6, 7, 9, 10, 12, 13 are removed from the comments and used in the code. This states that all the 3 tiers would be using type 1 VMs for solving the problem at hand as the other 2 types are set to zero. This method can be applied to any combination and even all the types can be used however the system decides the best optimized solution and allocates it to the system.

Case – 2:

This case focuses on a system with at least k number of VMs running in order to make the system work. The same logic used in case -1 is used and different inequalities are set. But the major difference in both cases is, in case-2 the condition of number of VMs can be changed to any number from 1 to maximum number of VMs that can be made available.

4.2 Results

All the different conditions are compared with respect to each other. All results are recorded and a comparison histogram is created. Both case-1 and case-2 are evaluated for different conditions that are discussed in section-2.

4.2.1 Impact of using minimum type VMs on optimal solution and cost

In this paper, three tiers were used and each tier can use any of the three types of VMs that are allocated for the task. To denote what VMs are used for a particular solution, we use ***minType*** and ***fixType***. Since we have considered 3 tiers, ***minType*** is represented as ($minType_1$, $minType_2$, $minType_3$) and ***fixType*** is represented as ($fixType_1$, $fixType_2$, $fixType_3$), where $***Type_1$ denotes tier-1, $***Type_2$ denotes tier-2 and $***Type_3$ denotes tier-3. For example, ($minType_1$, $minType_2$, $minType_3$) = *(1, 2, 3)* (or simply denoted as *minType(1, 2, 3)*) denotes that tier 1 can use VMs from type 1 to maximum type available (i.e., type 3), tier 2 can use type 2 and type 3 VMs, and tier 3 can use type 3 VMs to run the task. ($fixType_1$, $fixType_2$, $fixType_3$) = *(1, 2, 3)* (simply denoted as *fixType(1, 2, 3)*) implies that tier 1 can use only type 1, tier 2 can use only type 2 and tier 3 can use only type 3 to run the given task.

In this thesis, we consider 10 different combinations for ($minType_1$, $minType_2$, $minType_3$): (1, 1, 1), (2, 2, 2), (3, 3, 3), (1, 2, 3), (1, 1, 2), (1, 2, 2), (1, 1, 3), (1, 3, 3), (2, 2, 3), and (2, 3, 3). All these combinations are run and their optimal solutions are recorded for both cases 1 and 2 for a fixed steady-state availability of 0.99999.

Case-1: In this case, application is available if at least one working copy of each of the three servers is available (i.e. $k_1 = k_2 = k_3 = 1$).

($minType_1$, $minType_2$, $minType_3$)	Number and type of VMs allocated to Web server. $[n_{1,\,type1}, n_{1,\,type2}, n_{1,\,type3}]$	Number and type of VMs allocated to Web server. $[n_{2,\,type1}, n_{2,\,type2}, n_{2,\,type3}]$	Number and type of VMs allocated to Web server. $[n_{3,\,type1}, n_{3,\,type2}, n_{3,\,type3}]$	Total Cost ℂ (in dollars)
(1, 1, 1)	[0, 1, 1]	[1, 0, 1]	[0, 1, 1]	804.46
(2, 2, 2)	[0, 1, 1]	[0, 1, 1]	[0, 1, 1]	808.11
(3, 3, 3)	[0, 0, 2]	[0, 0, 2]	[0, 0, 2]	871.62
(1, 2, 3)	[1, 0, 1]	[0, 1, 1]	[0, 0, 2]	825.63
(1, 1, 2)	[1, 0, 1]	[0, 1, 1]	[0, 1, 1]	804.46
(1, 2, 2)	[1, 0, 1]	[0, 1, 1]	[0, 1, 1]	804.46
(1, 1, 3)	[0, 1, 1]	[1, 0, 1]	[0, 0, 2]	825.63
(1, 3, 3)	[1, 0, 1]	[0, 0, 2]	[0, 0, 2]	846.8
(2, 2, 3)	[0, 1, 1]	[0, 1, 1]	[0, 0, 2]	829.28
(2, 3, 3)	[0, 1, 1]	[0, 0, 2]	[0, 0, 2]	850.45

Table 1. *minType* allocation of VMs to the servers and total cost (where $k_1=k_2=k_3=1$) with $SSA_{req} = 0.99999$

From the above table, it is observed that for different combinations of ($minType_1$, $minType_2$, $minType_3$), the allocation of VMs is different. For the combination of (3, 3, 3) the overall cost is maximum (in this case $871. 62) and the overall minimum cost is 804.46.

Case – 2: In this case, the application is available if at least k_i copies are working for server i (i.e. $k_1 = k_2 = k_3 = 2$).

($minType_1$, $minType_2$, $minType_3$)	Number and type of VMs allocated to Web server. $[n_{1,\,type1}, n_{1,\,type2}, n_{1,\,type3}]$	Number and type of VMs allocated to Web server. $[n_{2,\,type1}, n_{2,\,type2}, n_{2,\,type3}]$	Number and type of VMs allocated to Web server. $[n_{3,\,type1}, n_{3,\,type2}, n_{3,\,type3}]$	Total Cost ℂ (in dollars)
(1, 1, 1)	[0, 0, 3]	[0, 0, 3]	[0, 1, 2]	1286.26
(2, 2, 2)	[0, 0, 3]	[0, 0, 3]	[0, 1, 2]	1286.26
(3, 3, 3)	[0, 0, 3]	[0, 0, 3]	[0, 0, 3]	1307.43
(1, 2, 3)	[0, 0, 3]	[0, 1, 2]	[0, 0, 3]	1286.26
(1, 1, 2)	[0, 0, 3]	[0, 1, 2]	[0, 0, 3]	1286.26
(1, 2, 2)	[0, 0, 3]	[0, 1, 2]	[0, 0, 3]	1286.26
(1, 1, 3)	[0, 1, 2]	[0, 0, 3]	[0, 0, 3]	1286.26
(1, 3, 3)	[0, 1, 2]	[0, 0, 3]	[0, 0, 3]	1286.26
(2, 2, 3)	[0, 1, 2]	[0, 0, 3]	[0, 0, 3]	1286.26
(2, 3, 3)	[0, 1, 2]	[0, 0, 3]	[0, 0, 3]	1286.26

Table 2. *minType* allocation of VMs to the servers and total cost(where $k_1 = k_2 = k_3 = 2$) with $SSA_{req} = 0.99999$

From the table 3, similar to case - 1 maximum cost occurs in the combination (3, 3, 3). However, in all the other cases due to a high availability condition, there exists only a fixed combination for all the other combinations and the total cost recorded is similar.

4.2.2 Single type of VM instances vs. Different type of VM instances

As stated before, ***minType*** and ***fixType*** are used to find out the number of VMs required to be allocated to obtain the optimized cost. The results are formulated for ***fixType*** using the same above stated combinations and the optimal solutions are recorded for both case-1 and case-2. The steady-state availability remains fixed at 0.99999.

Case-1: The application is said to be available if at least one working copy of each of the three servers is available (i.e. $k_1 = k_2 = k_3 = 1$).

