



Adama Science and Technology University
School of Electrical Engineering and Computing
Department of Computing

Hybrid BA-GA Approach for Cost-Effective SaaS Placement
In Cloud Environment

Jemal Nuradis

Feb 2017



Adama Science and Technology University
School of Electrical Engineering and Computing
Department of Computing

**Hybrid BA-GA Approach for Cost-Effective SaaS Placement
In Cloud Environment**

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Software Engineering

Jemal Nuradis

Feb 2017

Adama, Ethiopia



Adama Science and Technology University
School of Electrical Engineering and Computing
Department of Computing

Hybrid BA-GA Approach for Cost-Effective SaaS Placement
In Cloud Environment

By

Jemal Nuradis

Names and Signature of Members of the Examining Board

Chair Person

Signature

Frezawud Lema (PHD)
Advisor

Signature

External Examiner

Signature

Internal Examiner

Signature

Department Head

Signature

School Dean

Signature

DECLARATION

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other work, and that all source of materials used for the thesis have been duly acknowledged.

Declared by:

Name: Jemal Nuradis

Signature: _____

Date: _____

Confirmed by advisor:

Name: Frezewud Lema(PhD)

Signature: _____

Date: _____

Place and date of submission: Adama, Feb 2017

ACKNOWLEDGEMENT

First and for most, I thank Allah for granting me knowledge, health, and the patience to successfully complete this thesis.

I would also like to show my appreciation to the following people:

Dr. Frezewud Lema, my advisor, for his good feedback and continuous encouragement constructive feedback for the report and valuable comments in the progress meetings throughout the thesis.

Special thanks and gratitude to Dr. Mesfin Abebe for his valuable comments and suggestions and also to all my colleagues and friends, especially freezer guteta, seifu berihanu and dagnachew Getachew, who have been there encouraging and supporting.

Last, but not least, I would like to express my appreciation to my parents, my brothers, and my sisters who have always been supportive and encouraged me morally.

TABLE OF CONTENTS

DECLARATION	i
ACKNOWLEDGEMENT	ii
LIST OF FIGURES	vi
LIST OF TABLES	vii
ABSTRACT.....	x
CHAPTER ONE: INTRODUCTION	1
1.1 Background	1
1.1.1 Software Task Placement Problem in Cloud Environment.....	3
1.2 Statement of the Problem	4
1.3 Research Objectives.....	5
1.3.1 General Objectives	5
1.3.2 Specific Objectives	5
1.4 Research Methodology	5
1.5 Scope and Limitation of the Study.....	7
1.6 Research Questions	8
1.7 Significance of the Study	8
1.8 Organization of the Paper	10
CHAPTER TWO: LITERATURE REVIEW AND RELATED WORK	11
2.1 Literature Review.....	11
2.1.1 Introduction.....	11
2.1.2 Cloud Computing.....	11
2.1.2.1 Cloud Computing Service Models	12
2.1.2.1.1 Infrastructure as a Service	13
2.1.2.1.2 Platform as a Service.....	13
2.1.2.1.3 Software as a Service	13
2.1.2.2 Cloud Computing Deployment Models	14
2.1.2.2.1 Public Cloud Model	15
2.1.2.2.2 Private Cloud	15
2.1.2.2.3 Hybrid Cloud	15
2.1.2.2.4 Community Cloud.....	16
2.1.2.3 Virtualization	16

2.1.2.4	SaaS in Cloud Environment	17
2.1.2.4.1	SaaS Service Providers	19
2.1.2.4.2	SaaS Deployment	20
2.1.2.4.3	Application Delivery Model of SaaS	21
2.1.2.4.4	Maturity Model of SaaS	22
2.1.3	Optimization Algorithms	24
2.1.3.1	Evolutionary Algorithm	25
2.1.3.2	Swarm Intelligent Algorithms	26
2.1.3.3	Genetic Algorithm	26
2.1.3.4	Particle Swarm Optimization Algorithm	27
2.1.3.5	Ant Colony Algorithm	28
2.1.3.6	Greedy Algorithm	29
2.1.3.7	Bat Algorithm	30
2.1.4	Cloud Simulation Tools	33
2.1.4.1	CloudSim	33
2.1.4.2	Datacenter Simulation Tool(DCSim)	34
2.1.4.3	Greencloud Simulator	35
2.2	Related Works	38
	Summary	43
	CHAPTER THREE: PROPOSED SOLUTION	44
3.1	Introduction	44
3.2	ST Placement	44
3.3	ST Placement Representation Model Across Datacenters	45
3.4	Proposed Solution	47
3.4.1	Structure of Proposed Algorithm	48
3.5	Placement Approach	50
	Summary	60
	CHAPTER FOUR: EXPERIMENTAL SETUP AND IMPLEMENTATION	61
4.1	Introduction	61
4.2	CloudSim	61
4.2.1	Classes in CloudSim	62
4.2	Simulation Environment	65

4.2.1	CloudSim Configuration	67
4.2.2	Datacenter Physical Hosts Configuration	67
4.2.3	Datacenter Virtual Machines Configuration	68
4.2.4	Datacenter Broker Configuration	68
4.2.5	Datacenter Characteristics Configuration	68
4.3	Implementation	70
4.3.1	Simulation Scenario	70
4.3.2	Terminology and Resource Attribute Used in the Experiment	70
4.3.3	Experiment Scenario 1	71
4.3.4	Experiment Scenario 2	72
4.3.5	Experiment Scenario 3	73
4.3.6	Experiment Scenario 4	74
4.3.7	Experiment Scenario 5	75
	Summary	76
	CHAPTER FIVE: RESULT ANALYSIS AND EVALUATION	77
5.1	Result Analysis	77
5.1.1	Introduction	77
5.1.2	Experimental Result On changing Size of VM Resources On Located DC	77
5.1.3	Experimental Result Changing the Size of ST on Located DC	79
5.1.4	Experimental Result on Changing Number of DC	80
5.1.5	Experimental Result on Changing Number of ST Across DC	82
5.1.6	Experimental Result on Changing Number of VM Across DC	83
5.2	Result Evaluation	85
	Summary	87
	CHAPTER 6: CONCLUSION AND FUTURE DIRECTION	88
6.1	Conclusion	88
6.2	Future Direction	90
	REFERENCES	91
	APPENDICES I: Simulation Environment Configuration	96
	APPENDICES II: Algorithms	98

LIST OF FIGURES

Figure 1.1: Software task Placement Scenario in Cloud Environment	3
Figure 1.2: Research Method Phases Diagram.....	6
Figure 1.3: Thesis Organization	10
Figure 2.1: Cloud Service Model	14
Figure 2.2: Cloud Deployment Model.....	16
Figure 2.3: SaaS in Cloud Environment.....	21
Figure 2.4: SaaS Maturity Model	23
Figure 2.5: Optimal Design Procedure.....	24
Figure 2.6: Genetic Algorithm	27
Figure 2.7: Particle Swarm Optimization Algorithm	28
Figure 2.8: Anti-colony Algorithm.....	29
Figure 2.9: Greedy Algorithm	30
Figure 2.10: Bat Algorithm	31
Figure 2.11: Architecture of CloudSim	33
Figure 2.12 Architecture of DCSim	34
Figure 2.13: Architecture of Greencloud Simulator.....	35
Figure 3.1: Proposed Solution.....	47
Figure 3.2: Placement Strategy Using BA.....	50
Figure 3.3 BA Control Attribute	53
Figure 3.4: Initial Placement Representation Across DC	53
Figure 3.5 Generated ST into Virtual Machine Resources	54
Figure 3.6: The Second Placement Strategy Using GA.....	55
Figure 3.7: Placement Constraint Checking Procedure Using GA.....	57
Figure 3.8: Placement Configuration Using Crossover	57
Figure 3.9: Placement Configuration Using Mutation	58
Figure 3.10: Identification of New Placement Position Using BA.....	59
Figure 4.1: CloudSim Class Design Diagram.....	64
Figure 4.2 Simulation Environment of CloudSim	65
Figure 4.3: Simulation Screen	69
Figure 4.4: Sample Experiment Scenario 1 Simulation Results.....	71
Figure 4.5: Sample Experiment Scenario 2 Simulation Results.....	72
Figure 4.6: Sample Experiment Scenario 3 Simulation Results.....	73
Figure 4.7: Sample Experiment Scenario 4 Simulation Results.....	74
Figure 4.8: Sample Experiment Scenario 5 Simulation Results.....	75
Figure 5.1 Placement of ST Components on Located DC by Varying Number of VMs	77
Figure 5.2 Placement of ST Components on Located DC by Varying Number of ST.....	79
Figure 5.3 Placement of ST Across DC by fixed the VM and ST Resources	80
Figure 5.4 Placement of ST Across DC by Changing The size of ST Components	82
Figure 5.5 Placement of ST Across DC by Changing The VM Resources.....	83

LIST OF TABLES

Table 2.1: SaaS Service Provider.....	19
Table 2.2: Comparison of Optimization Algorithms	31
Table 2.3: Cloud Computing Simulation Tool	36
Table 2.4: Summery of Related Works	41
Table 4.1: Simulation Component and Configuration of the Experiment	66
Table 4.2: Terminology Used in The Experiment.....	70
Table 4.3: Resource Attribute Used in The Experiment.....	71
Table 5.1 Placement Algorithm Performance Evaluation on Finding Optimum Placement Area on located and Across DC	85

ACRONYMS AND ABBREVIATION

ACO:	Anti-Colony Optimization Algorithms
APPs:	Applications
ASP:	Application Service Provider
AWS:	Amazon Web service
BA:	Bat algorithm
BW:	Bandwidth
CSD	Current Selected Datacenter
CD:	Compact Disk
CPU:	Central Processing Unit
DC:	Datacenter
DCSim:	Datacenter Simulations
EA:	Evolutionary Algorithm
GA:	Genetic Algorithm
HA:	Hybrid Algorithm
IaaS:	Infrastructure as a Service
JDK	Java Development Kit
IP:	Internet Protocol
IT:	Information Technology
MIPS:	Millions of Instructions per Second
OS:	Operating System
PaaS:	Platform as a Service
PSO:	Particle Swarm Optimization Algorithm
RAM:	Random Access Memory
RSD	Random selected Datacenter
SaaS:	Software as a Service
SI:	Swarm Intelligent Algorithms
SLAs:	Service Level Agreements
SOA:	Service Oriented Architecture
ST:	Software Tasks

TC: Transfer Cost
VLANs: Virtual local Area Network
VM: Virtual Machine
VMM: Virtual Machine Monitor

ABSTRACT

The increasing demand of software service in cloud environment needs strategic placement in the cloud infrastructure. Thus, in which the users use the service based on the service model of the provider and pays based on their use of the resources. These resources are storage, memory processing element and bandwidth. Efficient optimal placement is the main issue in order to provide a cost effective service to the user. In the Software task placement activity, the problem is a selection of virtual machine resource to place software task component that meets the requirement of the resources. However, selection of virtual machine resource to optimize the infrastructure resources is not an easy task to place software task component in the cloud environment. The problem is an optimization problem for initial software task placement. The aim of this study focus on how software task component can be placed in a cost-effective way to optimize the infrastructure resources.

This research has proposed hybrid approaches to addresses the initial software task placement problem by exploring the advantage of both Bat algorithm (BA) and Genetic algorithm (GA), to make the initial ST placement processes optimum and cost effective. In order to evaluate the performance of the proposed hybrid algorithms, an experimental environment is configured using CloudSim simulation tool. The proposed solution performance has evaluated and compared with those existing placement algorithms such as genetic algorithm (GA) and particle swarm optimization algorithm (PSO). Accordingly, the performances of the algorithms are examined by varying the number of datacenters, virtual machines, and tasks. In the experimentation, the proposed algorithm has produced an optimum solution in both located and across datacenters that are compared with genetic and particle swarm optimization placement algorithms. The result from the hybrid algorithm has reduced the initial placement cost by up to 2 -13% on a cloud environment.

Keywords: Cloud computing, Cloud provider, Datacenter, Virtual machine, SaaS, GA, PSO

CHAPTER ONE: INTRODUCTION

1.1 Background

Cloud computing provides different services and resources using Internet services. In these days a lot of information and computing service are provided by different cloud providers. In a cloud environment, cloud computing contains a collection of multiple Datacenters. Each Datacenter has a lot of computing resources to deliver efficient service to the user. These datacenter resources are important to run a different application in a cloud environment. To provide these cloud resources, cloud providers have used own model. To deliver the service, the cloud providers are used different service model such as models Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS) and also four deployment models (Private cloud, Community cloud, Public cloud, Hybrid cloud). In the cloud, the future like On-demand service, Network access, Resource pooling, Rapid elasticity and Measured service are important to provide well-organized and on demand service to the user.

SaaS is one of the cloud computing service models that the user can access the service without having or buying any hardware resources. Using SaaS, users can host their application and use the service without installing and upgrading (licensing) the software. To provide and access their resources to the users, SaaS providers use cloud infrastructure resources to deploy their applications and provide an efficient service to the user. This resources include memory, processing power, the storage element and bandwidth resources.

In traditional application software provider, the user pays once for the software and continue to pay license cost. But in SaaS, the provider care about the maintenance of the software, the data center, the backup, and the support of the system. Currently, some Companies change the desktop software based system into cloud-based software service. Because SaaS providers used a multi-tenant architecture to deliver a flexible and on demand based application. This multi-tenancy concept is important to access a single application by many users and customers can access their data anywhere anytime and pay based on they use. In addition to this, SaaS vendor has provided the service with minimum subscription cost for the user.

SaaS vendor, infrastructure provider and SaaS user are the main participants in SaaS scenario on a cloud environment. The first participant is SaaS application vendor; it has created the software based on their custom interest and submits to the cloud provider. The cloud providers are the second participant, that provide the SaaS service. The cloud provider provides the SaaS service based on their own model. Finally, the SaaS user is the third participant, that can access the resources or services using internet connections. SaaS implemented in a different application area such as resource management application, gaming applications and designing software. Currently, the users of SaaS application services are increased. The 2016 cloud service report shows [1] that the cloud based service has reached 240 billion dollars and each year the growth ratio increase by 16.5%. SaaS is one of the main cloud services that is provided by the cloud provider. This service has contained 18.5% (37.1 billion dollars) and each year the market grows by 20.3%.

The infrastructure resource such as a server, network device, bandwidth and virtual machine are important computing resources to provide software service to the user. Placing ST component on this cloud resource is one of the main challenges for managing resources and providing services in a dynamic cloud environment because ST components need an efficient and strategic placement process. In order to provide an optimal and cost effective service, it needs an efficient resource management mechanism on cloud resources.

Cost effective ST placement in a cloud environment has a lot of advantage to the SaaS provider that increasing the provider profit by minimizing the placement cost. It also gives the user reduced subscription cost and provides efficient service. So, optimal ST placement in cloud infrastructure becomes a critical problem in a cloud environment. The main aim of this thesis is to find an optimal placement area for initial ST placement problem in a cloud environment in order to address the challenge of initial ST placement problems.

1.1.1 Software Task Placement Problem in Cloud Environment

Placement is an important concept in cloud computing infrastructure. Currently, many services are placed in cloud environment to provide cloud service to the user. SaaS is one of cloud service model that gives different SaaS to the end user via the internet. At the initial stage of ST placement, the ST has to be deployed in virtual machine across the physical server. In this placement scenario, the placement process is done by the cloud provider. In the first stage of ST placement, the placement activity decision depends on the physical infrastructure resource like storage capacity, processing capacity, memory capacity input/output capacity and bandwidth. Figure 1.1, shows, the representation of initial ST placement scenario in a cloud environment.

The main aim of optimal placement in a cloud environment is to give efficient service with minimum cost to the end provider. To get an efficient SaaS, it needs strategical placement on cloud computing environment. The problem of placing ST component in a cloud environment is *ST placement problem* [2, 3, 4]. To handle this problem, it needs efficient placement approach, before placing ST component in a cloud environment. So, determining where and how to place the ST components makes the placement optimal and cost effective.

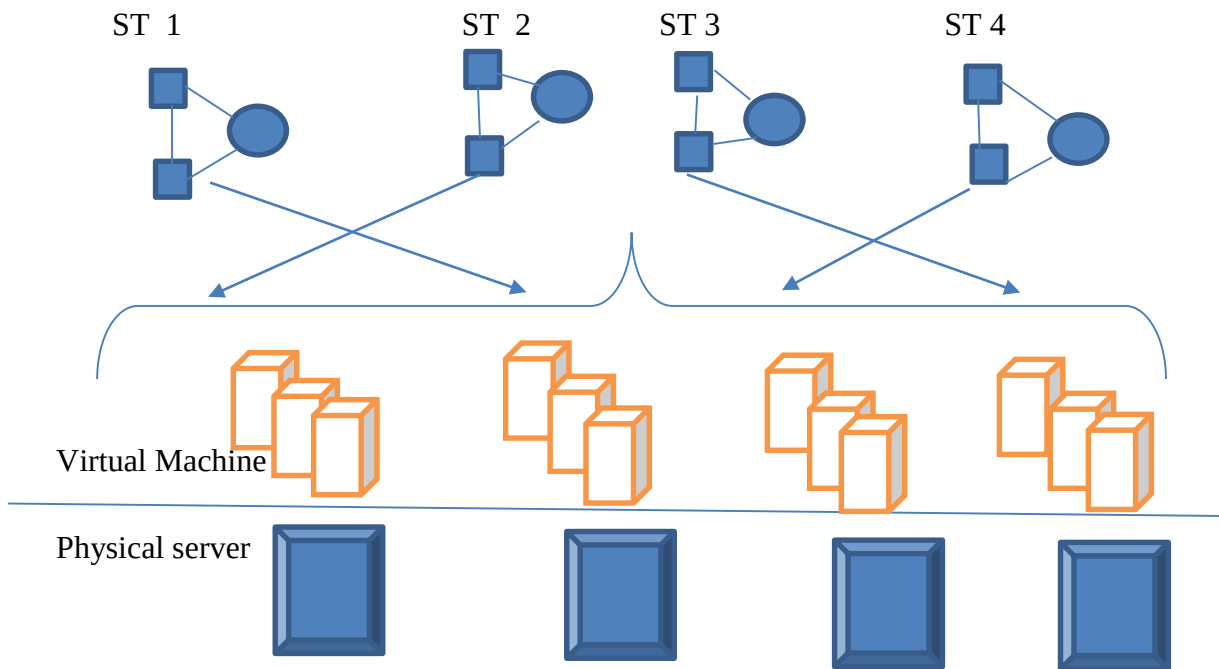


Figure 1.1: Software task Placement Scenario in Cloud Environment [2, 5].

1.2 Statement of the Problem

Software Task placement remains one of the research agenda in Cloud Computing arena. Among other things, determining an optimal placement is a challenging task. ST placement requires the highest cost as indicated on annual cloud reports [1]. These costs include infrastructure resource costs such as memory, storage, and a processing element. Deploying the ST component on these cloud resources, the provider needs an efficient strategy to optimize the infrastructure resources.

Amazon, Microsoft, and Google are currently among the well-known SaaS providers in a cloud environment. These providers host and provide their own SaaS services using Internet connections. For example, amazon is one of the main SaaS providers in the world. The AWS operates thirty-five availability zone with thirty geographical regions around the world. In the near future, it will have a plan to launch more availability zone and regional location [6]. In this dynamic cloud environment, the issue is managing the increasing number of heterogeneous DC resources to provide cost-effective services to the user.

Currently, researchers are attempting to find out the problems of placement. For instance, some researchers [2, 3, 4] focused on the selection of appropriate virtual machines, which they considered on performance ST execution and data transfer time of the ST placement on cloud infrastructure. This includes storage and memory from the located datacenter to improve the execution time of online placement of the ST. Other researchers also [7, 8] focused on the resource optimization problem on strategic ST placement in a cloud environment to optimize the infrastructure resources, which they are considered the selection of optimal virtual machine that increase the profit of the provider to meet SLA and tried to perform local optimization on located DC.

However, in those existing works, most of the current ST placement solutions placed the applications on located datacenter, usually considered storage, memory, processing element and bandwidth requirements. This work takes into account those requirement and incorporates transfer cost for optimal placement in located and across DC resources in order to provide cost-effective ST placement in this dynamic and heterogeneous cloud environment.

As a result, we proposed a new hybrid approach algorithm for ST placement in a cloud environment. The proposed solution is designed to search an optimal placement area in located and across DC.

1.3 Research Objectives

1.3.1 General Objectives

The main objective of this research is to study initial ST placement problem and to design hybrid approach for cost effective ST placement on a cloud environment.

1.3.2 Specific Objectives

Based on the above general goal in mind the research concentrates more specifically on:

- To Review the literature to assess what was done in this area and to see the gap from previous works.
- To study various optimization algorithms, in order to implement a scalable solution to SaaS placement optimization problem.
- To analyze different cloud computing simulation tool, in order to identify a suitable simulation tool to implement the proposed algorithm and solve the problem.
- To Design experiment setup environment using the selected simulation tool
- To test the performance of the proposed algorithm using different testing attributes.
- To conclude and give a future direction based on the results.

1.4 Research Methodology

The main aim of this research work is to address optimization problem of initial ST placement problems to provide cost effective service to the user. To solve the problem and achieve the research objective, the following six research phases are used throughout the research process as shown figure 1.2.

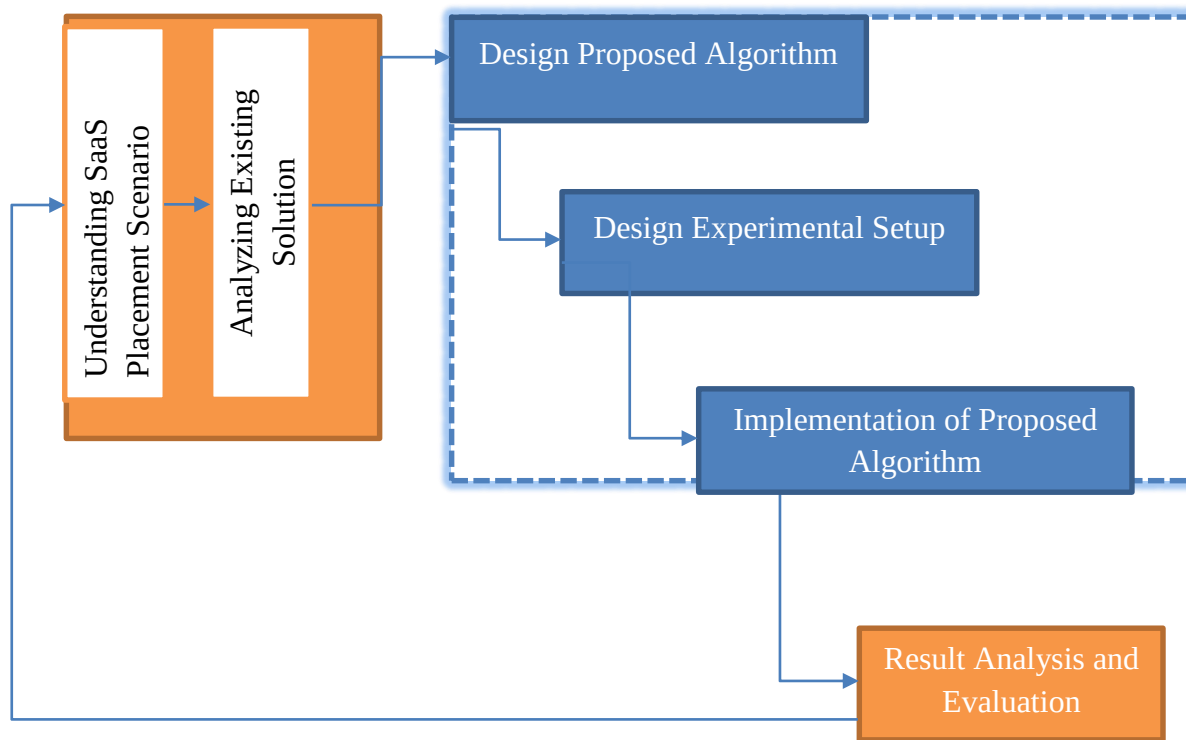


Figure 1.2: Research Method Phases Diagram

Understanding Basic Concepts of SaaS Placement Scenario

In this phase the existing placement activity on cloud computing environments is reviewed to understand the basic concepts of SaaS placement on cloud computing environment.

Analyzing Existing Solution

After understanding the basic concepts of SaaS placement in cloud computing, the next step has analyzed existing placement solutions in order to identify gaps and design the proposed solutions.

Design the Proposed Algorithm

The main aim of this paper is to proposed an efficient hybrid approach for cost efficient ST placement on cloud computing environment. In this phase, describes the major contribution of this research and tried to design hybrid approaches for cost effective ST placement in a cloud environment.

Design Experimental Setup

To implement the proposed solution first the experiment environment has been designed. In this phase, the study more about designing experiment setup using CloudSim simulation tools.

Implementation of Algorithm

After designed the experimental setup, the implementation of the algorithm for the experiments is done according to the objectives.

Result Analysis's and Evaluation

To evaluate the performance of the proposed algorithm; the results of the proposed solution has been analyzed using different placement evaluation criteria and the simulation result is compared with other existing placement algorithm.

1.5 Scope and Limitation of the Study

The main intention of this research is to develop an efficient algorithm for optimal placement of ST component in cloud computing environment. It concentrates only on the initial ST placement in cloud environment to provide a cost effective SaaS service to the user. In general, the study has focused on the following issues:

- Study initial ST placement problem in a cloud environment.
- Develop an algorithm for initial ST placement problem.

As the limitation, it is only analyzing some of the heuristic algorithms such as GA, PSO, and BA, to implement the ST placement optimization problem. In a cloud environment, different simulation and real testing environment are available to implement and test cloud scenario. But, the study investigates only some selected cloud simulation tools which are open source and implemented the proposed algorithm in CloudSim simulation environment.

1.6 Research Questions

Generally, the following are our research questions of this thesis

1. Is it possible to improve the performance of existing SaaS placement algorithm and can we design a better solution for initial ST placement problem in cloud computing?
2. Can a hybrid approach produce an optimal placement solution?

1.7 Significance of the Study

The result of this thesis lies in the area of placement for solving placement problem especially ST placement in cloud computing environment. Mainly, the main contribution of the study is an identification of the problem for initial ST placement to deliver cost effective service to the user. The outcome of this research can benefit entities that are involved in SaaS service including cloud infrastructure provider, Software service vendor, and Software users.

For cloud providers: The cloud infrastructure resource providers provide their resources to the Software vendor. This provider uses the infrastructure, platform, and SaaS service. The infrastructure provider must provide an effective and efficient application service to the user. In order to provide their service to the user, the application service needs an efficient placement. In order to provide cost effective service, the proposed efficient algorithm has optimized their resource in cloud infrastructure environment.

For SaaS users: Now a day, different SaaS application available in a cloud environment. These applications handle many users and a high number of the service request. The service process is costlier because it needs more cloud resources. Managing this resources, efficient placement strategy and managing cloud resources are the main issue to provide a cost efficient service to the user. The proposed solution will provide a cost effective service to the user by optimizing the cloud resources across data centers.

For SaaS vendors: Many cloud infrastructure providers have provided different resources such as memory, storage, and bandwidth. A SaaS provider rents these physical resources from infrastructure cloud providers and provides the service to the user by deploying their SaaS applications. The main aim of a SaaS provider is providing a cost-effective and efficient service to the user by using the resources efficiently from the infrastructure provider. Initial placement is the main issue to provide a cost-effective SaaS service. This proposed solution has provided an optimal initial ST placement in a cloud environment to the SaaS vendors.

1.8 Organization of the Paper

The rest of this thesis has organized as dedicated in (figure 1.3). The second chapter is dedicated to a detailed review of the conceptual literature and related works of SaaS placement problem in a cloud environment. In chapter three of this thesis, proposed solution for initial ST placement resource optimization problem has discussed. The detail experimental procedure and the implementation of the proposed algorithm has presented in chapter four. Chapter five covers discussion of analysis and evaluation results of the lab the experimentation. Finally, in chapter six general conclusions and future recommendations of the thesis has made based on results of the experiment.

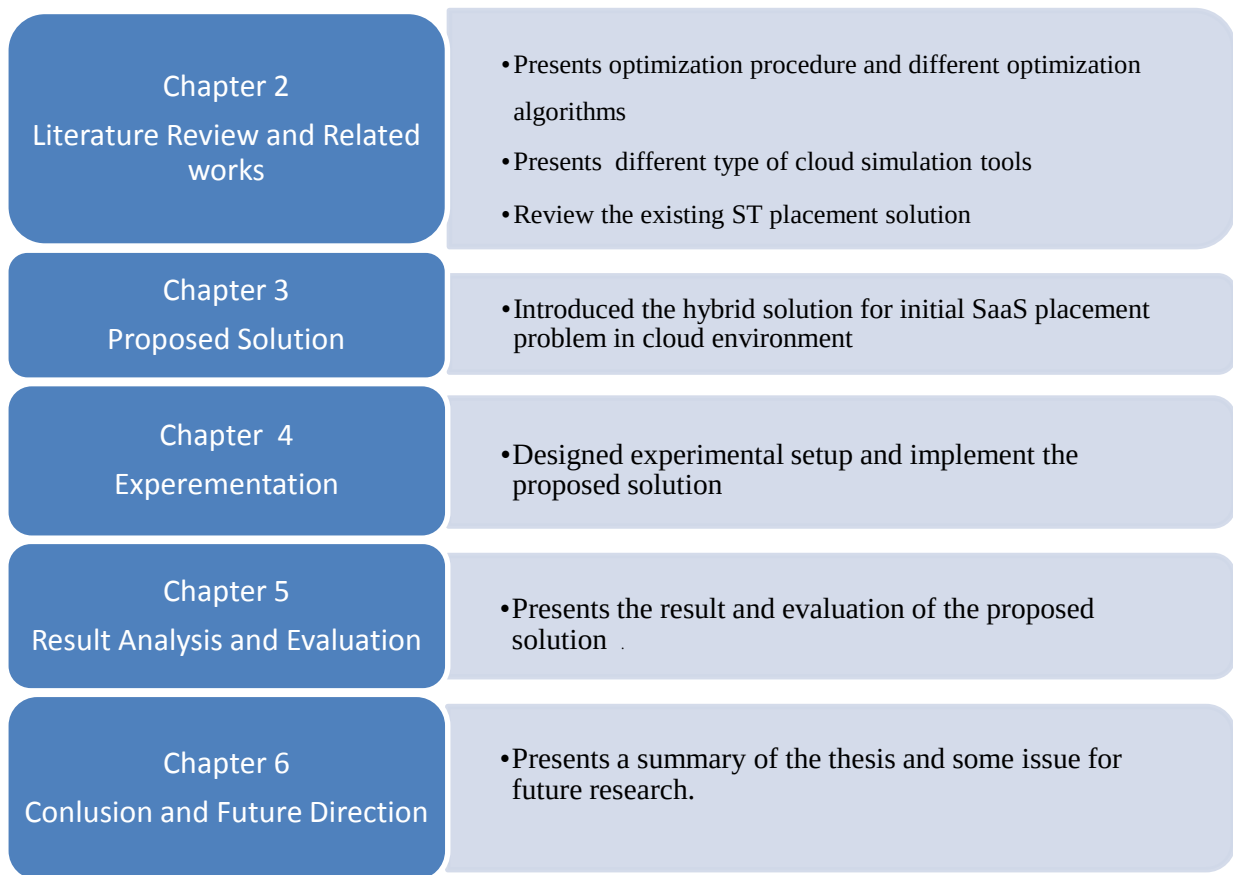


Figure 1.3: Thesis Organization

CHAPTER TWO: LITERATURE REVIEW AND RELATED WORK

2.1 Literature Review

2.1.1 Introduction

In Chapter 1, the problem, research objective and research methodology of this research has presented. This chapter is broadly divided into two main sections. The first section is about concepts related to cloud computing and Software as a service. In this section, the concept related to cloud computing and SaaS has reviewed to understand SaaS in a cloud environment. In the next section, recent related work of initial ST placement has reviewed to find out the gap. Finally, optimization procedures and different optimization algorithms has analyzed for a better understanding of the existing solution. Finally, different cloud simulation tools were analyzed in order to implement the proposed solution.