($fixType_1$ $fixType_2$, $fixType_3$)	Number and type of VMs allocated to Web server. $[n_{1,\ type1}, n_{1,\ type2}, n_{1,\ type3}]$	Number and type of VMs allocated to Web server. $[n_{2,\ type1}, n_{2,\ type2}, n_{2,\ type3}]$	Number and type of VMs allocated to Web server. $[n_{3,\ type1}, n_{3,\ type2}, n_{3,\ type3}]$	Total Cost ℂ (in dollars)
(1, 1, 1)	[3, 0, 0]	[3, 0, 0]	[3, 0, 0]	1084.05
(2, 2, 2)	[0, 3, 0]	[0, 3, 0]	[0, 3, 0]	1116.9
(3, 3, 3)	[0, 0, 2]	[0, 0, 2]	[0, 0, 2]	871.62
(1, 2, 3)	[3, 0, 0]	[3, 0, 0]	[0, 0, 2]	1024.19

(1, 1, 2)	[3, 0, 0]	[3, 0, 0]	[0, 3, 0]	1095
(1, 2, 2)	[3, 0, 0]	[0, 3, 0]	[0, 3, 0]	1105.95
(1, 1, 3)	[3, 0, 0]	[3, 0, 0]	[0, 0, 2]	1013.24
(1, 3, 3)	[3, 0, 0]	[0, 2, 0]	[0, 0, 2]	942.43
(2, 2, 3)	[0, 3, 0]	[0, 3, 0]	[0, 0, 2]	1035.14
(2, 3, 3)	[0, 3, 0]	[0, 0, 2]	[0, 0, 2]	953.38

Table 3. *fixType* allocation of VMs to the servers and total cost (where $k_1=k_2=k_3=1$) with $SSA_{req} = 0.99999$

The cheapest combination in this case occurs at (3, 3, 3) with the cost being $871.62 and the combination with highest cost is at (2, 2, 2) with a cost of 1116.9.

Case – 2: In this case, the application is available if at least k_i copies are working for server i (i.e. $k_1 = k_2 = k_3 = 2$).

($fixType_1$, $fixType_2$, $fixType_3$)	Number and type of VMs allocated to Web server. $[n_{1,\ type1},\ n_{1,\ type2},\ n_{1,\ type3}]$	Number and type of VMs allocated to Web server. $[n_{2,\ type1},\ n_{2,\ type2},\ n_{2,\ type3}]$	Number and type of VMs allocated to Web server. $[n_{3,\ type1},\ n_{3,\ type2},\ n_{3,\ type3}]$	Total Cost ℂ (in dollars)
(1, 1, 1)	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	-
(2, 2, 2)	[0, 4, 0]	[0, 4, 0]	[0, 4, 0]	1489.2
(3, 3, 3)	[0, 0, 3]	[0, 0, 3]	[0, 0, 3]	1307.43

(1, 2, 3)	[4, 0, 0]	[0, 4, 0]	[0, 0, 3]	1414.01
(1, 1, 2)	[4, 0, 0]	[4, 0, 0]	[0, 4, 0]	1460
(1, 2, 2)	[4, 0, 0]	[0, 4, 0]	[0, 4, 0]	1474.6
(1, 1, 3)	[4, 0, 0]	[4, 0, 0]	[0, 0, 3]	1399.41
(1, 3, 3)	[4, 0, 0]	[0, 0, 3]	[0, 0, 3]	1353.42
(2, 2, 3)	[0, 4, 0]	[0, 4, 0]	[0, 0, 3]	1428.61
(2, 3, 3)	[0, 4, 0]	[0, 0, 3]	[0, 0, 3]	1368.02

Table 4. *fixType* allocation of VMs to the servers and total cost(where $k_1 = k_2 = k_3 = 2$) with $SSA_{req} = 0.99999$

In case-2, the costliest model is at (2, 2, 2) with $1489.2 and the cheapest is at (3, 3, 3) which is $1307.43. At (1, 1, 1) however, there is no possible allocation that would satisfy the optimization model and hence the condition does not produce any results.

A comparison is made between both strategies (***minType*** and ***fixType***) for both the cases based on the outputs noted in tables 1 to 4. A fixed combination (*type(1, 2, 3)* (which represents the configuration (1, 2, 3))) is taken to study the difference between the both strategies clearly. Table 5 and table 6 show the difference in both strategies for case – 1 and case – 2 respectively and the number of VMs dispatched to run the task.

Strategy	Number and type of VMs allocated to Web server. $[n_{1,\ type1},\ n_{1,\ type2},\ n_{1,\ type3}]$	Number and type of VMs allocated to App server. $[n_{2,\ type1},\ n_{2,\ type2},\ n_{2,\ type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\ type1},\ n_{3,\ type2},\ n_{3,\ type3}]$	Total cost ℂ (in dollars)
minType	[1, 0, 1]	[0, 1, 1]	[0, 0, 2]	825.63
fixType	[3, 0, 0]	[3, 0, 0]	[0, 0, 2]	1024.19

Table 5. A Comparison of *minType(1, 2, 3)* and *fixType(1, 2, 3)* strategies under case-1

Strategy	Number and type of VMs allocated to Web server. $[n_{1,\ type1},\ n_{1,\ type2},\ n_{1,\ type3}]$	Number and type of VMs allocated to App server. $[n_{2,\ type1},\ n_{2,\ type2},\ n_{2,\ type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\ type1},\ n_{3,\ type2},\ n_{3,\ type3}]$	Total cost ℂ (in dollars)
minType	[0, 0, 3]	[0, 1, 2]	[0, 0, 3]	1286.26
fixType	[4, 0, 0]	[0, 4, 0]	[0, 0, 3]	1414.01

Table 6. A Comparison of *minType(1, 2, 3)* and *fixType(1, 2, 3)* strategies under case-2

By observing the above tables, it is observed that when compared to ***minType*** and ***fixType*** the results for ***minType*** are cheaper. This cost variance shows that in comparison ***minType*** offers cheaper VMs for the same steady-state availability. Therefore, ***minType*** strategy is considered a better pick.

To further study the effect of availability, both ***minType*** and ***fixType*** are compared for different values of steady- state availability for a fixed combination (*type(1, 2, 3)*) and results are

noted. Table – 7 and table – 8 note the values of case – 1 and case – 2 respectively for a ***minType*** combination. Table – 9 and table – 10 show the values of case – 1 and case – 2 respectively for a ***fixType*** combination.

SSA_{req}	Number and type of VMs allocated to Web server. $[n_{1,\ type1},\ n_{1,\ type2},\ n_{1,\ type3}]$	Number and type of VMs allocated to App server. $[n_{2,\ type1},\ n_{2,\ type2},\ n_{2,\ type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\ type1},\ n_{3,\ type2},\ n_{3,\ type3}]$	Total cost ℂ (in dollars)
0.99999	[1, 0, 1]	[0, 1, 1]	[0, 0, 2]	825.63
0.9999	[1, 1, 0]	[0, 2, 0]	[0, 0, 2]	783.29
0.999	[0, 0, 2]	[0, 0, 1]	[0, 0, 1]	581.08
0.995	[0, 0, 1]	[0, 0, 1]	[0, 0, 1]	435.81
0.9	[1, 0, 0]	[1, 0, 0]	[0, 0, 1]	389.82
0.8	[1, 0, 0]	[0, 1, 0]	[0, 0, 1]	389.82
0.7	[1, 0, 0]	[0, 1, 0]	[0, 0, 1]	389.82

Table 7. Total cost and VMs allocated for different SSA_{reqs} through case-1 based on *minType*

SSA_{req}	Number and type of VMs allocated to Web server. $[n_{1,\ type1},\ n_{1,\ type2},\ n_{1,\ type3}]$	Number and type of VMs allocated to App server. $[n_{2,\ type1},\ n_{2,\ type2},\ n_{2,\ type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\ type1},\ n_{3,\ type2},\ n_{3,\ type3}]$	Total cost ℂ (in dollars)
0.99999	[0, 0, 3]	[0, 1, 2]	[0, 0, 3]	1286.26
0.9999	[1, 0, 2]	[0, 3, 0]	[0, 0, 3]	1219.1
0.999	[3, 0, 0]	[0, 3, 0]	[0, 0, 3]	1169.46
0.995	[0, 0, 2]	[0, 0, 2]	[0, 0, 2]	871.62
0.9	[2, 0, 0]	[0, 2, 0]	[0, 0, 2]	779.64
0.8	[2, 0, 0]	[0, 2, 0]	[0, 0, 2]	779.64
0.7	[2, 0, 0]	[0, 2, 0]	[0, 0, 2]	779.64

Table 8. Total cost and VMs allocated for different SSA_{reqs} through case-2 based on *minType*

SSA_{req}	Number and type of VMs allocated to Web server. $[n_{1,\ type1},\ n_{1,\ type2},\ n_{1,\ type3}]$	Number and type of VMs allocated to App server. $[n_{2,\ type1},\ n_{2,\ type2},\ n_{2,\ type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\ type1},\ n_{3,\ type2},\ n_{3,\ type3}]$	Total cost ℂ (in dollars)
0.99999	[3, 0, 0]	[3, 0, 0]	[0, 0, 2]	1024.19
0.9999	[3, 0, 0]	[0, 2, 0]	[0, 0, 2]	900.09
0.999	[2, 0, 0]	[0, 2, 0]	[0, 0, 1]	634.37
0.995	[2, 0, 0]	[0, 2, 0]	[0, 0, 1]	634.37
0.9	[1, 0, 0]	[1, 0, 0]	[0, 0, 1]	389.82

0.8	[1, 0, 0]	[0, 1, 0]	[0, 0, 1]	389.82
0.7	[1, 0, 0]	[0, 1, 0]	[0, 0, 1]	389.82

Table 9. Total cost and VMs allocated for different SSA_{reqs} through case-1 based on *fixType*

SSA_{req}	Number and type of VMs allocated to Web server. $[n_{1,\ type1}, n_{1,\ type2}, n_{1,\ type3}]$	Number and type of VMs allocated to App server. $[n_{2,\ type1}, n_{2,\ type2}, n_{2,\ type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\ type1}, n_{3,\ type2}, n_{3,\ type3}]$	Total cost ℂ (in dollars)
0.99999	[4, 0, 0]	[0, 4, 0]	[0, 0, 3]	1414.01
0.9999	[4, 0, 0]	[0, 3, 0]	[0, 0, 3]	1289.91
0.999	[3, 0, 0]	[0, 3, 0]	[0, 0, 3]	1169.46
0.995	[0, 0, 2]	[0, 0, 2]	[0, 0, 2]	871.62
0.9	[2, 0, 0]	[0, 2, 0]	[0, 0, 2]	779.64
0.8	[2, 0, 0]	[0, 2, 0]	[0, 0, 2]	779.64
0.7	[2, 0, 0]	[0, 2, 0]	[0, 0, 2]	779.64

Table 10. Total cost and VMs allocated for different SSA_{reqs} through case-2 based on *fixType*

At any instant (say when using *type(1, 2, 3)* configuration), it is observed that using a fixed type of VM may reduce using multiple resources (VMs) to only a single type of VM but the overall cost is increased. However, allowing the use of different types of VMs provides similar results for a lesser cost. This shows that using different types of VMs is better compared to a fixed type.

There is another variable which changes the output of the system, which is steady-state availability. Having a higher SSA results in making the system more effective but what changes does it make to the final output are studied from the below table for a fixed type (*type(1, 2, 3)*).

Using these values, a histogram can be developed to show the effect of ***minType*** and ***fixType*** on both the cases (case – 1 and case – 2)

Figure 6 shows the effect of SSA on optimized cost for case – 1 and figure 7 shows the effect of SSA on optimized cost for case – 2.