2.1.2 Cloud Computing

Different scholars and institutes the cloud computing arena has given their own definition for the term cloud computing. The following definition is cited from different experts and institutes:

According to National Institute of Standards and Technology (NIST) [9]: **Cloud Computing** is a model for allowing on-demand service to a shared resource of cloud infrastructure resources such as networks, servers, storage, applications, and different services provide to the user and manage the resource easily. Cloud computing [10] is a style of computing that provide services, such as infrastructure, platform, applications, and storage space. In order to deliver the service, cloud computing has used scalable and flexible capabilities.

Kumar, A. [11] Cloud computing is a system that containing a collection of different interconnected and virtualized infrastructure resources that are dynamically provisioned and presented as one or more computing resources based on SLA established between the service provider and consumers.

Currently, Cloud computing has great change to deliver different resource and services, that provide on-demand service to share the infrastructure resources in a self-service, dynamically scaled and managed manner, Cloud computing offers a lot of advantages in terms of cost, speed, and efficiency [12].

Cloud computing has three key technical characteristics [13, 14] including:

Scale: Cloud computing contain a various number of servers, providing processing elements using the concept of scaling. This scaling is very important to manage the demand. For example, cloud providers provide some services to the organization. This organization's computing resource demand will be high during some period of time. Based on the demand, the provider should easily manage the resource using this scaling characteristic of the cloud.

Flexibility: Cloud computing can control a large number of tasks, and adjusted to a real-time cloud environment. The main difference between cloud and traditional computing is that, the cloud provider can provide service based on the need of the user. For example, if an organization needs some servers one day and increase the resource demand in the next time, the cloud can handle such increase demand of the resources.

Efficiency: Is the third key characteristics of cloud computing. It provides huge resources of clouds for the end user with a minimum cost and high processing speed. Traditionally deployments require applications to be bound to a particular infrastructure. This results in low utilization, diminished efficiency, and inflexibility. Cloud brings in capabilities to allow applications to be dynamically deployed onto the appropriate infrastructure at runtime. This dynamic deployment behavior of Cloud computing allows applications to increase their efficiency.

2.1.2.1 Cloud Computing Service Models

Cloud computing service models is a service model describe how cloud services are available or provided to the users. Mainly in a cloud environment, there are three main division of cloud computing including IaaS, PaaS, and SaaS.

2.1.2.1.1 Infrastructure as a Service

The first deployment model (**IAAS**) provides the user access to different computing resources such as physical machines, virtual machines, virtual storage etc. And also the IAAS offer Virtual machine disk storage, Virtual local area network (VLANs), Load balancers, IP addresses by managing the infrastructure resources [15]. The main advantage of this service model is users can use the service without use own different cloud server and network materials. In this model, the user has controlled the operating systems, storage, and deployed applications. Based on the cloud computing advantage like elasticity and scaling techniques, the resources are able to increase (or decrease) requirements for computational capacity and provide different resources using pay-per-use models [16].

2.1.2.1.2 Platform as a Service

PaaS is the second service model [15] often referred as cloud ware, that gives an environment platform to develop the different application. In this cloud service model the user, not need software download or installation for all software user that found in the different area. The users do not manage or control the cloud infrastructure including network, servers, operating systems, or storage, but the users can manage the deployed applications and some configuration settings for the deployment environment and building infrastructure environment for their applications [17]. PaaS is very important where many developers will be working on the project or other users need to participate in the development process and the users don't want to the responsibility of managing operating system. Using PaaS services developers and users to customize software environments, and paid based on their use.

2.1.2.1.3 Software as a Service

SaaS is the third service model provided to the consumer to use the provider's applications running on a cloud infrastructure. Most of the time SaaS applications are accessible from various client devices via the internet. Today in cloud computing environment there are different SaaS applications like Billing and Invoicing System, Customer Relationship Management (CRM) applications, help Desk Applications, human Resource (HR) Solutions

[17]. In SaaS service model, the user does not control the cloud computing infrastructure including network, servers, operating systems, storage, and software provider (SaaS). But, the users can manage the application software. This model is important to the user because the user can use the application without pay installation, upgrading, and maintenance cost and only paid the cost based on the model of the provider, that means the users pay based on their use principles.

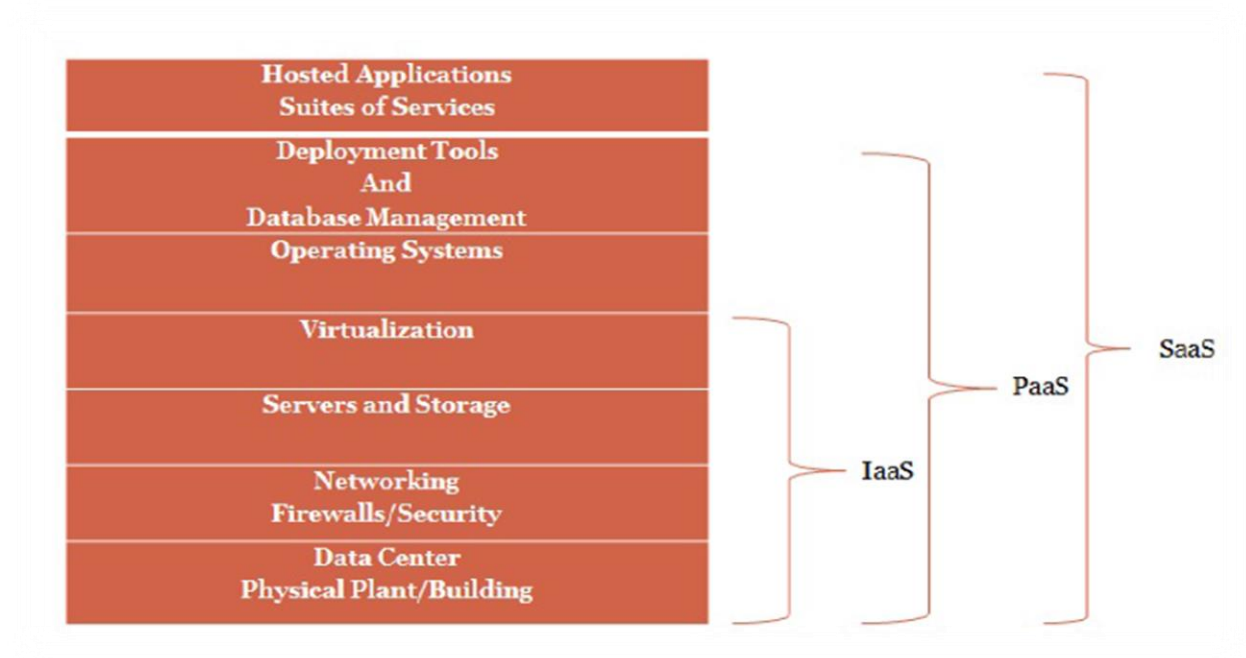


Figure 2.1: Cloud Service Model [11]

2.1.2.2 Cloud Computing Deployment Models

Cloud hosting deployment models [18] represent how cloud service are deployed and which are mainly differentiate by the proprietorship, size, and access. To use the cloud computing service, currently, most of the organizations have used different cloud deployment model. According to National Institute of Standards and Technology (NIST) [19], there are mainly four **cloud deployment models**, which are discussed below.

2.1.2.2.1 Public Cloud Model

The public cloud model is the first deployment model which the cloud services are delivered over a network which is open for public usage. This model is a true representation of cloud hosting; in this, the service provider renders services and infrastructure to various clients [17, 18]. The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds). Using multitenancy cloud characteristics, this cloud model provides efficient service by handles many users instead of a single user. The public cloud model is important for controlling different application like SaaS that used by many users.

2.1.2.2.2 Private Cloud

The second model is a **private cloud** [18] also known as the internal cloud. This model is important to create a secured cloud environment that is protected by different hardware and software tools. Private cloud is accessed only by authorized users, gives the organization greater and direct control over their data. Private cloud use by a single organization comprising multiple consumers. In this model, the infrastructure management is done by the organization, a third party, or some combination of them [17].

2.1.2.2.3 Hybrid Cloud

Hybrid cloud is the third type of cloud computing deployment model, which is combined with two or more cloud models, such as private, public or community cloud that is bound together. Benefits of the multiple deployment models are available in a hybrid cloud hosting. In a hybrid cloud, the resources are managed and provided by external providers. It is an adaptation among two platforms in which the workload exchanges between the private cloud and the public cloud as per the need and demand [18]. A lot of PaaS service provider use this model to deploy their application to get the advantage by integrating this hybrid model. But security is the main challenge in hybrid approach

2.1.2.2.4 Community Cloud

The final cloud computing deployment model in a cloud environment is **Community Cloud**. In this deployment model, cloud resource is provided for the user and used by some community of users from organizations that have shared resources. It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them [17]. This model is important to the user that are interested in working in a cooperative way and running the similar project. In this community deployment model, the resource of the cloud used by several organizations using same rules and agreement considerations. The infrastructure cost of community cloud is minimum when compared to other models like a private cloud [19].

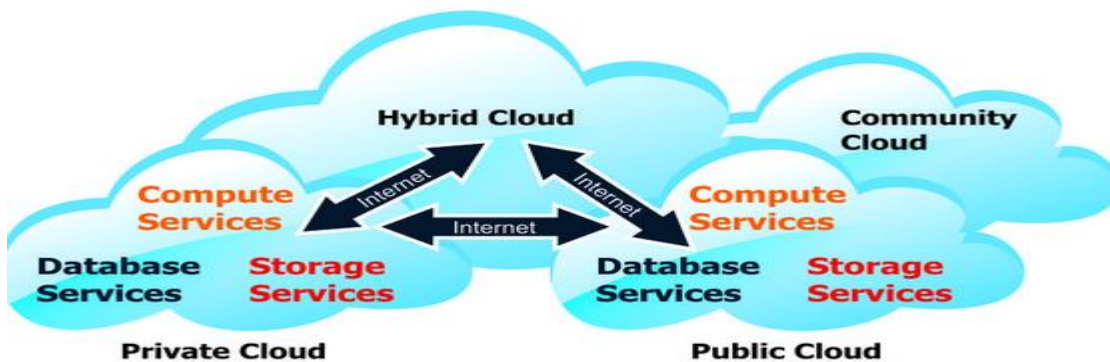


Figure 2.2: Cloud Deployment Model [20].

2.1.2.3 Virtualization

In cloud computing, virtualization is an important concept and it gives a lot of advantage to provide a huge and complex computing service using the internet. In a computing environment, virtualization is the process of creating a virtual infrastructure to manage and use the resource effectively such as cloud server, a storage component, and network. To create a virtualized resource, we need the hypervisor. This hypervisor is a software that is used to emulate the hardware resources. The idea of virtualization has been implemented in storage, networks, application and operating system. Commonly there are five types of virtualization process in cloud environment [21].

Storage Virtualization

Using network storage device, we can encapsulate a number of storage device in a single storage. In cloud computing storage virtualization are implemented via storage area network. Most of the time we use this virtualization, to control a collection of the resource from the central management console.

Server Virtualization

Server virtualization is the process of operating systems to use similar hardware infrastructure resource to move operating systems to different hardware when the applications are running. In this process, the server resources are hidden, that means the infrastructure resource is divided into different multiple virtual machines. The main aim of server virtualization is to increase the infrastructure computing resources sharing and utilization.

OS-level Virtualization.

Os level virtualization performed on at operating system level. The cloud physical resource and the operating system are virtualized in too many different partitions; each partition has its own physical servers. The virtualized kernel of operating system runs in located server.

Network Virtualization

A process of combining the infrastructure resources in a network for managing the available bandwidth. The idea of network virtualization is virtualization minimizing the network complexity by separating it into controllable parts.

2.1.2.4 SaaS in Cloud Environment

According to the Gartner Group [22] : SaaS is software that is controlled, delivered and managed centrally by the service providers. Based on the model, the providers provide service using a common language is metadata data, that are developed in a one-to-many model in order to deliver the service to the customers, at any time and places on a pay-for-use basis, usage metrics that are subscribed. SaaS is software [23] that's is deployed and managed by the provider and the customer accesses over the Internet. But, in traditional applications provider,

the software users use the applications on their computers or other devices, the SaaS provider controls the applications deployed in located datacenter.

In a cloud environment, SaaS is placed on the top of cloud infrastructure. The users are using this service using the service based model of the SaaS provider. SaaS applications also must be able to interact with other data and other applications in an equally wide variety of environments and platforms. SaaS is most often implemented to provide different business software applications to the various customer at a low cost while allowing those customers to obtain the licensed software everywhere [24].

Currently, different organizations are using SaaS because it's easy to use and cost efficient. The applications are deployed on located datacenter and used anywhere without buying hardware resources. In SaaS, the organizations pay a monthly fee instead of paying for licensed applications, Also, they are not paying installation, set-up, and maintenance costs. That means the user or the organization can use or access the service without payment of developing, implementing or supporting it. The users can access the application using internet services. SaaS has its own characteristics to provide efficient service to the user, this characteristic includes [25, 26]:

- **Configurability** - In this characteristics, the delivered software interface environment has its own configuration options such as network access, blog interface, Social Site (social media) extensions, and custom tool modules. At this level, many customers can use separate instances of the same application.
- **Multi-Tenant Efficiency** – In this architecture, a single instance of a software application is deployed to serve multiple customers. Because of multi-tenant architecture advantage, developers are able to use the same infrastructure and expand their services and make it economical and much easier for them to maintain this application. The user of the SaaS has used the single application using single provider site. Multi-tenancy refers to software architecture where a single instance of the software runs as a SaaS, serving multiple users in the organizations.
- **Scalability** – In this scalability characteristics, the load balancer has manage the services by controlling the cloud resources. A scalable architecture is supporting a load-

balanced activity of similar application instances running on a different number of servers. in cloud environment, the need of the resources changing dynamically. In this changing environment, this scalable characteristic is very important to manage the resources.

2.1.2.4.1 SaaS Service Providers

Today there are a lot of SaaS providers are available to provide different resources to the user. **Salesforce.com** [22] is one of SaaS service vendor that provide the service using Force.com platform in multi-tenant and scalable cloud computing environment. In this software provider, CRM is one of the main product that provides the management of customer information in the multi-tenant model. **Microsoft:** SaaS vendor [22] is also the second SaaS vendor and the main user of this Microsoft SaaS service are the customers who have used Microsoft's package software such as Windows and Microsoft office. Using the strategy of software and service, this service provider has delivered the web to existing software in comparison with the other vendors who provide their service through the network by using Web browser [22]. **Google** [22], [4] is the third SaaS provider and it has used its own service model to provide SaaS service to the user. Google Apps one of the main application that provided by Google and Google provide this service using its own integrated environment is called Google Apps.

Table 2.1: SaaS Service Provider [22]

Vendor	Service Description	Business Model	Origin	Strategy
Amazon	Computing Resource providing	Amazon web service	Web service	Service Infrastructure
Salesforce	Web-based CRM	Force.com	Web service/CRM	Platform as a service
Microsoft	Personal/office tools	Microsoft office live	Package software	Software +service
Google	Web office tools	Google apps	Web-based service	Service on the web

2.1.2.4.2 SaaS Deployment

SaaS is deployed in a cloud environment using infrastructure resources such as memory, storage, and processing element. the provider hosts the applications on the top of the virtual machine based on their architecture. SaaS deployment in a cloud environment requires a number of consideration to optimize the cloud resources. Identification of placement for initial placement and multitenancy is the main issue to provide an efficient service to the user. AWS is one of the SaaS service providers, that deliver scalable and cost-effective infrastructure resource in cloud environment. The following components are used by the provider to host and deliver the SaaS service [27].

Amazon Elastic Cloud (Amazon EC2). This component provides the manageable computing resources and the users can define the virtual environment. In this service amazon offers a full control of application environment console.

Amazon Storage Service (Amazon S3). Amazon S3 is the second amazon web service component, that used store and retrieve information from anywhere and anytime. This service also stores enormous data.

Amazon Relational Database Service (Amazon RDS). This amazon web service component provides an efficient database service to store and retrieve information in cloud environment.

Amazon Cloud Front. In this amazon service component, amazon provides the applications easily to the user by improving the transfer speed and minimizing the latency.

Amazon Simple Queue Service (Amazon SQS). This component provides the application services in a secure and easily accessible manner to the user in distributed environment.

Amazon DevPay. This service component manages the billing activity of the service to provide pay per use model to the user.

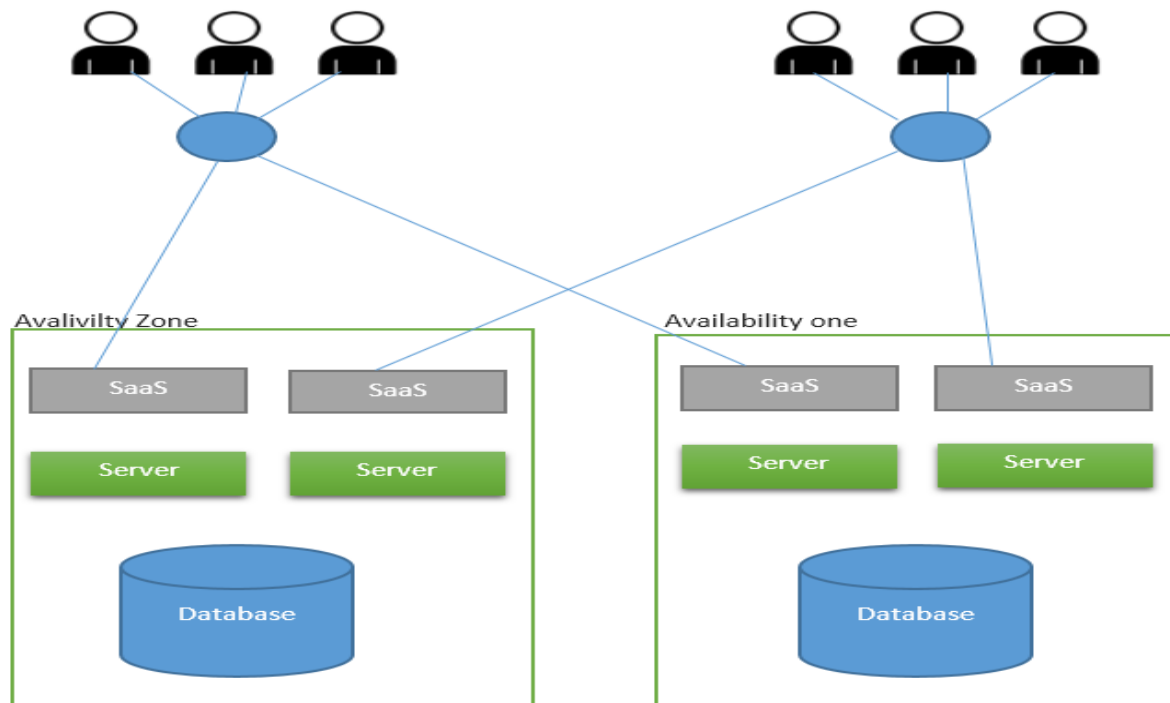


Figure 2.3: SaaS in Cloud Environment [28].

2.1.2.4.3 Application Delivery Model of SaaS

The delivery model of SaaS shows how software providers are providing their software product to the user. Software providers use their own model to provide their service to the user. Traditional software user uses the software by purchasing the software from a provider rather than use the software directly from the provider using the internet [29]. This traditional software provider has a greater advantage for software users based on the user interest, But if the user needs like software update and patch then software provider would need an extra charge of maintenance fee in addition to the cost of purchasing the software [25]. Security patches, application enhancements are some of the maintenance fees. There are a lot of SaaS service provider in the world and this SaaS provider delivers their application based on their model.

Currently, software services provide for many users using the concept of SOA [30]. The SOA service is used to communicate the software applications. A software application components are deployed in a cloud environment and accessed by various users in a different location. to

manage the communication of the software service the SOA artifact is important in order to provide an efficient and cost effective service to the user. The service model using SOA architecture has supports user need even the demand is high and the large numbers of transactions process in a secure and reliable environment.

2.1.2.4.4 Maturity Model of SaaS

Today many types of SaaS application are using a new model to improve the SaaS flexibility and reduced the cost incurred by the application provider. Using this model and application framework most of the application provider reduced the time and cost. According to Microsoft [24], SaaS architectures can be classified into one of four maturity levels whose key attributes are ease of configuration, multitenant efficiency, and scalability. This four level of maturity model such as *Ad hoc/custom, Configurable, Multi-Tenant, and Scalable*.

Level 1: (Ad-hoc / Custom)

This maturity model level represents traditional ASP model with customizing to each individual user. In this maturity model, there is no cloud computing characteristics advantage because the user has used its own application program and the applications run on the hosted server. In this level, it is difficult to manage many users at a time because there are no multiple tenants' concept and scalability issue in this maturity model [24].

Level 2: (Configurable)

In this maturity model level, the users can use the same application by separating the application instances. This is important to the SaaS provider to fulfill the need of customer by using configuration option and decrease the maintenance burden. In this level the maturity model does not contain the multi-tenant concept, each customer runs or uses its own applications. The separation of application instances can be either on the same machine (virtualization) or different machines [31] .

Level 3 :(Multitenant)

The third maturity level (**Multitenant**), that adds the multi-tenancy concept into the second level. In this level using a single application, the service provider gives the services for all users efficiently. This model [24] enables more efficient use of resources without any apparent difference to the end user, but in this level scalability issue is difficult.

Level 4: (Scalable)

Level 4(Scalable) is the highest level in SaaS maturity model, at the fourth level of SaaS maturity model, the maturity model contains all the capability of the above 3 level [24, 31]. In this maturity level scalability is implemented using scalable based multi-tenancy architecture. The architecture has important to manage application components placement on a cloud environment. This scalability issue is important to provide applications by balancing the infrastructure resources and to use the cloud resources effectively and efficiently. At this level, the architecture is scalable, multi-tenant and customizable.

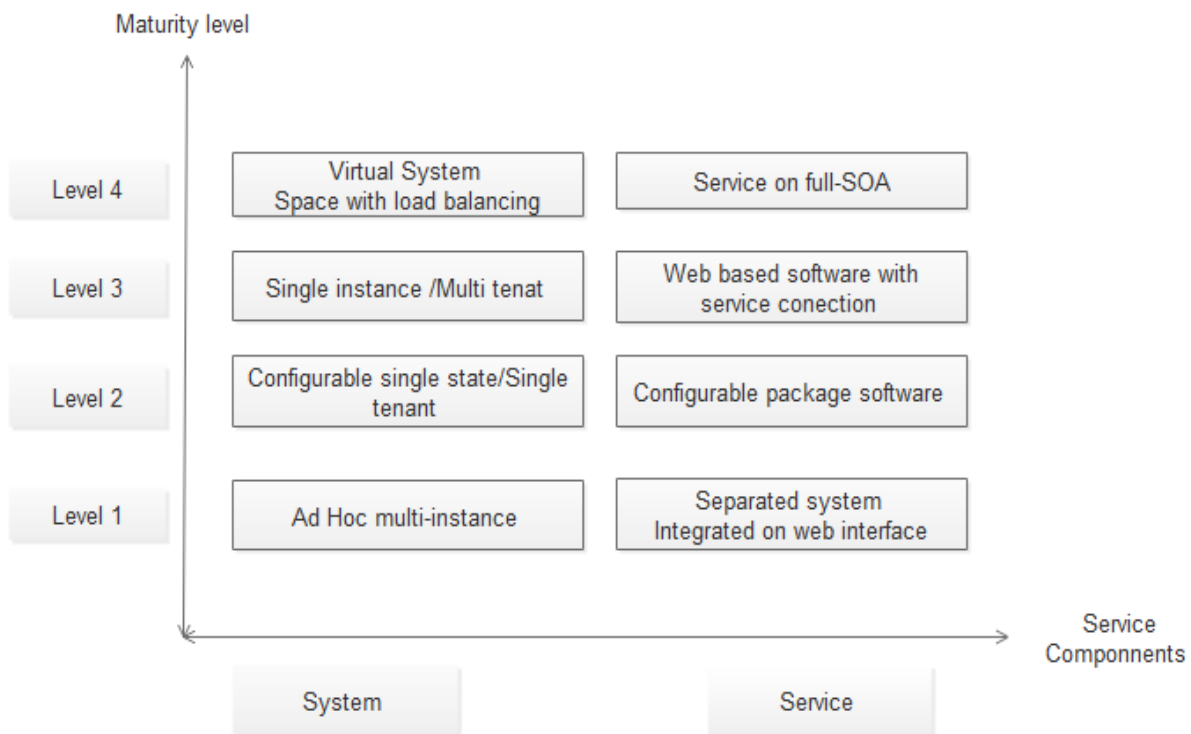


Figure 2.4: SaaS Maturity Model [4]

2.1.3 Optimization Algorithms

Today complex or hard problems have been solved by a different strategy. This strategy sometimes cannot have solved more quickly and efficiently. Optimization is the process of determining the best solution in a given feasible solution using their objective function [33]. Most of the optimization algorithms solve optimization tasks work based on their searching capability. Different engineering, scheduling, network routing and planning problems can be solved using optimization procedures. Optimization is a great tool to identify the best way to solve the problem. The main aim of optimization method is to minimize or maximize the objective function while satisfying the problem constraints [34].

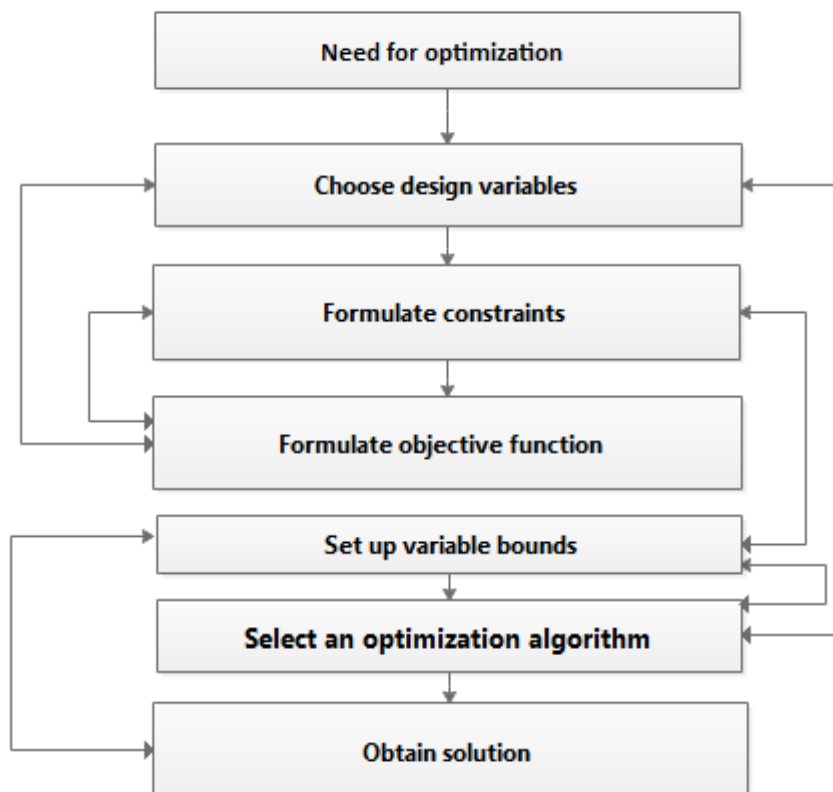


Figure 2.5: Optimal Design Procedure [34].

There are two types of optimization techniques; exact and heuristic [32]. Exact strategies produce the optimal solution, and work well for many problems. This technique is important when the search area is small. However, when the search area or the problem is large it is difficult to implement this strategy because it requires high computational cost and long time. In complex real world problem like placement problem in cloud computing environment we need different heuristic approaches in order to solve the problem in a reasonable amount of time. Heuristic algorithms solution is an optimal solution, when compare to the exact solution, because the exact solution need more time to find the solution, so for these complex problems a heuristic approach is appropriate. Heuristic algorithms refer to algorithms that produce high-quality results by trial and error methods in an acceptable computational time.

Most of the time Heuristic algorithms are generally inspired by nature, and these algorithms are also called nature inspired algorithms. These algorithms are used by many users because it's flexible and easy to implement in different optimization problem area. Single state and population-based algorithm are the two main classifications of heuristic algorithms [35]. The single state algorithms, Tabu search, and simulated annealing have a single candidate solution that is stored and used to compare against possible new solutions. Population-based algorithms store many candidate solutions and compare the solution to finding the best solution. When the searching area is large and for continuous space regions, single state algorithms are not very efficient because of the lack of comparisons between candidate solutions and are generally not as good at dealing with escaping local optima. In this case, population-based heuristics can be much faster. Evolutionary and swarm intelligent algorithms are the two main classifications of the population based heuristic algorithm.

2.1.3.1 Evolutionary Algorithm

In stochastic searching strategy, evolutionary algorithms are one of population-based heuristic algorithm. This evolutionary algorithm works based on natural biological evolution system. This biologically inspired algorithm proposed by Charles Darwin in 1859 [36]. The evolutionary algorithm works based on the survival of the fittest of Charles Darwin biological rules to perform different complex problems. Selection, crossover, and mutation are the main basic operator that are used to implement the evolution based algorithm. Selection is the first

operator that used to choose the best fitness value of candidate from the population. After selecting the best population, the next step is performing crossover operation to create a high fitness generation in a given population. The final evolution operation is a mutation. Sometimes the selection and crossover operators don't generate best fitness population, because of this reason evolution algorithm used a mutation operator created the best performance population in a given candidate that is not produced by selection and crossover genetic operators.

2.1.3.2 Swarm Intelligent Algorithms

Swarm intelligent algorithms are the second population-based algorithm. The swarm intelligent algorithm was introduced by Gerardo Beni and Jing Wang in 1989 [34]. These algorithms are inspired by social behavior of a group of an organism such as ants, bees, and birds. Swarm intelligence is governed by the rule of social behavior of a group of animal. The interaction behavior of animals is an important tool to communicate and gather information to each other.