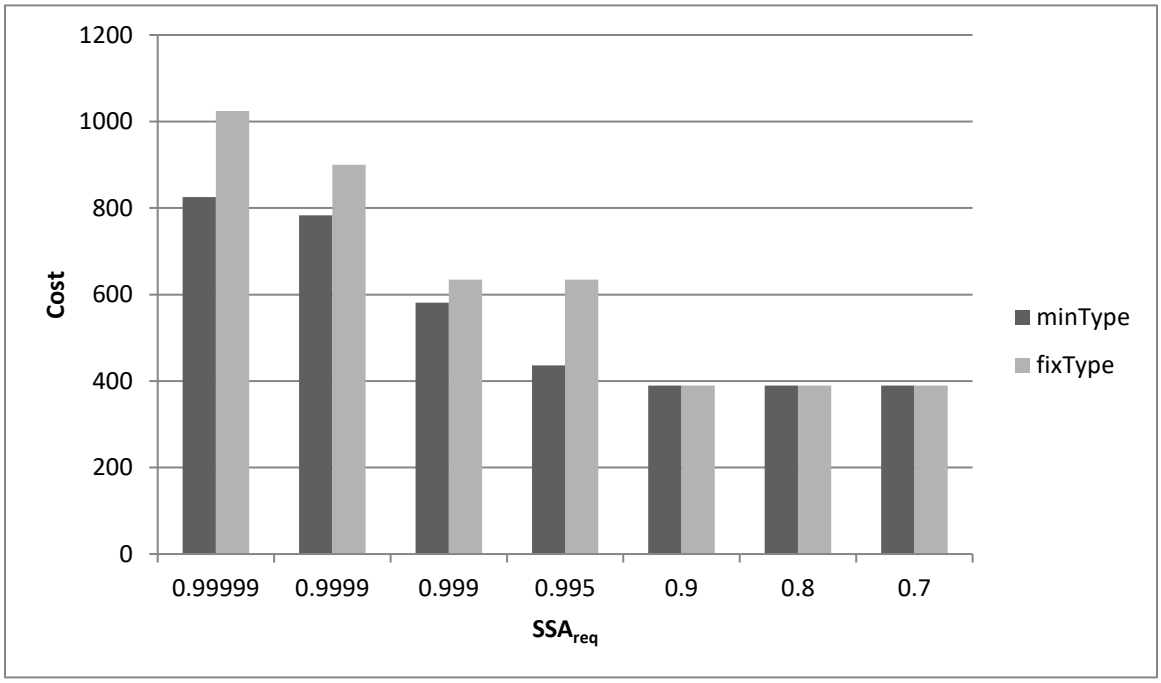

Figure 6. Effect of SSA_{req} on cost in *minType* and *fixType* under case – 1

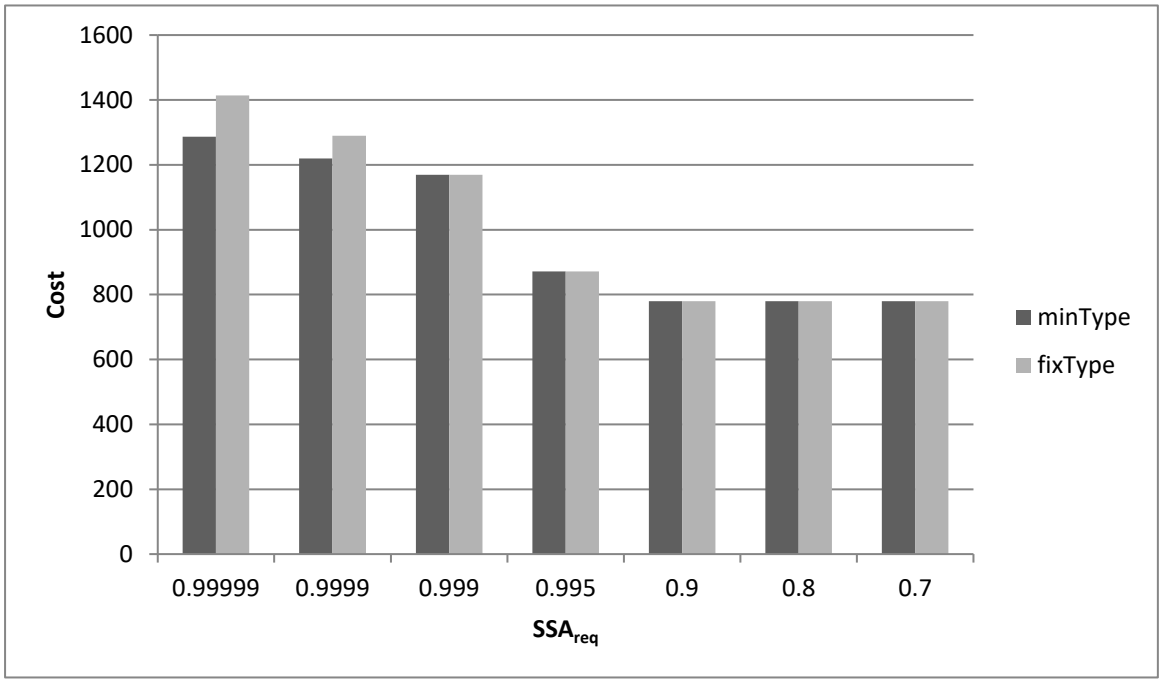

Figure 7. Effect of SSA_{req} on cost in *minType* and *fixType* under case - 2

4.2.3 Selection of VMs best suited for the application

In a general situation, the ASPs incline on buying the cheapest VM for running the application. But is it the best option? Does it really justify the money spent on buying the cheapest VM? Are the ASPs saving any money by doing this? All these questions are unravelled in this section. In this section, three conditions are considered: the first one is to go for only cheapest VMs, the second is to pick the most expensive VMs, and the third is to allow a mixture of all available VMs to run the application.

***Condition – 1*:** It is considered that all the servers must only run by using type 1 VM instances, since type 1 VMs are the cheapest among the three VMs considered. This refers to the combination ($fixType_1$, $fixType_2$, $fixType_3$) = (1, 1, 1) (also represented as *fixType(1, 1, 1)*).

This case is implemented by adding the following constraints to the optimization problem given by (3), (4) and (5):

$n_{i,j} = 0, \forall i,j \text{ where } j \neq 1$

i.e. for server i, set $n_{i,j} = 0$ for $j = 2, 3$.

***Condition – 2*:** In this condition, all the servers must run only on type 3 VM instances. Among the three types, type 3 VM instances are the most expensive ones. This refers to the combination ($fixType_1$, $fixType_2$, $fixType_3$) = (3, 3, 3) (or *fixType(3, 3, 3)*).

This case is implemented by adding the following constraints to the optimization problem given by (3), (4) and (5):

$n_{i,j} = 0, \forall i,j \text{ where } j \neq 3$

i.e. for server i, set $n_{i,j} = 0$ for $j = 1, 2$.

***Condition – 3*:** This condition uses a mixture of all three VM instances to run the application i.e., in this condition, the Web server can run on VMs of type 1 or higher, App server can run on VMs of type 1 or higher and DB server can also run on VMs of type 1 or higher. This refers to the combination ($minType_1$, $minType_2$, $minType_3$) = (1, 1, 1) (or *minType(1, 1, 1)*).

All these 3 conditions are compared to obtain optimized cost for a fixed availability requirement, $SSA_{req} = 0.99999$.

For Case – 1:

Table 11 shows the optimal cost and VM allocation for conditions 1, 2 and 3 for a fixed SSA requirement of 0.99999.

Condition	Number and type of VMs allocated to Web server. $[n_{1,\ type1},\ n_{1,\ type2},\ n_{1,\ type3}]$	Number and type of VMs allocated to App server. $[n_{2,\ type1},\ n_{2,\ type2},\ n_{2,\ type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\ type1},\ n_{3,\ type2},\ n_{3,\ type3}]$	Total cost ℂ (in dollars)
fixType(1, 1, 1)	[3, 0, 0]	[3, 0, 0]	[3, 0, 0]	1084.05
fixType(3, 3, 3)	[0, 0, 2]	[0, 0, 2]	[0, 0, 2]	871.62
minType(1, 1, 1)	[0, 1, 1]	[1, 0, 1]	[0, 1, 1]	804.46

Table 11 Comparison of conditions 1, 2, 3 under case – 1

From Table 11, it is observed that Condition 1 allocates the highest number of VMs when compared to conditions 2 & 3. Condition 1, which uses the cheapest VMs allocates 9 VMs and as a result has the highest cost. When considering both condition 2 & condition 3, they each allocate 6 VMs. From the 3 conditions, condition 3 offers the best optimized output with comparatively low cost and less number of VMs. It is also observed that even the condition with allocating highest number of VMs is offered at lower cost than the condition that allocates the cheaper VMs.

The above discussed conditions are further tested for the effect of steady state availability on the system and VM allocation. They are tested for different SSA_{req} values and a histogram is developed to display the outcomes of all the three conditions.