This SI works based on two main principles such as self-organization and stigmergy [37]. The first rules self-organization deal about communication between component. This rule contains three parameters such as structure (the state of the swarm), multi-stability (the existence of the stable state of the swarm) and state transition (the behavioral change of the swarm). The next core principle of swarm used an environment is to save work state memory is called Stigmergy. Currently, SI algorithms are implemented in different complex optimization problem like routing problem in the network area, a scheduling problem, data mining area and other optimization problem area.

2.1.3.3 Genetic Algorithm

GA is one of the heuristic optimization algorithm based on natural evolutionary process [38]. GA algorithm works initially generated a group of pupation randomly. The generated population are evaluated based on fitness value using the objective function. After evaluating the fitness value of each generated population, the next procedure is to select the best fitness value among the generated population. To get the most fitness value or easy to converge local or global optima the GA algorithm use crossover and mutation operators. These operators are the most powerful operators to find the best solution in short period of time.

Crossover: Crossover is the first genetic operators. This operator selects genes from parent chromosomes and creates a new population. In order to create a new best population, this operation has done by choosing crossover point randomly.

Mutation: After a crossover operation is performed, mutation takes place. This is important to converge the solution easily into a local optimum. This operator changes the solution randomly

```
Choose initial population
Evaluate the fitness of each individual in the population
Repeat
    Select best-ranking individuals to reproduce
    Breed new generation through crossover and mutation
    Evaluate the individual fitnesses of the offspring
    Replace worst ranked part of population with offspring
Termination condition
```

Figure 2.6: Genetic Algorithm [38]

2.1.3.4 Particle Swarm Optimization Algorithm

The Particle Swarm Optimization algorithm is natural -inspired algorithm motivated by a social behavior of a group of birds [39]. The PSO algorithm is population-based heuristic algorithm: a set of potential solutions evolves to approach a convenient solution (or set of solutions) for a problem. The main aim of PSO algorithm is to solve the optimization problem by finding the global optimum in a given search space. This algorithm follows a certain rule to find a good result. The rules are [32].

- The searching environment (its fitness value)
- The previous state of the swarm (its memory)
- The neighborhood history of the swarm individuals

In PSO algorithm each individual particle is distribute into different of searching space and particles contain its own previous information to identify the best position. Using on previous placement position each particle has shares information on their neighborhood. The PSO algorithm is generated population particles randomly in a given searching space to find an optimum solution with a group of random particles (solutions) and then searches for an optimal solution by updating groups. In each iteration, PSO algorithm used two main best value. The first fitness value is pbest value. This value is called pbest. Another "best" value that is identified by the particle is the best value, obtained so far by another particle in the group. This best value is gbest. A particle fitness value is best from neighbors; this best value is a local best or lbest.

```

For each particle
    Initialize particle
END
Do
    For each particle
        Calculate fitness value
        If the fitness value is better than the best fitness
            value(pbest)  in history
            set current value as the new pbest
        End
        Choose the particle with the best fitness value of all the
        particles as the gbest
        For each particle
            Calculate particle velocity according equation
            Update particle position according equation
        End
    While maximum iterations or minimum error criteria is not attained

```

Figure 2.7: Particle Swarm Optimization Algorithm [32]

2.1.3.5 Ant Colony Algorithm

The ant colony algorithm is a heuristic optimization algorithm for finding optimal paths that are based the searching behavior of ants [40]. The first Ants randomly woke at the first iteration.

When ant searching a source of food, it finds back to the colony leaving "path " (chemicals) that show the path has a source food. Based on the first ants' other ants come across the path, they are likely to follow the path with a certain probability. If they do, they populate the path with their own markers as they bring the food back. As more ants find the path, it gets stronger until there are a high number of ants traveling to various food sources near the camp. The ants use some special chemicals to show the path easily the next ant. Other ants use these chemicals to follow the first ants. This process has continued until the best path has been found by the ants. In this process the ants easily found the solution because the ants use special pheromones every time. ACO algorithm implemented in different optimization area of such systems include computer networks, and artificial intelligence simulations area.

```
initialize the pheromone trails and parameter
initialize the population of the solution
for each individual ant calculate the fitness
for each ant determine the best position
    determine the best global position
    update pheromone
    check the termination
```

Figure 2.8: Anti-colony Algorithm [40].

2.1.3.6 Greedy Algorithm

A greedy algorithm is the most known method to solve the different optimization problem. Many recent heuristic algorithms have been developed based on greedy approaches. The main aim of greedy algorithm is to find a reasonable solution using stochastically searching strategy to find local optimal solution easily. A GA approach always makes the choice best fitness value at the moment [41]. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. A greedy algorithm obtains an optimal result to a problem by making the selection strategy is sequential. For each stage of the algorithm, the algorithm choice the current best solution. This GA algorithm sometimes not found an optimum solution, but this algorithm is more effective like selection and routing problem. This approach is implemented in different optimization problem areas like routing problem in the networking

area, artificial intelligence area, and game application. To solve the problem, the greedy approach divides the problem into subproblems in order to get the optimal solution. It also uses the fitness function like another heuristic approach to evaluating the performance of the solution.

```
N input of the solution
initialize the solution
for each solution
    select the solution
    if the solution is feasible
        take the solution
    end if
return the solution
```

Figure 2.9: Greedy Algorithm [42]

2.1.3.7 Bat Algorithm

Bat algorithm is heuristic-based optimization algorithm used to solve the different optimization problem in the different area. This algorithm uses an echolocation concept searching strategy that inspired by natural bat [35]. To search the target location, BA use pulse rate echo location to identify their obstacles and target location. First, Initialize the bat population, the pulse rates, the loudness, and define the pulse frequency. After initializing the bat population, update the velocities to change the location of the bats, and decide whether the current position is optimum or not. If the current position is not optimum, the bat population search another location in order to identify the current best solution found by updating the loudness and the emission rate control attributes. Finally, Check the termination condition to decide whether go back to the first step or end the process and output the result. The BA algorithm is implemented in different real-world problems and more powerful when the searching dimension is large.

```

initialize the bat population and attribute
Generate new solutions by adjusting frequency,
and updating velocities and locations
    if (Rand (0,1) > ri)
        Select a solution among the best solutions
        Generate a local solution around the selected best solution

    end if
    Generate a new solution by flying randomly

    if (random pulse rate < Ai)

        Accept the new solutions
        Increase ri and reduce Ai

    end if
    Rank the bats and find the current best

```

Figure 2.10: Bat Algorithm [35]

Table 2.2: Comparison of Optimization Algorithms

Optimization Algorithm	Advantage and Disadvantage	Application Area
Genetic algorithm	• It is evolution based optimization algorithm • There are multiple local optima and local searching capability is very high	• It is implemented in different artificial creativity, Control engineering, and scheduling application
Particle swarm optimization algorithm	• It is population (swarm based) optimization algorithm • It has contained less control parameter • Global searching capability is high • It is inexpensive in terms of computation time and memory	• This algorithm implemented in different area such as in neural networks, power control system, and scheduling problems area.

Anti-colony Algorithm	• It is population (swarm based) optimization algorithm • Has strong robustness • Converging into local optima is very low (slow to converge to local optima)	• This algorithm implemented in different areas such as in power diagnostic and data Ming application area.
Greedy Algorithm	• It is first in first out based optimization algorithm • It usually requires sorting the choices • Greedy algorithms don't work for some other complex problems	• This algorithm implemented in different areas such as in networks routing and path selection problems.
Bat algorithm	• It is population (swarm based) optimization algorithm • It is more powerful to find global minimum. • Easily solve constrained based optimization algorithm	• This algorithm implemented in the different complex areas such as image processing, routing problems, and engineering design.

Generally, most of the above optimization algorithms are inspired by biological and swarm behavior of nature and implemented in the different real-world optimization problem. The performance of each optimization algorithms is different by their optimal solution for the objective function. In this thesis, hybrid approach optimization algorithm is proposed using the advantages of both BA and GA to produce an optimal solution for initial ST placement problem in a cloud environment. The major weakness of BA is, the searching capability of to find an optimum solution in a small searching area is not efficient. But, this algorithm is to converge easily into optimum solution when the searching environment is large. The second GA algorithm, convergence capability to find the local optimum solution is high. But, the global searching is not optimum when compared to BA. So, to improve the performance of existing placement algorithms, we used the hybrid approach.

2.1.4 Cloud Simulation Tools

Currently, a lot of simulation tools have been available for modeling and simulation cloud computing scenario. Modeling and simulation technologies are important and suitable for evaluating and testing the real cloud environment area. These simulation tools are important to test or emulate cloud computing environment that is difficult to test in real simulation area. Because testing simulation activity in real world environment is very costly. In this section, different cloud simulation tools were analyzed in order to identify the best tool to implement the proposed solution. These simulation tools include CloudSim, Green cloud, and Datacenter Simulation Tool.

2.1.4.1 CloudSim

CloudSim is one of the cloud simulation tool that developed in clouds laboratory in the university of Melbourne computer science and engineering department, Australia [42, 43]. CloudSim has contained the basic class that used to manage different cloud-based simulation activity. Currently, a lot of could base projects are simulated by CloudSim because this simulation tool is easily model such as datacenters, virtual machine allocation policy, broker policy and it is also extended by another simulation tool like cloudAnalyst. when compare to another simulation tool, CloudSim is a powerful simulation tool that creates 10,000 virtual machines in a few minutes [43]. The following figure 2.5 has shown CloudSim architecture [44].

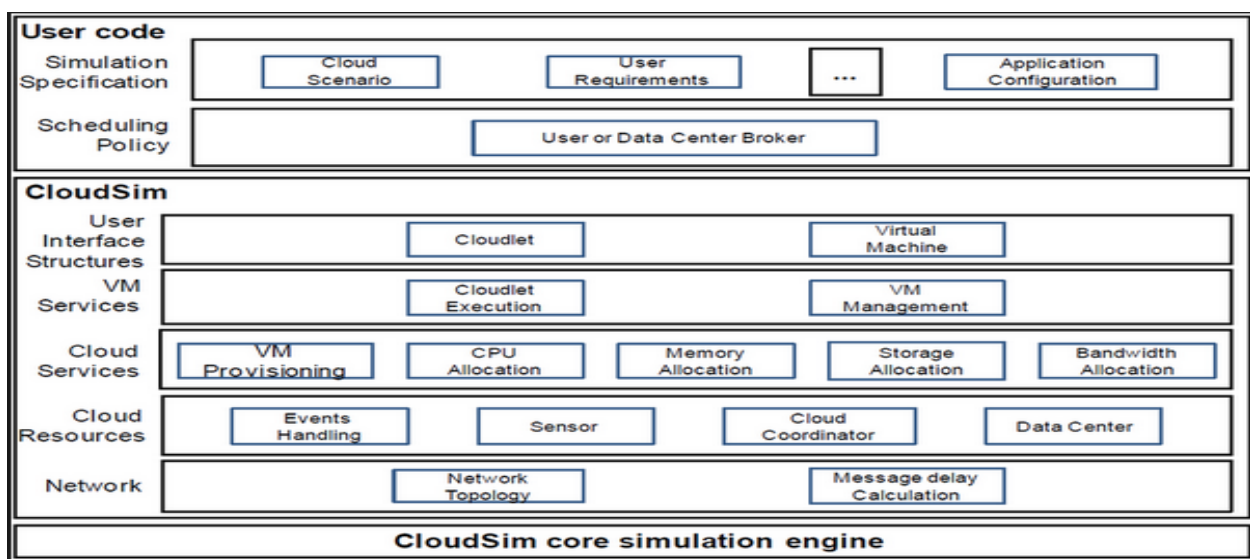


Figure 2.11: Architecture of CloudSim [44]

The above figure 2.5 shows the layered architecture of CloudSim. The architecture has contained three main layers. In the first layer (user code layer), can develop their own custom cloud scenario, defined user own requirement, application configuration and datacenter broker policy. The second layer CloudSim layer is the main simulation layer that the user access different CloudSim services like user interface, VM service, cloud service, cloud resources and network. Finally, the simulation tool has contained the core simulation engine that used to communicate the whole simulation services.

2.1.4.2 Datacenter Simulation Tool(DCSim)

DCSim is an event driven simulator tool that developed using java to simulate infrastructure as services of cloud [44, 45]. This simulation tool more focused on creating virtualized datacenter to control and analyses the performance of datacenter management. The virtualized datacenter has contained hosts and VM resource. Each resources are managed using different scheduling and management policies.

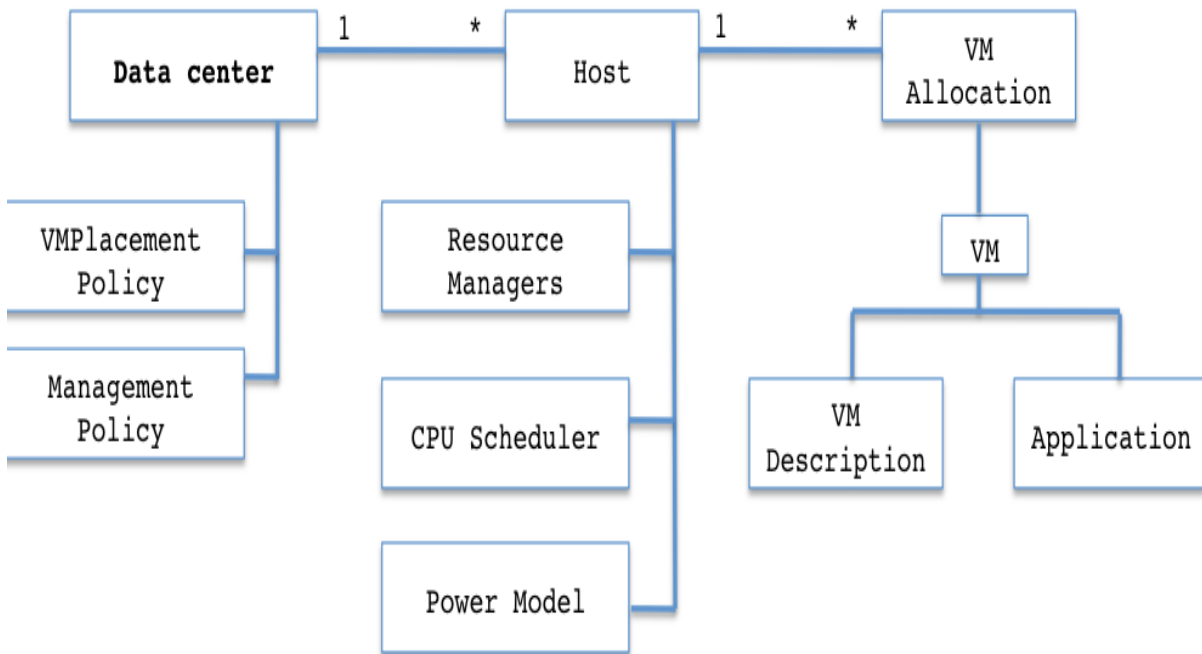


Figure 2.12 *Architecture of DCSim* [44].

DC, Host, and VM are the main components of datacenter simulation tool. DC used to manage the host component using allocation and management policy of the datacenter. Each datacenter has to contain hosts which are controlled using resource manager policy and scheduler algorithms. The hosts are created a VM based resource managers' module policy and also assign the CPU resources based on the application resource workload

2.1.4.3 Greencloud Simulator

Greencloud is a packet-level simulator that extends in CloudSim and more focused on cloud communication like switches, servers and communication links. Greencloud simulator is an open source developed by using C++ and runs on windows environment. This simulation tool used to measure the energy consumption the datacenter and communication pattern between Greencloud and more important to energy aware study in cloud computing environment [44].

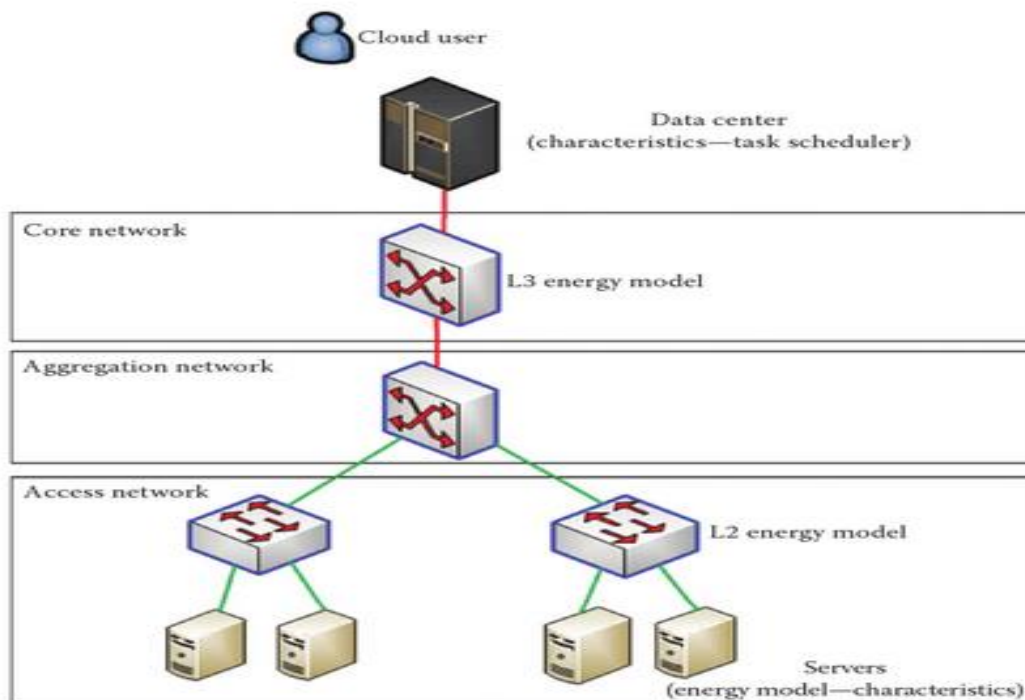


Figure 2.13: Architecture of Greencloud Simulator [45]

The Greencloud simulation tool designed based on three-tier architecture structure [45]. The main component of this architecture is servers, switches, and workloads. The server is the first simulator component; it is important to manage the execution application. The second component is workload, that used to manage different users' services like content delivery and instant message. The switch is the last component of the simulator that is responsible to for controlling the workload and message communication between any two servers.

Table 2.3: Cloud Computing Simulation Tool

Simulation Tool	Program	Operating System Support	Properties and Limitation	Availability
CloudSim	Java	Any platform	➤ Provide core simulation class for manage datacenter components ➤ It performs huge simulation activity ➤ Support complex scheduling algorithms ➤ Many simulation tools develop on the top CloudSim class	Open source
DCSim	Java	Any platform	➤ Event driven based simulation tool ➤ More focused on virtualized datacenter environment ➤ It is important to design the transactional workload	Open source
Greencloud	C++	windows environment	➤ Packet level simulation tool ➤ It is focused on cloud communication ➤ It is important to measure the power consumption and load of datacenter	Open source

Table 2.3, shows the analysis and comparison of three cloud computing simulators, such as CloudSim, DCSim, and Greencloud based on language, operating system support, properties and limitation, and availability. Most of these simulators are used to simulate cloud-based simulation scenario and developed using Java. Each simulator tool has its own advantage and limitation. To choose the suitable simulator tool the user must be depending on their requirement and need. However, our problem is more focused on the management of cloud resources. So, CloudSim simulation tool was selected. Because this tool has support easily management of infrastructure resources to place ST on cloud environment using the CloudSim class. Generally, CloudSim is popular and many simulation activities use and also other cloud computing simulation tools such as CloudAnalyst and Networkcloudsim are developed by extending the simulation class module of CloudSim.

2.2 Related Works

In this section, existing work has been discussed that related to ST component placement optimization problem in cloud computing environment.

Cloud computing provides to the user to use and deploy their applications and data from anywhere that are place in a cloud environment. Placement of data in cloud environment needs efficient and strategical placement to provide efficient service to the user. Currently, an enormous amount of data and ST placed across each data cloud data center. The SaaS placement problem is a new challenge in placement scenario in cloud computing environment. In this placement scenario, ST must placement must be optimum. There are a lot of studies tried to addressed related online placement problem in a cloud environment, but few number of research were addressed initial ST placement optimization problem in cloud computing infrastructure.

According to the study of Zeratul Izzah et al. [46] , that addressed SaaS placement problem in cloud datacenter with the objective of the study has to find a suitable placement server in datacenter for composite SaaS that satisfied the performance and to find the best placement. The researchers had been proposed a penalty based genetic algorithm to improve the performance SaaS service. To improve the performance of the SaaS service, they used estimated execution and data transfer time between cloud servers for effective SaaS placement in cloud computing environment.

Zeratul Izzah et al. [3], addressed SaaS optimization problem by proposing an evolutionary algorithm (genetic algorithm). The authors proposed PGA in previous research works by focusing on identifying the suitable server to improve execution time performance. Now they focused on SaaS resource management on cloud computing datacenter environment to minimize the resource optimization problem of SaaS placement. They tried to solve the problem focused on SaaS constraint, requirement and interdependency the SaaS service in a cloud environment. To solve the SaaS placement, the authors used resources management and maintenance window to optimize the cloud computing resource using this resource management window by assigning the resources based on the datacenter need without affecting the performance of SaaS components. To solve the problem, the paper focused on the placement problem at application level, instead of platform level (virtualization level). In this

study, the proposed algorithm has randomly generated the initial task population and virtual machine resources. After generating the resource, the algorithm places the generated population randomly virtual resources and calculating the placement cost by checking the placement constraint problems. Moreover, the GA algorithm has controlled the placement constraint virtual machine such as memory, storage and bandwidth problems to assigned the resource in a cloud environment.

Alok [11] addressed SaaS placement problem in a cloud environment. This thesis more focused on how a SaaS component is placed on a virtual machine to optimize its performance while satisfying the SaaS resource and response time constraints. The researcher presents SaaS placement problem as an optimization problem and proposed the genetic algorithm to improve or maximize the profit of the SaaS providers. To place the task components, first initialized the infrastructure and task resources using the proposed GA algorithm. And check placement constraints before placing the task component across virtual machine resources. To identify the optimum placement location, the researcher only considered the rental cost of the virtual machine and the revenue cost in order to evaluate the profit of SaaS provider. To evaluate the performance of the proposed solution the researcher tried to compare the GA algorithm with first fit randomization algorithm.

In a study [7] , the author stated that SaaS placement problem is combinatorial optimization problem, from a computational point of view this problem known as Np- hard problem. The main objective of this paper had been to minimize the cost of SaaS placement that incurred by SaaS provider. The author of this study was addressed the initial SaaS placement on cloud environment considering mainly the cloud resources, like processing capacity. memory and secondary storage, it is also considered SLA constraints to know the cost of SaaS resource and the profit of cloud provider. The researcher had been proposed that mathematical model using particle swarm optimization approach (PSO) for efficient SaaS placement on cloud computing. In this work, the proposed algorithm has generated the task and virtual machine resources randomly like a genetic algorithm, while in particle swarm optimization algorithm, the placement initial performed by comparing randomly selected SaaS components with the local best SaaS component using placement cost (virtual machine cost). After comparing the randomly selected SaaS component, the next step is to find a global best placement position so

far by local best position. After that, the algorithm updates the placement position using this two attribute (local best placement and global best placement). Finally, the researcher has tried to compared the performance of proposed algorithm with another heuristic algorithm (greedy algorithm) and, they said that the proposed algorithm is scalable.

The study of Zerzeberg et al. [2] addressed that composite SaaS placement problem by proposing a mathematical model using anti-colony algorithm. The main aim of this paper has to minimize the total execution time of the data and application placement on a virtual machine in datacenter. To achieve their objective, the researchers focused on SaaS requirement such as computational capacity, memory requirement, and storage requirement at the application level. In this study, the proposed algorithm, that is the first placement each component of SaaS component placed randomly selected virtual machine finding the good path by calculating data transfer time and component execution time. After finding the best placement path, the next process is select a suitable virtual machine based on placement selection probability.

The study of Madalalina [8] tried to address resource management problem that focused a single application placement in the single virtual machine by proposing a solution which is deployed a different application that satisfies placement constraint problem in a cloud environment. In this thesis, the researcher used three main criteria to place applications on cloud resources. The first rule is each application instance placed at list on one virtual machine. The second rule is the conflict application is not deploying on in the single virtual machine and the virtual machine must host one application instances. To satisfy the above main placement constraints the researcher has proposed exact and heuristic based approaches. In the first searching mechanism exact approach has used when the searching space environment is small (e.g. The number of application minimum). But, the application component and the searching infrastructure environment is large, the population-based searching using GA was used.

Table 2.4: Summery of Related Works

Researcher	Proposed Solution	Drawback
Zeratul Izzah et al. [46]	The researchers has proposed penalty based genetic algorithm to improve the performance SaaS service	- They only considered service execution time (response time of the service). - It is only identified suitable server to execute the task component.
Zeratul Izzah et al. [3]	Proposed evolutionarily algorithm using management and maintenance window to optimize the cloud computing resource	- At initial placement stage, the author placed many service components in a virtual machine to minimize the cloud resources. - Focused on a number of tasks to place on virtual machine resources.
Alok [11]	Proposed genetic algorithm for SaaS placement	- The main aim of the researcher was increasing the profit of the SaaS vendor. - Only considered the rental cost of virtual machine and revenue cost.
Bibhudatta and Bhardwaj [7]	The paper proposes particle swarm optimization approach (PSO) for efficient SaaS placement on cloud	- They are considered mainly service level agreement constraint to provide cost effective software service. - They focused on placement cost including processing cost, memory cost, storage cost and bandwidth cost.

Zerzeberg et al [2]	• Proposed Anti-colony algorithm to minimize the execution time.	• He tried to identify the powerful virtual machine to improve the response service time. • To identify suitable virtual machine , it only considered data transfer time between virtual machine and the processing capability of selected server
Madalalina [8]	• Proposed exact and heuristic approaches to solve placement problem on selected datacenter resources.	• The authors only considered the number of tasks and the size of searching placement environment. • It only performs local optimization across virtual machine on selected server, not across physical server of cloud resources.

In related work, the placement algorithms make use of infrastructure resource, usually considering memory storage [2, 3, 46] and bandwidth resource in located datacenter [7] by proposed algorithm that is the convergence capability either global optima or local optima are high. Selection of optimum virtual machine to increase the profit of the vendor by proposed GA algorithm, that is the local convergence capability is high [11]. In order to fulfill the above-mentioned gaps, in this thesis, we have proposed an efficient placement strategy by proposing hybrid algorithms such as BA and GA using the performance of global and local convergences capability considering both infrastructure resources cost and transfer cost for cost effective ST placement in a cloud environment.

Summary

This chapter mainly discussed the detail literature survey on concepts of SaaS placement, optimization algorithms, cloud simulation tool and also presented the related work. In the first literature section, we discussed cloud computing deployment, service model, identifying maturity level of software service model and show how it works in a cloud scenario. Beside this, it's more discussed concepts and types of optimization algorithm by dividing into two categories such as swarm based and evolutionary based algorithm, that is important to tackle different optimization problem. Moreover, in this chapter different cloud simulation tool were analyzed using different evaluation criteria like a programming language, availability, and platform support to select better tool to implement the proposed algorithm. Finally, the related works have reviewed that are related to ST placement problem to identify the gap. The remaining chapters have explored about finding the solution to fill the gap that is presented in chapter 1. Additionally, the next chapter of the thesis has proposed the HA solution for initial ST placement problem in the cloud.

CHAPTER THREE: PROPOSED SOLUTION

3.1 Introduction

In this chapter, the proposed solution is presented in order to optimize the placement of ST in a cloud environment. The main objective of this paper is to design algorithm for initial ST placement in a cloud environment and as presented in Chapter1, the ST placement has been improved with the use of hybrid approach. Therefore, this chapter has designed the proposed solution by hybrid the evolutionary algorithm of GA with the swarm based BA algorithm for efficient initial ST placement in a cloud environment by using the advantage of global and local convergence capability BA and GA respectively. In the first section of this chapter, the introduction of the proposed solution has present. In the second section, the proposed solution system architecture has present. In the third section, the representation of ST placement in the cloud environment has presented. In the fourth section, the structure of the proposed solution and the strategy to implement the proposed solution has present.

3.2 ST Placement

Optimal placement is the process of finding an efficient placement area in a given infrastructure. Efficient optimization algorithm needs to use cloud resources effectively and efficiently. SaaS is one of the main cloud computing that is delivered by different cloud providers. Placement of data or Tasks in large and complex cloud environment, especially SaaS in distributed cloud environment has been complex and resource intensive. Because ST components need more resource. Efficient initial placement is the main issue to make the SaaS service optimum. Identification of optimum placement location by handling the placement constraint is the crucial activity in SaaS placement in a cloud environment. As reviewed in the literature section, the previous placement solution was considered the placement activity in located datacenter and more focused on finding the solution by proposing optimization algorithm that has the local convergence capability of the algorithm is efficient. To make the placement activities are optimum, it needs an efficient algorithm. In this thesis, hybrid approaches were proposed for cost effective placement in cloud environment by using both local and global convergent capability of the proposed solution

3.3 ST Placement Representation Model Across Datacenters

To model the representation ST placement in a cloud environment, the first step is identifying the resource and its property of cloud infrastructure. The second step is identifying the decision variable that used defend the objective function. The third step is formulating the objective using those decision variables and finally, the obstacles are defined, that are the constraint to optimizes the objective function. Thus, representation and objective function formulation for initial ST placement are represented as follow.