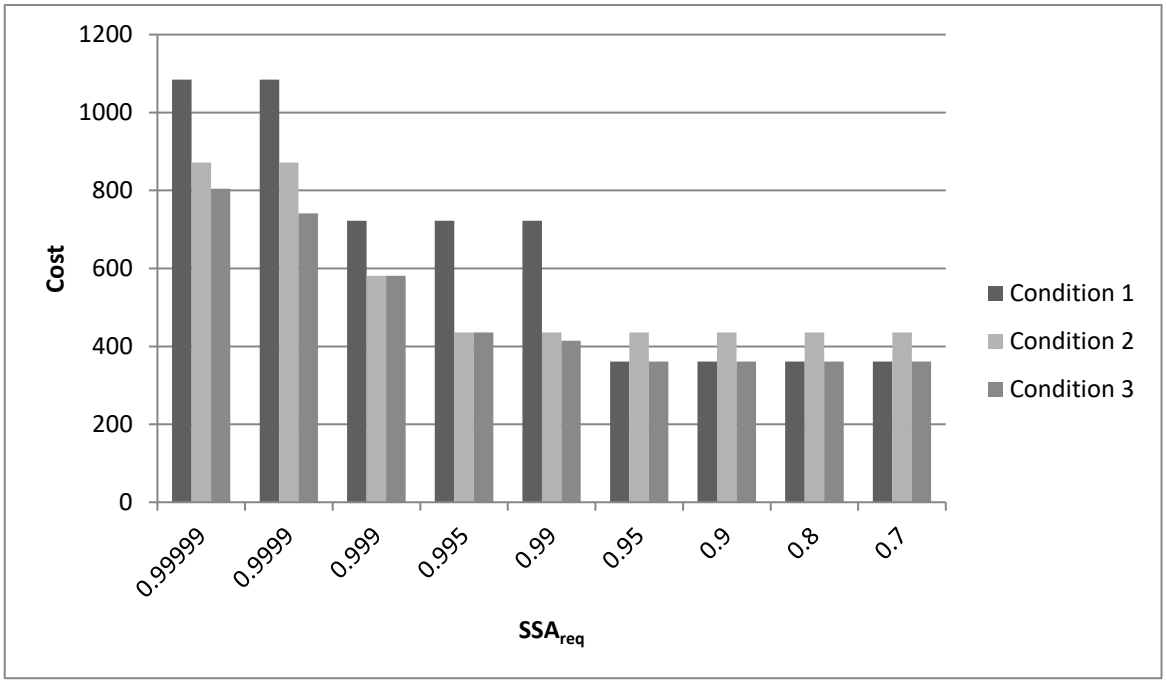

Figure 8. Cost comparison between conditions for different SSA_{req} values in case – 1

Figure 8 shows that if the application's availability requirement is 0.95 or less, only one VM is allocated for each server. For Condition – 1 and Condition – 3, the allocation is quite similar for all the servers, which is one type 1 VM with a minimum cost of $361.35. Also when all values are considered, condition – 3 has a minimum cost and is regarded as the better option.

For Case – 2:

In this case, the application is available if at least k_i copies are working for server *i*. We assume $k_1 = k_2 = k_3 = 2$. Table 12 shows the optimal VM allocation and cost for all the three conditions. The results shown are for a fixed steady – state availability of 0.99999.

Condition	Number and type of VMs allocated to Web server. $[n_{1,\,type1}, n_{1,\,type2}, n_{1,\,type3}]$	Number and type of VMs allocated to App server. $[n_{2,\,type1}, n_{2,\,type2}, n_{2,\,type3}]$	Number and type of VMs allocated to DB server. $[n_{3,\,type1}, n_{3,\,type2}, n_{3,\,type3}]$	Total cost ℂ (in dollars)
fixType(1, 1, 1)	[0, 0, 0]	[0, 0, 0]	[0, 0, 0]	-*NA*-
fixType(3, 3, 3)	[0, 0, 3]	[0, 0, 3]	[0, 0, 3]	1307.43
minType(1, 1, 1)	[0, 0, 3]	[0, 0, 3]	[0, 1, 2]	1286.26

Table 12 Comparison of conditions 1, 2, 3 under case – 2

Table 12 shows the VM allocation and cost for all the conditions where SSA = 0.99999. It is observed from Table – 12 that for condition – 1 there is no possible allocation to satisfy the optimal problem. It is also observed that, condition – 2 & condition - 3 allocate 9 VMs and condition – 2 has the highest cost. Similar to case – 1, it is better to buy a mixed set of VMs rather than buying the cheapest VMs. The problem being, allocating cheapest VMs reduces the system availability and in result requires more VMs to run the application. In this case, both the working conditions condition – 2 & condition – 3 allocates 9 VMs but in condition – 2, all the 9 VMs are of type 3. In condition – 3 however, 8 VMs are of type – 3 and 1 VM is of type – 2.

Similar to case – 1, the above conditions are further tested with different values of SSA_{req} to study the effect of availability on VM allocation and cost. A histogram is developed to compare the three conditions.

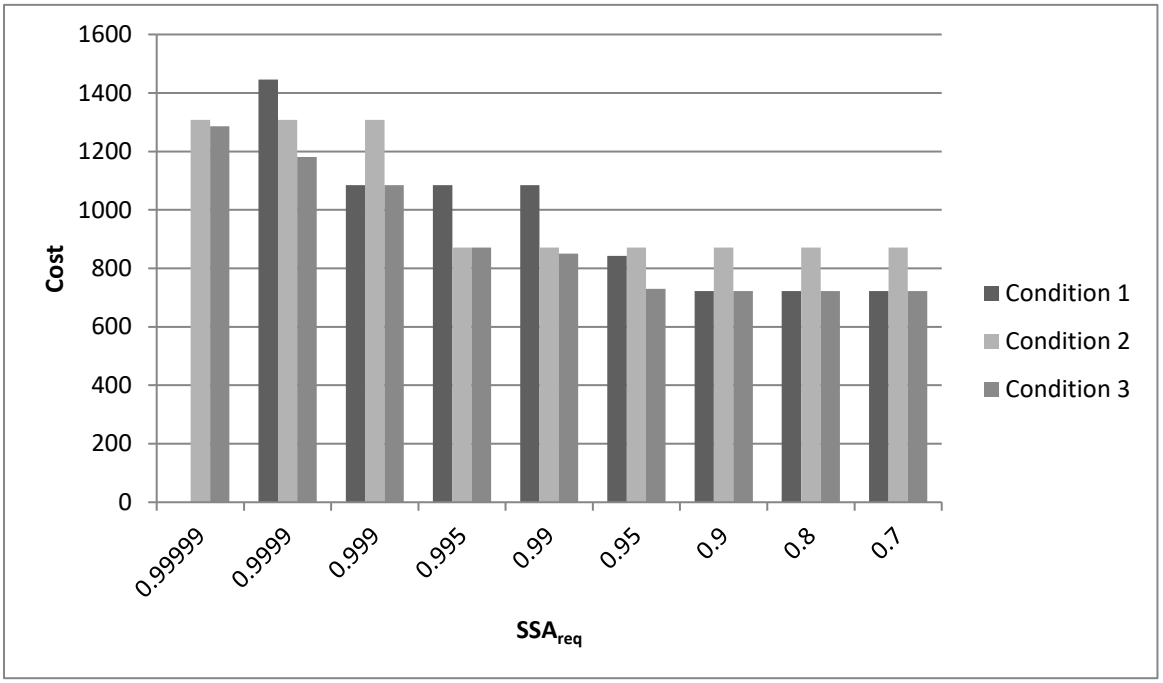

Figure 9. Cost comparison between conditions for different SSA values in case - 2

From the figure, it is observed that compared to all the three conditions, the condition – 3 has a lesser cost. The minimum being $722.7 for systems with SSA_{req} = 0.9 and less. The VM allocation stays constant from 0.9 with ach server using 2 VMs of type – 1. The number of VMs further decreases when the steady – state availability reaches values less than 0.5.

From Figure 5.2.3 and 5.2.4, it is concluded that when the cost of the system is considered, buying only the cheapest VMs or expensive VMs (high availability VMs) to run an

application is not a best decision. To get an optimal VM allocation with minimized system cost, it is always preferred to consider a combination of all available VMs.

4.3 Framework

This section shows the layout of running the code discussed in chapter 3. The entire code is written in AMPL and is divided into 3 different levels. They are: mod file, data file and run file.

The code for these 3 files can be written in notepad and saved as a respective file type by adding an extension. The solution model to be solved is entered into the mod file. Once the model is entered into a text file, the file is saved with a *.mod* extension and saved in the database. Similarly, the necessary data required for the model to run is saved in the data file with a *.dat* extension. The order of running the code and output structure is presented in the run file. This is created by using the *.script.run* extension [11, 21, 23, 37].

All these files together create the final framework of the code. Once the files are created, the solver is run to generate output. In the solver, *ampl* is entered to specify the type of language being used. To open or run a file the command used is *include file_name*. *file_name* is the name of the file that is being used to solve the code. For example, consider a code *example1* having 3 different files *example1.mod*, *example1.dat* and *example1.script.run*. The main layout of the code is presented in *example1.script.run* and this is considered to be the run file. So, different commands entered in the solver are as follows:

```
sw: ampl
ampl: include example1.script.run
```

This command opens the run file and all the commands in the file are run. The final output is displayed after the solver finds an optimal output.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 Summary

The main task of an ASP is to provide the required number of VMs depending on the workload so that the application runs smoothly without any drop in QoS. They have to determine the number of VMs required for each type to run the copies of each server.

In this thesis, a three - tiered cloud system has been proposed with each tier having multiple copies of servers. Each tier is assumed to have one or more VMs, each running a single instance of server relevant to the tier. Another assumption made is that the workload is equally distributed among the servers at any given tier.

The user's request is first sent to a web server for processing and the request is processed exactly once at each tier. After the processing is done at the third tier, the response is returned to the user. Some other assumptions made are that the software servers do not fail, the application's availability depends on

the availability of the VMs and the minimum number of copies needed for each server and the minimum computing power of VM instances are specified as input.

An optimization approach is developed which helps the ASPs in making decisions on some important questions faced related to VM acquisition. The model proposed is non-linear in nature and uses ILOGCP solver developed by IBM to solve the problem. This approach benefits the ASPs in answering questions such as, will it more cost-effective to run the server copies on a fixed type of VMs or on different types of VM instances? Will it be more cost-effective to buy cheapest VMs or a combination of VMs?

5.2 Conclusion

Through this thesis, it can be decided that when it comes to selecting between the cheapest VMs and combination of different types of VMs, it is always best to go with a combination of VMs because although the cost is reduced by selecting the cheapest VMs, the total system availability is greatly affected and to produce the given availability SLA the application has to allocate additional VMs. In case of selecting if it is best to run the servers on a fixed type of VMs or different types of VMs, it is observed that running on different types of VMs provides a similar result for a cheaper cost when compared to a fixed type.

5.3 Future Works

The future works might include eliminating some of the assumptions. The following are some of the proposed future works:

i. The proposed model can be further developed to produce a heuristic method that is much efficient compared to the discussed optimal solution.

ii. In this thesis, the usage of VMs has been limited to a specific set for each tier which can be explored and can be made to choose freely across different tiers. This might further reduce the number of VMs needed to run the application leading to a cheaper system.

iii. Addition of other QoS attributes such as system response time is also recommended to improve the user's experience.

Appendix

Case – 1:

mod file:

```
set TIER;

set VM;


param cost {VM} > 0;

param unavail {VM} > 0;


var Buy{TIER,VM} integer >= 0, <= 10;


minimize Total_Cost: sum {j in VM} ( cost[j] * (sum {i in TIER} Buy[i,j]) );


subject to inequality1:

prod {i in TIER} (1- (prod{j in VM} (unavail[j])^ Buy[i,j] )) >= 0.999;


subject to inequality2:

#sum {j in VM} Buy[i,j] >= 1;

Buy["TIER1","TYPE1"] + Buy["TIER1","TYPE2"] + Buy["TIER1","TYPE3"] >= 1;

subject to inequality3:

Buy["TIER2","TYPE1"] + Buy["TIER2","TYPE2"] + Buy["TIER2","TYPE3"] >= 1;

subject to inequality4:

Buy["TIER3","TYPE1"] + Buy["TIER3","TYPE2"] + Buy["TIER3","TYPE3"] >= 1;
```

```
/*
subject to inequality5:
Buy["TIER1","TYPE1"] = 0;
*/
subject to inequality6:
Buy["TIER1","TYPE2"] = 0;

subject to inequality7:
Buy["TIER1","TYPE3"] = 0;



subject to inequality8:
Buy["TIER2","TYPE1"] = 0;
/*
subject to inequality9:
Buy["TIER2","TYPE2"] = 0;
*/
subject to inequality10:
Buy["TIER2","TYPE3"] = 0;



subject to inequality11:
Buy["TIER3","TYPE1"] = 0;

subject to inequality12:
```

```
Buy["TIER3","TYPE2"] = 0;

/*

subject to inequality13:

Buy["TIER3","TYPE3"] = 0;

*/
```

dat file:

```
data;


set TIER := TIER1 TIER2 TIER3 ;

set VM := TYPE1 TYPE2 TYPE3 ;


param:  cost      unavail :=

TYPE1 120.45  0.01

TYPE2 124.10  0.005

TYPE3 145.27  0.0005;
```

run file:

```
reset;

model Case1.mod;

data  Case1.dat;

option solver ilogcp;

let Buy["TIER1","TYPE2"] := 0;

let Buy["TIER1","TYPE3"] := 0;

solve;
```

```
display Buy;
display Total_Cost;
```