To place P number of ST component $P = \{P1, P2, P3 \dots Pn\}$ and its requirement $Preq = \{\text{memory, storage and cpu}\}$ on D number of datacenters $D = \{D1, D2, D3 \dots Dn\}$. Those Datacenter contain H no of virtual machine $H = \{H1, H2, H3 \dots Hn\}$. Each datacenter virtual machine has its own resource element such as memory, storage, and a processing element. The representation of ST component across datacenters are described below.

$$F(Px) = \{Px1, Px2, Px3 \dots PxN\} \dots \dots \dots (3.1)$$

$$\text{Where } \left\{ \begin{array}{l} Px1 = ST \text{ placement on datacenter 1} \\ Px2 = ST \text{ placement on datacenter 2} \\ \vdots \\ PxN = ST \text{ placement on datacenter N} \end{array} \right.$$

The main aim of this thesis is to minimize a placement cost including transfer cost and virtual machine cost that fulfill placement constraints. The transfer cost is the cost of bandwidth where i component ST transfer from one located virtual machine resource to the other VM resources.

$$Tc = TSize * BWcost \dots \dots \dots (3.2)$$

$$\text{Where } \left\{ \begin{array}{l} TSize = Total ST size \\ BWcost = Bandwidth cost \end{array} \right.$$

The second placement cost is virtual machine cost. This cost is the resource cost including memory cost, storage cost and processor cost that is needed to place the ST components in a virtual machine [5].

$$Vc = MMcost + STcost + CPUcost \dots\dots\dots (3.3)$$

$$\text{Where} \left\{ \begin{array}{l} MMcost = \text{memeory cost of VM} \\ STcost = \text{Storage cost of VM} \\ CPU cost = \text{processing cost of VM} \end{array} \right.$$

Based on the above equation 3.2 & 3.2, the total placement costs used as an objective function. This objective function including both transfer cost and virtual machine cost to optimize the placement cost in a cloud environment. Thus, the problem can be mathematically formulated as placement optimization problem as follows:

$$\text{MIN(PC)} = \sum_{i=1}^D Tc + \sum_{j=1}^S Vc \dots\dots\dots (3.4)$$

$$\text{Where} \left\{ \begin{array}{l} PC = \text{Placement cost} \\ S = \text{Total no of virtual machine} \\ D = \text{Total no of dataceneter} \\ Tci = \text{transfer cost of ST } i=1, 2, \dots\dots\dots D \\ Vcj = \text{virtual machine cost } j=1, 2, 3, \dots\dots\dots S \end{array} \right.$$

The initial placement of ST in a virtual machine is fulfil the virtual machine resources. In order to place the ST components, the ST components requirement must always be less than or equal to the virtual machine capacity.

$$Tl \leq Vi \dots\dots\dots (3.5)$$

$$\text{Where} \left\{ \begin{array}{ll} Tl = \text{Size of ST} & l = 0 \dots\dots\dots N \\ Vi = \text{virtual machine capacity} & i = 0 \dots\dots\dots N \end{array} \right.$$

3.4 Proposed Solution

In cloud computing environment cloud infrastructure has contained a collection of the different datacenter that consists of multiple resources. This datacenter contains virtual machine resources to deployed the ST component a cloud environment. The system architecture of the proposed solution (as shown in figure 3.1) has four components. The first component is data center component; each datacenter has its own attribute and distributed across the different region. The second component is DC information, this component has contained each datacenter physical properties information like storage, memory, processing element. And those properties are accessed by the third component (Hybrid algorithm) of the proposed solution architecture. This algorithm has contained a hybrid of the two main optimization algorithms such as BA and GA. In this thesis, we focused on both located and distributed datacenter for cost effective ST placement by introducing an efficient hybrid approach. Identification of local VM resources and reconfiguration process can be performed efficiently with the help of high-performance local convergence capability of GA and identification of optimum placement is performed using the global convergence capability of BA. Finally, the placement approaches are designed in order to place the ST components. In this approach, two main strategies were used to identify an optimum placement location based on the proposed algorithms searching capability.

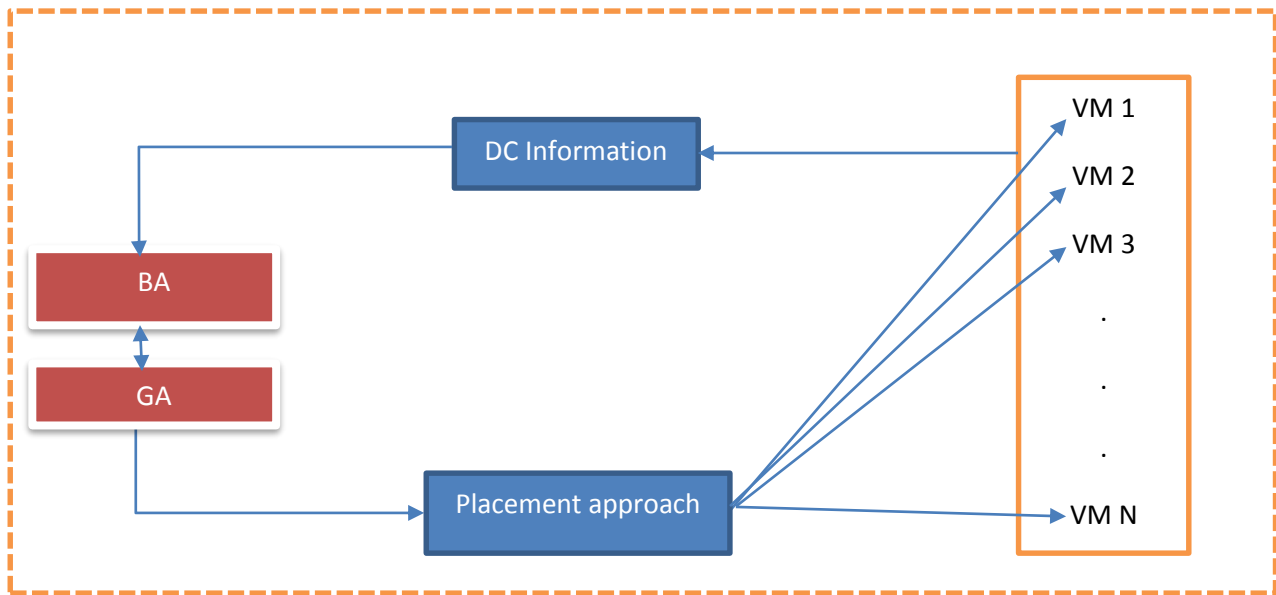


Figure 3.1: Proposed Solution

The proposed algorithm provides the two main searching strategy such as global search and local searching for identifying the suitable placement area across datacenter in cloud computing environment. In the first searching strategy (global searching) identify the best location based on placement objective. After finding the best location area, the next procedure of the placement strategy has to find the best virtual machine resource to deploy the ST on selected virtual machine using searching and repairing capability of the second hybrid algorithm(GA) in the proposed solution

3.4.1 Structure of Proposed Algorithm

The main aim of the proposed solution is to find an optimal placement area to place the ST components into appropriate VM resources, that meet the requirement of ST such as memory requirement, storage requirement, and processing requirement. And also handle resource constraint of a VM like memory capacity, storage capacity and processing power. Because the placement process must be handling requirement in order to deploy each ST component into different virtual machine resources.

Generally, the proposed solution has two main structure. The first structure of the hybrid approach is inspired by natural Bat [47]. This algorithm finds an optimal searching way in given searching space. The initial bat population is generated randomly because the location of food resources is not identified in the first optimal searching strategy. Velocity (V_i) , frequency(F_i) and position(X_i) are the main attributes of BA [35]. These Bat attributes are used to evaluate efficiency the ST component placement using the generated population in their placement area. The ST component move with velocity(V_i) and some predefined random frequency(F_i) to find an optimum placement location. Finally, each Bats update new location position of the ST component in a searching area using their previous position information.

Evolutionary algorithm convergence capability in the multi-dimensional problem is efficient, especially the GA is a powerful algorithm with their operators such as crossover and mutation. In the proposed solution, the second hybrid algorithm (GA) is used to improve searching capability of the first hybrid algorithm (BA) by handling the placement constraint, that is an obstacle to achieving optimum placement. The GA algorithm has used to repair and handle the placement constraint using GA operators such as crossover and mutation [38]. The first genetic

operators (crossover) repair the ST placement in VM to meet the placement requirement by changing the placement location. In first configuration process, the crossover operator is not fulfilling the placement requirement. To handle this situation, the second genetic operator (mutation) has used to find placement area that meets the ST requirement.

Algorithm 1

Initial ST placement Using BA-GA

1. Initialize DC resources cost (per storage and per memory)
 2. Initialize VM resources
 3. Initialize ST components and Bat attribute
 4. Change Placement iteration
 5. Updating location and velocity using equation (3.6 & 3.7)
 6. if (Random selected placement TC < Current placement TC)
 7. Select and use datacenter
 8. for (each ST component)
 9. for (each created datacenter VM)
 10. if (ST requirement < VM resources capacity)
 11. Place ST component in a given VM
 12. Calculate VM cost using *equation 3.3*
 13. Calculate TC cost using *equation 3.2*
 14. else
 15. **Perform cross over operation**
 16. **Perform mutation operation**
 17. end if
 18. end if
 19. if (Random selected loudness $A_i < \text{Current } A_i$ && f (Randomly select PC cost < The Best PC))
 20. Select the placement location
 20. Increase r_i and reduced A_i
 21. end if
-

3.5 Placement Approach

In this thesis, two main placement approach were used to identify the cost-effective placement area. Because efficient placement depends on the strategy that used to place the ST component of on cloud environment. The efficient hybrid approach has proposed to find out the best ST placement location in order to manage the cloud resource effectively and efficiently to give cost effective service to the user. In this section, the algorithms were described in detail.

In the proposed solution, the first hybrid algorithm (BA) has to find an optimal placement solution across datacenter using the global searching capability. In the proposed solution, the second algorithm also finds the best local placement area using GA. Figure 3.2 shows a flow chart of the first placement approach. In this strategy, first initialized datacenter resource properties including transfer cost, virtual machine cost (per storage, per memory and processing cost). Based on this infrastructure resources properties an optimal datacenter's locations are identified for cost-effective initial ST placement. This placement strategy is described in algorithm 1.

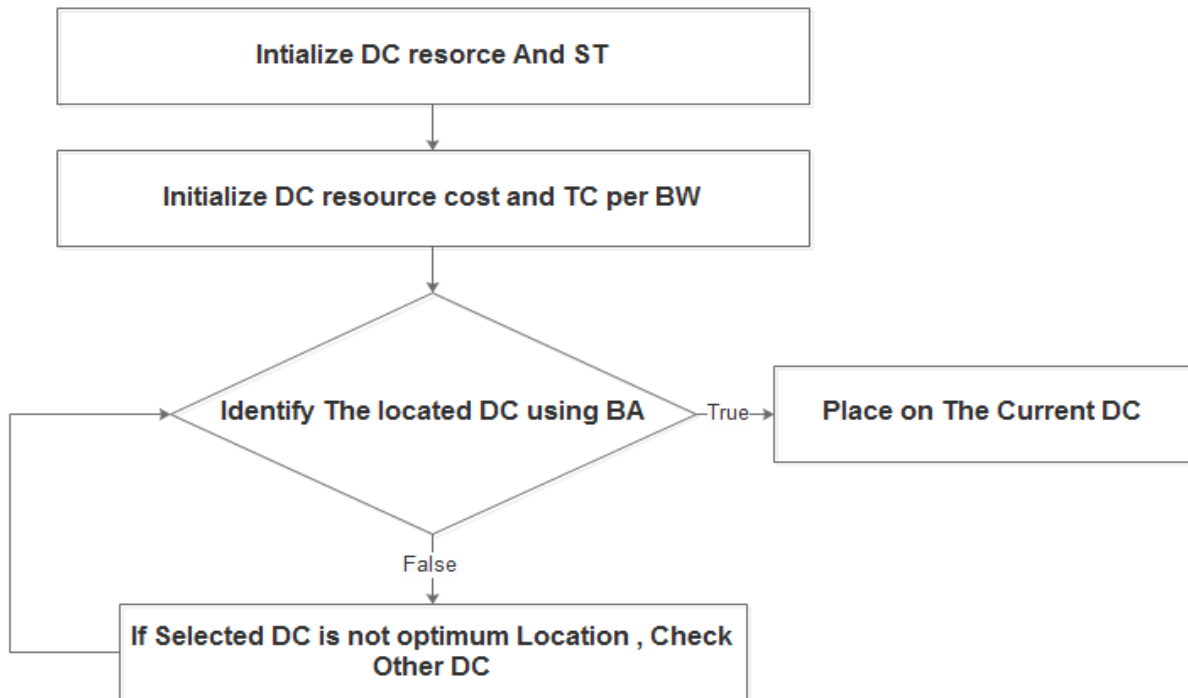


Figure 3.2: Placement Strategy Using BA

Steps for Selected Optimum VM

- Step 1: First initialized the DC physical resource such as virtual machine resource, virtual machine resource cost
- Step 2: Initialized the placement activity
- Step 3: Check the DC location based on TC
- Step 4: Generated the initialized ST components that are initialized in *step 1*
- Step 5: Generated the VM that are initialized in *step 1*
- Step 6: Placed the ST component on generated VM resources
- Step 7: If the placement is feasible, that means the ST component requirement size must less than the VM capacity
- Step 8: Place and calculate VM and TC using ***equation 3.4.***
- Step 9: If the placement is not feasible due to placement constraint, that means ST component requirement is not meet the VM cost
- Step 10: Repair the placement using genetic operators**
- Step 11: Repairing process is continuing (*Step 7*) until the placement criteria meet**
- Step 12: Check the final placement criteria including both TC and VC (Current selected PC cost must be less than the best PC cost)
- Step 13: If step 12 conditions are meet, selected the current placement and update the placement A_i and R_i .
- Step 14: But step 12 condition is not meet, change placement iteration by updating v_i and x_i using ***equation (3.7&3.8).***

In the ST component placement activity, in the proposed solution the first hybrid algorithm (BA algorithm) has used the attribute to control the searching capability and managed the placement operations. The BA algorithm has initialized the attribute of X_i, V_i, F_i is the frequency, velocity and position of BA algorithm attributes respectively. Thus, BA algorithm attributes has different role on the performance of the algorithm. Each BA attribute has randomly assigned the velocity, frequency and position at the first position identification using equation 3.7, 3.6 and 3.8 [35]. In the next round of target identification, the BA update the position and select the best current location using equation 3.8.

$$f_i = f_{\min} + (f_{\max} - f_{\min}) \dots\dots\dots(3.6)$$

$$V_i(t) = V_i^{(t-1)} + (X_i^{(t-1)} - X^*) * f_i \dots\dots\dots(3.7)$$

$$X_i(t) = X_i^{(t-1)} + V_i^{(t)} \dots\dots\dots(3.8)$$

where

- F_i = frequency of at position i
- $f_{\max}$ = maximum frequency
- $f_{\min}$ = minimum frequency
- $V_i(t)$ = velocity at t (current velocity)
- $V_i(t - 1)$ = pervious velocity at $t - 1$
- $X_i(t)$ = position at t (current velocity)
- X_i = previous position at $t - 1$

The next figure (figure 3.3) shows, how this BA control attributes work to identify the optimal placement location on ST placement. Thus, attributes such as V_i, f_i and X_i are generated at the initial stage of ST placement. After that, the next procedure is update the ST placement position for each placement iteration using the above equation.