Case – 2:

mod file:

```
set TIER;
set VM;

param cost {VM} > 0;
param unavail {VM} > 0;

var Buy{TIER,VM} integer >= 0, <= 4;

minimize Total_Cost: sum {j in VM} ( cost[j] * (sum {i in TIER} Buy[i,j]) );

subject to inequality1: exists {k in 0..4} k = Buy["TIER1","TYPE1"] &&
                    exists {l in 0..4} l = Buy["TIER1","TYPE2"] &&
                                exists {m in 0..4} m = Buy["TIER1","TYPE3"] &&
                                exists {a in 0..4} a = Buy["TIER2","TYPE1"] &&
                    exists {b in 0..4} b = Buy["TIER2","TYPE2"] &&
                                exists {c in 0..4} c = Buy["TIER2","TYPE3"] &&
```

```
                    exists {d in 0..4} d = Buy["TIER3","TYPE1"] &&
               exists {e in 0..4} e = Buy["TIER3","TYPE2"] &&
                    exists {f in 0..4} f = Buy["TIER3","TYPE3"] &&
                    ( sum {x in 0..k, y in 0..l, z in 0..m: x+y+z>=2}
                ((prod {p in 0..k} if p = 0 then 1 else p)/((prod {p in 0..(k-x)} if p = 0 then 1 else
p)*(prod {p in 0..x} if p = 0 then 1 else p)) * (1-unavail["TYPE1"])^x * unavail["TYPE1"]^(k-x)) *
                    ((prod {q in 0..l} if q = 0 then 1 else q)/((prod {q in 0..(l-y)} if q = 0 then 1
else q)*(prod {q in 0..y} if q = 0 then 1 else q)) * (1-unavail["TYPE2"])^y * unavail["TYPE2"]^(l-y))
*
                    ((prod {r in 0..m} if r = 0 then 1 else r)/((prod {r in 0..(m-z)} if r = 0 then 1
else r)*(prod {r in 0..z} if r = 0 then 1 else r)) * (1-unavail["TYPE3"])^z * unavail["TYPE3"]^(m-z)) )
*
                    ( sum {x in 0..a, y in 0..b, z in 0..c: x+y+z>=2}
                ((prod {p in 0..a} if p = 0 then 1 else p)/((prod {p in 0..(a-x)} if p = 0 then 1 else
p)*(prod {p in 0..x} if p = 0 then 1 else p)) * (1-unavail["TYPE1"])^x * unavail["TYPE1"]^(a-x)) *
                    ((prod {q in 0..b} if q = 0 then 1 else q)/((prod {q in 0..(b-y)} if q = 0 then 1
else q)*(prod {q in 0..y} if q = 0 then 1 else q)) * (1-unavail["TYPE2"])^y * unavail["TYPE2"]^(b-y))
*
                    ((prod {r in 0..c} if r = 0 then 1 else r)/((prod {r in 0..(c-z)} if r = 0 then 1 else
r)*(prod {r in 0..z} if r = 0 then 1 else r)) * (1-unavail["TYPE3"])^z * unavail["TYPE3"]^(c-z)) ) *
                    ( sum {x in 0..d, y in 0..e, z in 0..f: x+y+z>=2}
                ((prod {p in 0..d} if p = 0 then 1 else p)/((prod {p in 0..(d-x)} if p = 0 then 1 else
p)*(prod {p in 0..x} if p = 0 then 1 else p)) * (1-unavail["TYPE1"])^x * unavail["TYPE1"]^(d-x)) *
                    ((prod {q in 0..e} if q = 0 then 1 else q)/((prod {q in 0..(e-y)} if q = 0 then 1
else q)*(prod {q in 0..y} if q = 0 then 1 else q)) * (1-unavail["TYPE2"])^y * unavail["TYPE2"]^(e-y))
```

```
*
                    ((prod {r in 0..f} if r = 0 then 1 else r)/((prod {r in 0..(f-z)} if r = 0 then 1 else
r)*(prod {r in 0..z} if r = 0 then 1 else r)) * (1-unavail["TYPE3"])^z * unavail["TYPE3"]^(f-z)) ) >=
0.9999;



subject to inequality2:
Buy["TIER1","TYPE1"] + Buy["TIER1","TYPE2"] + Buy["TIER1","TYPE3"] >= 2;
subject to inequality3:
Buy["TIER2","TYPE1"] + Buy["TIER2","TYPE2"] + Buy["TIER2","TYPE3"] >= 2;
subject to inequality4:
Buy["TIER3","TYPE1"] + Buy["TIER3","TYPE2"] + Buy["TIER3","TYPE3"] >= 2;


/*
subject to inequality5:
Buy["TIER1","TYPE1"] = 0;
*/
subject to inequality6:
Buy["TIER1","TYPE2"] = 0;


subject to inequality7:
Buy["TIER1","TYPE3"] = 0;



subject to inequality8:
```

```
Buy["TIER2","TYPE1"] = 0;

/*

subject to inequality9:

Buy["TIER2","TYPE2"] = 0;

*/

subject to inequality10:

Buy["TIER2","TYPE3"] = 0;




subject to inequality11:

Buy["TIER3","TYPE1"] = 0;


subject to inequality12:

Buy["TIER3","TYPE2"] = 0;

/*

subject to inequality13:

Buy["TIER3","TYPE3"] = 0;

*/
```

dat file:

```
data;


set TIER := TIER1 TIER2 TIER3 ;

set VM := TYPE1 TYPE2 TYPE3 ;
```

```
param:  cost      unavail :=

TYPE1 120.45  0.01

TYPE2 124.10  0.005

TYPE3 145.27  0.0005;
```

run file:

```
reset;

model Case2.mod;

data  Case2.dat;

option solver ilogcp;

solve;

display Buy;

display Total_Cost;
```

References

[1] A. Aldhalaan and D. A. Menascé, "Autonomic Allocation of Communicating Virtual Machines in Hierarchical Cloud Data Centers", In *International Conference on Cloud and Autonomic Computing (ICCAC 2014),* pp. 161-171, September 2014.

[2] A. Aldhalaan and D. A. Menascé, "Near-optimal Allocation of VMs from IaaS Providers by SaaS Providers," In *International Conference on Cloud and Autonomic Computing (ICCAC 2015),* pp. 228-231, September 2015.

[3] A. Alamri, M. M. Hassan, "Virtual Machine Resource Allocation for Multimedia Cloud: A Nash Bargaining Approach", In *International Symposium on Emerging Inter-networks, Communication and Mobility* (EICM-2014), vol. 34, pp. 571-576, December 2014.

[4] A. Anand, J. Lakshmi, and S. K. Nandy, "Virtual Machine Placement Optimization supporting Performance SLAs", In *5th International Conference on Cloud Computing Technology and Science (CloudCom),* pp. 298-305, December 2013.

[5] A. Undheim, A. Chilwan, and P. Heegaard, "Differentiated Availability in Cloud Computing SLAs", In *12th IEEE/ACM International Conference on Grid Computing* (GRID), pp. 129-136, September 2011.

[6] B. Jennings and R. Stadler, "Resource management in Clouds: Survey and research challenges," *Journal of Network and Systems Management,* vol. 23, no. 3, pp. 567-619, July 2015.

[7] D. A. Menascé and P. Ngo, "Understanding Cloud Computing: Experimentation and Capacity Planning," In *International Computer Measurement Group Conference*, January 2009.

[8] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-Service in Cloud Computing: Modeling techniques and their applications," In *Journal of Internet Services and Applications*, vol. 5, no. 1, pp. 1-17, January 2014.

[9] D. Petcu, G. Macariu, S. Panica, C. Crăcium, "Portable Applications – From theory to Practice", In *Future Generation Computer Systems*, vol. 29, no. 6, pp. 1417-1430, August 2013.

[10] E. Casalicchio, D. A. Menascé, and A. Aldhalaan, "Autonomic Resource Provisioning in Cloud Systems with Availability Goals," *In Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC 2013)*, August 2013.

[11] E. D. Dolan, "The NEOS Server 4.0 Administrative Guide", In *Technical Memorandum ANL/MCS-TM-250, Mathematics and Computer Science Division, Argonne National Laboratory*, 2001. This technical report, which discusses the implementation of the server and its use in detail, is available for download in PDF format.

[12] E. Gorelik, "Cloud Computing Models: Comparison of Cloud Computing Service and Deployment Models", Masachusetts Institute of Technology, Cambridge.

[13] E. K. Mece, E. Driza, "An Approach to Evaluate the Reliability of Web Applications in Cloud Computing using Dynamic Fault Tree", In *Balkan Conference in Informatics*, Thessaloniki, Greece, pp. 19-21, January 2013.

[14] G. Alves, C. Silva, E. Cavalcante, T. Batista and F. Lopes, "Relative QoS: A New Concept for Cloud Service Quality", In *2015 IEEE Symposium on Service-Oriented System Engineering(SOSE 2015)*, pp. 59-68, April 2015.

[15] G. Wei, A. V. Vasilakos, Y. Zheng, and N. Xiong, "A game-theoretic method of fair resource allocation for cloud computing services," In *The Journal of supercomputing*, vol. 54, no. 2, pp. 252-269, November 2010.

[16] H. Goudarzi, M. Ghasemazar, and M. Pedram, "SLA-based optimization of power and migration cost in cloud computing," In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012)*, pp. 172-179, May 2012.

[17] H. Goudarji and M. Pedram, "Multi-dimensional SLA-based resource allocation for multitier cloud computing systems," In *IEEE International Conference on Cloud Computing (CLOUD 2011)*, pp. 324-331, July 2011.

[18] http://www.eventhelix.com/RealtimeMantra/FaultHandling/system_reliability_availability.html, Date accessed: January 6, 2017

[19] H. Kamal Idrissi, A. Kartit, M. El. Marraki, "A Taxonomy and Survey of Cloud Computing", In *2013 National Security Days (JNS3)*, April 2013.

[20] IBM Archives, A Brief History of Cloud Computing, *https://www.ibm.com/blogs/cloud-computing/2014/03/a-brief-history-of-cloud-computing-3/*, Date accessed: January 6, 2017

[21] ILOG CP solver, *Demo version of AMPL and solvers.* http://www.ampl.com/try-ampl/download-a-free-demo/, 2016.

[22] I. Rodero, E. K. Lee, D. Pompili, M. Parashar, M. Gamell, and R. J. Figueiredo, "Towards energy-efficient reactive thermal management in instrumented datacenters," In *11th IEEE/ACM International Conference on Grid Computing*, pp. 321-328, November 2010.

[23] J. Czyzyk, M. P. Mesnier, and J. J. Moré, "The NEOS Server",In *IEEE Journal on Computational Science and Engineering*, vol. 5, no. 3, pp. 68-75, July 1998. This paper discusses the design and implementation of the NEOS Server.

[24] L. Wu. S. K. Garg, and R. Buyya, "SLA-based resource allocation for Software-as-a-service provider (SaaS) in cloud computing environments," In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2011)*, pp. 195-204, May 2011.

[25] M. Hadji and D. Zeghlache, "Minimum cost maximum flow algorithm for dynamic resource allocation in clouds," *In IEEE 5th International Conference on Cloud Computing (CLOUD 2012)*, pp. 876-882, June 2012.

[26] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, November 2011.

[27] O. Das, "Availability Modeling", Lecture notes for Computer Systems Modeling (EE8214), Ryerson University, Toronto, 2014.

[28] P. Sakhamuri, O. Das, "Acquisition of Virtual Machines for Tiered Applications with Availability Constraints", In *18th International Symposium on High Assurance Systems Engineering* (HASE 2017), Singapore, January 2017. Accepted for publication.

[29] Q. Zhang, Q. Zhu, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic service placement in geographically distributed clouds," In *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 762-772, December 2013.

[30] R. Sahner, K. S. Trivedi and A. Puliafito, "Performance and Reliability Analysis of Computer Systems – An Example-Based Approach using SHARPE Software Package", ISBN 978-1-4613-6005-6, Kluwer Academic Publishers, 1996.

[31] S. B. Shaw, A. K. Singh, "A Survey on Cloud Computing", In *International Conference on Green Computing Communication and Electrical Engineering (ICGCCEE),* March 2014.

[32] S. Dutta, S. Gera, A. Verma, and B. Viswanathan, "Smartscale: Automatic Application Scaling in Enterprise Clouds," *In IEEE 5th International Conference on Cloud Computing (CLOUD 2012),* pp. 221-228, June 2012.

[33] S. Hassan, A. A. Kamboh and Dr. Farooque Azam, "Analysis of Cloud Computing Performance, Scalability, Availability & Security", In *International Conference on Information Science and Applications (ICISA),* pp. 1-5, May 2014.

[34] S. Kamboj, N. S. Ghumman, "A Survey on Cloud Computing and Its Types", In *3rd International Conference on Computing for Sustainable Global Development* (INDIACom), pp. 2971-2974, March 2016.

[35] V. Prakash, and S. Gopalakrishnan, "Cloud computing solution – Benefits and testing challenges", *Journal of Theoretical and Applied Information Technology,* vol. 39, no. 2, pp. 114 – 118, May 2012.

[36] W. E. Dong, W. Nan, and L. Xu, "QoS – oriented Monitoring Model of Cloud Computing Resources Availability", In *5th International Conference on Computational and Information Sciences (ICCIS),* pp. 1537-1540, June 2013.

[37] W. Gropp, and J. J. Moré, "Optimization Environments and the NEOS Server", In *Approximation Theory and Optimization, M. D. Buhmann and A. Iserles, eds.*, Cambridge University Press, pp. 167-182, March 1997. This paper discusses the NEOS Server as a problem-solving environment that simplifies the formulation of optimization problems and the access to computational resources.

[38] Z. I. M. Youssh and M. Tang, "A penalty-based grouping genetic algorithm for multiple composite SaaS components clustering in cloud," In *IEEE International Conference on Systems, Man, and Cybernatics (SMC 2012)*, pp. 1396-1401, October 2012.

[39] Z. I. M. Yuosh and M. Tang, "Composite SaaS placement and resource optimization in cloud computing using evolutionary algorithms," *In IEEE 5th International Conference on Cloud Computing (CLOUD 2012)*, pp. 590-597, June 2012.

[40] Z. Xiao, W. Song, and Q. Chen, "Dynamic Resource Allocation using Virtual Machines for Cloud Computing Environment," In *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp:1107-1117, June 2013.