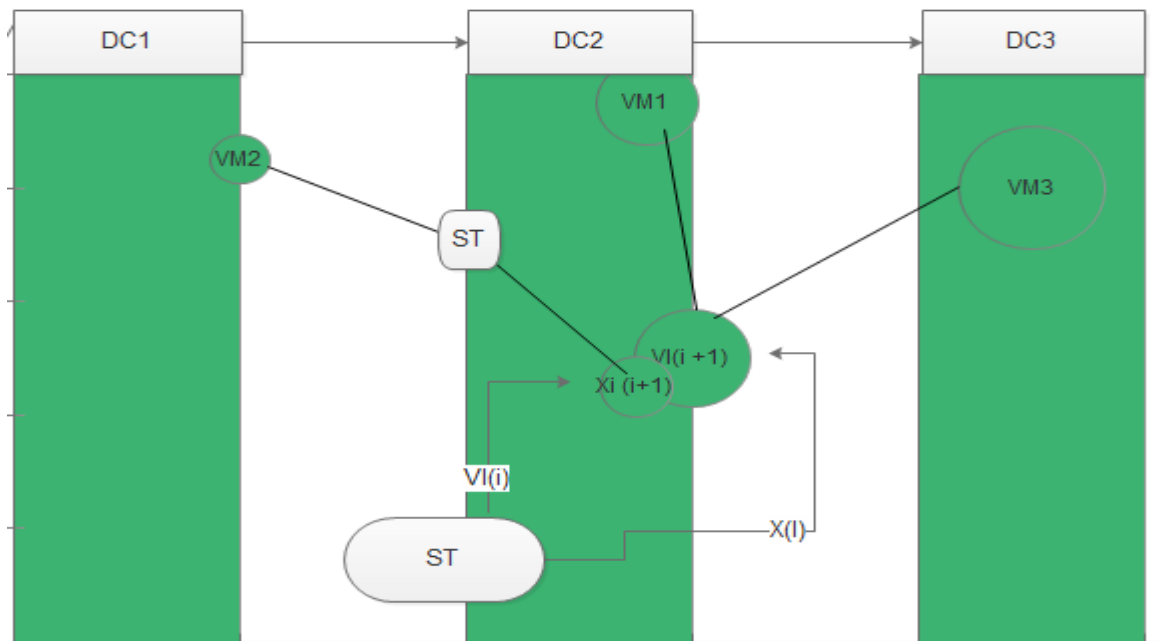


Figure 3.3 BA Control Attribute

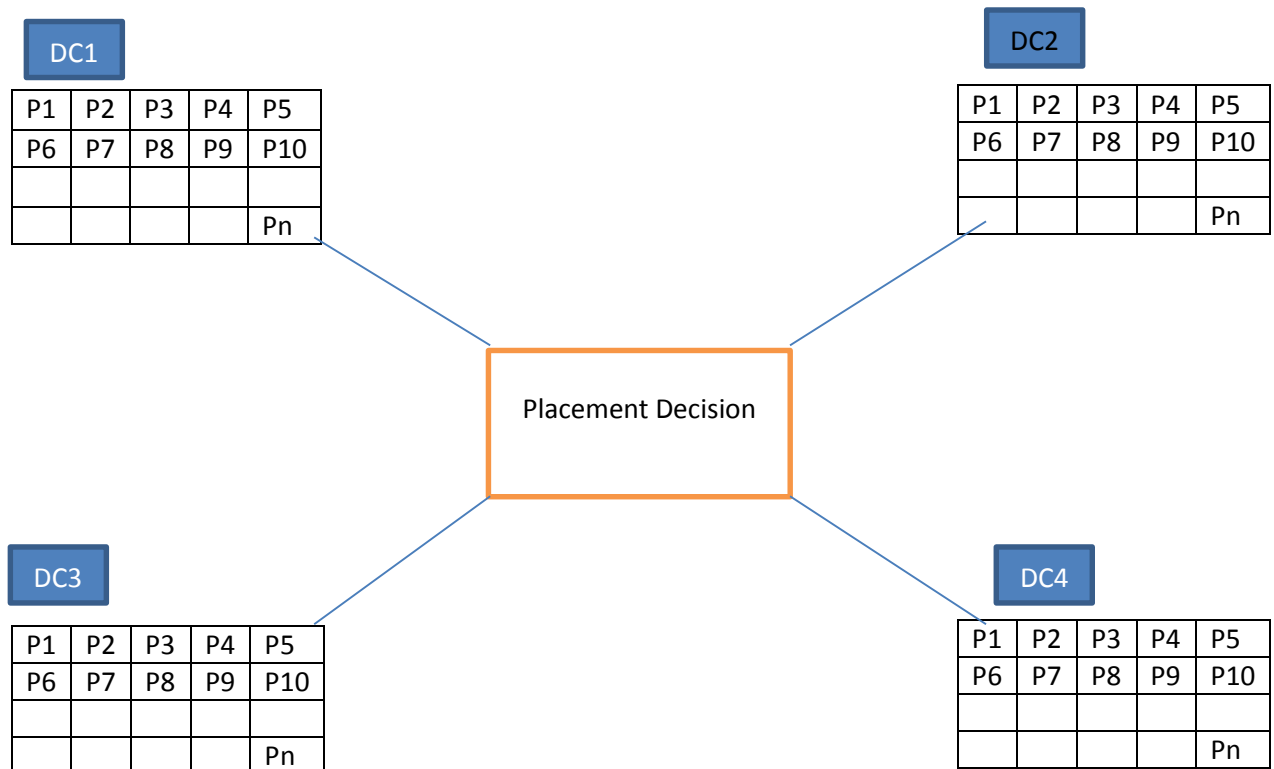


Figure 3.4: Initial Placement Representation Across DC

Graphical illustration of initial placement representation of ST component is given in figure 3.4. In this placement representation, the DC has located in the different region and each region contains multiple infrastructure resources. In the first placement strategy, the proposed algorithm identifies an optimal located DC using *Step 3*. If this condition is true, the algorithm has selected the located DC and generated the ST component into datacenter VM resources. The next figure 3.5 shows the placement of ST component on located datacenter.

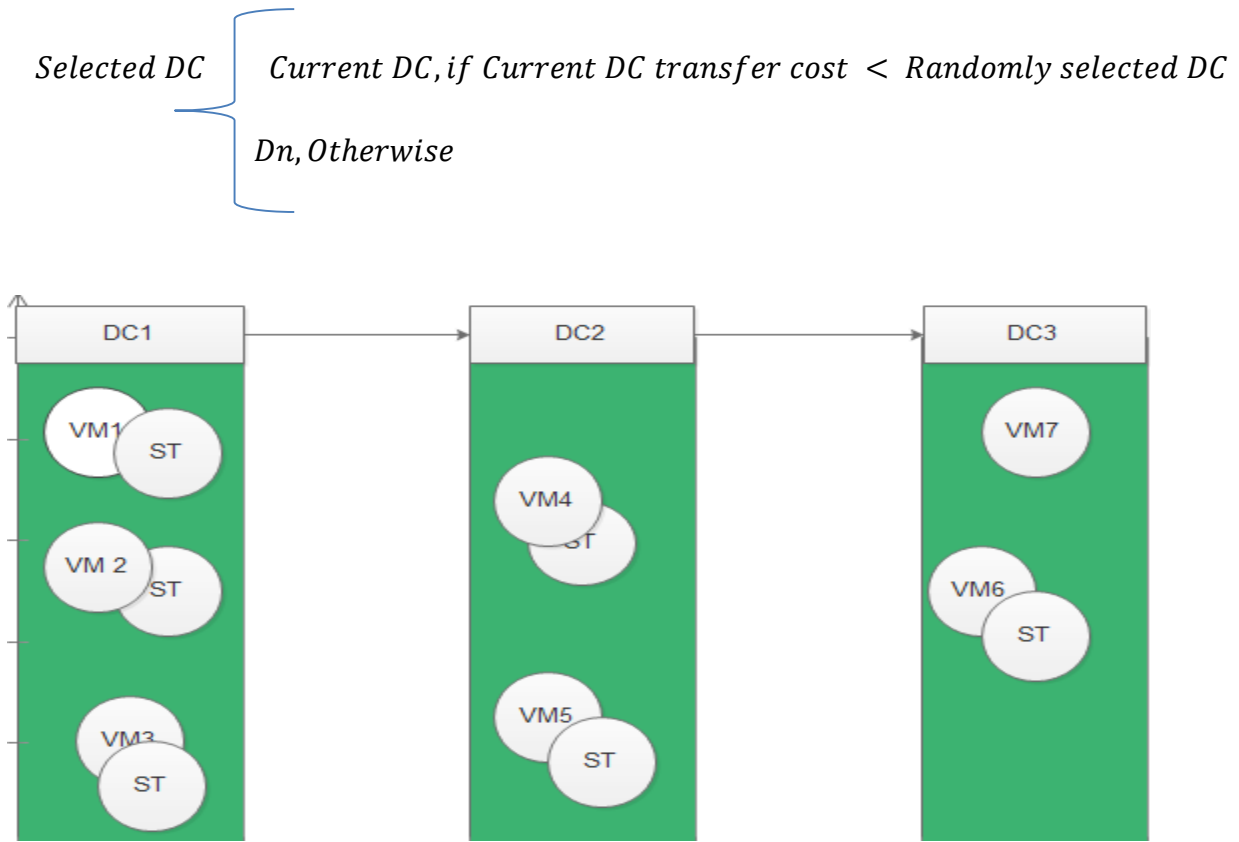


Figure 3.5 Generated ST into Virtual Machine Resources

After generating the ST component, the next procedure checks the placement constraint that shows in **step 7**. Sometimes the generated ST component requirement is not meet to place the random virtual machine resource. GA operator has used to manage the reconfiguration process of the placement by handling placement requirement constraint.

The next flowchart shows (figure 3.6) “The second placement approach “. The second strategy used to repair to meet the placement constraint problem and improve internal virtual machine searching strategy, to get an optimal virtual machine resource. The repairing strategy used to find an optimal virtual machine in located datacenter easily. In the first iteration of this strategy, the ST components are placed on generated VM resources. If the current placement activity is not fulfilling the requirement and constraint of ST and virtual machine resources respectively, the GA algorithm checks the placement constraint such as the current placement ST resource requirement. GA algorithm searches an appropriate local VM resource using mutation and crossover operators. The reconfiguration strategy and steps are described in the next figure 3.6.

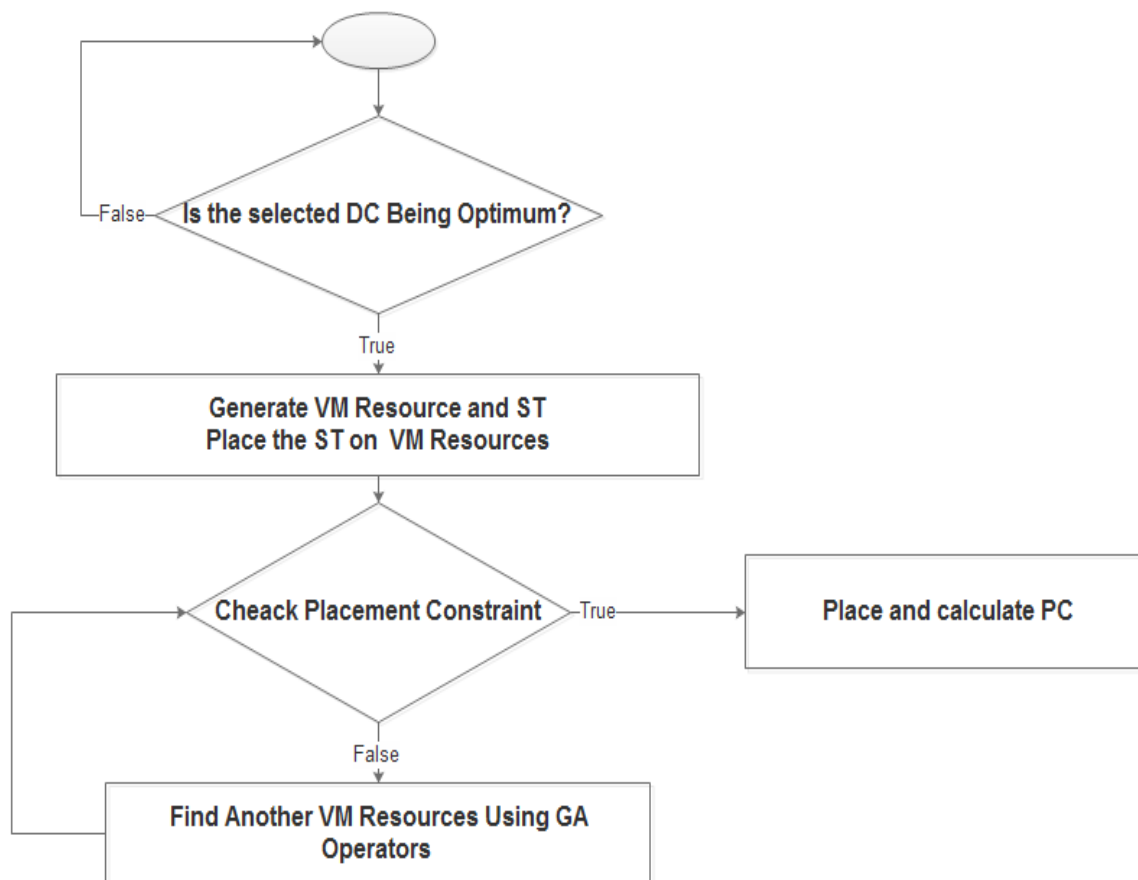


Figure 3.6: The Second Placement Strategy Using GA

Steps for Reconfiguring Placement Strategy On Selected DC

Step1: Select an optimal DC location using BA

Step2: Placed the generated ST component on selected DC

Step 3: If the ST component requirement (memory requirement, storage requirement, and processing element requirement) must be less than or equal to the VM capacity (memory capacity, storage capacity, processing capacity)

Step 4: If the above condition(Step3) is fulfilling

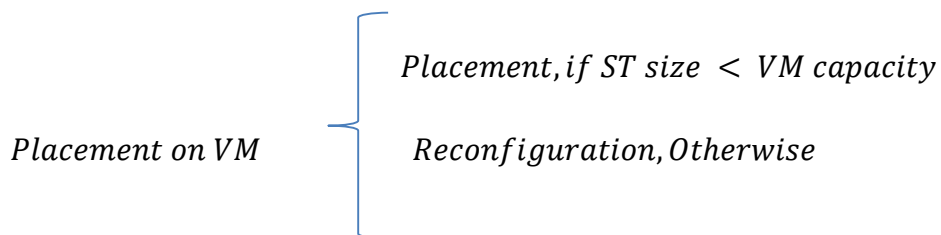
Step 5: Place the ST component on generated VM resources

Step 6: But, placement constraint is not meet in step 3 go to step 7.

Step 7: Perform crossover genetic operators (change by selecting a random crossover point)

Step 8: Sometimes crossover operation is not repaired, we have used the second genetic operator mutation (change placement configuration by changing a single ST placement location). Finally, the repairing process is continuing until the placement constraint meet that shows in step 3.

In the first round placement process, first, the proposed algorithm checks the placement constraints before placing the generated ST component into virtual machine resources. The next figure 3.7 shows the initial placement process by checking the constraint.



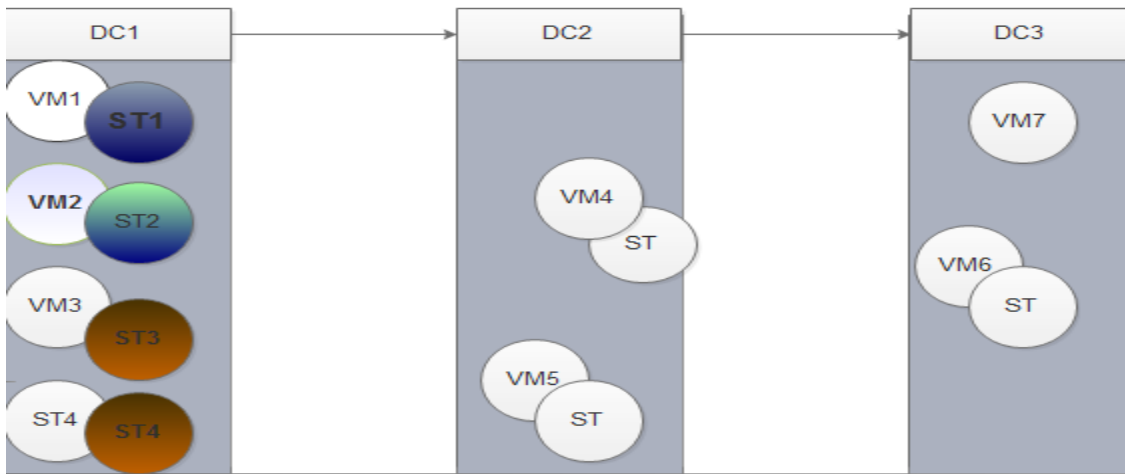


Figure 3.7: Placement Constraint Checking Procedure Using GA

The above figure 3.7 shows the representation of ST components placement in a given virtual machine with generated using BA. For example, in the above placement process, ST1 and ST2 do not meet the placement constraint conditions. The placement needs to reconfigure to find another virtual machine resources. The first genetic operators called crossover implemented to change placement location. The crossover operation is one of genetic control attribute, which gives the best fitness result by changing position. The crossover operation exchanges the ST placement from one VM location into another VM location to identify the optimal solution and handle the placement constraint.

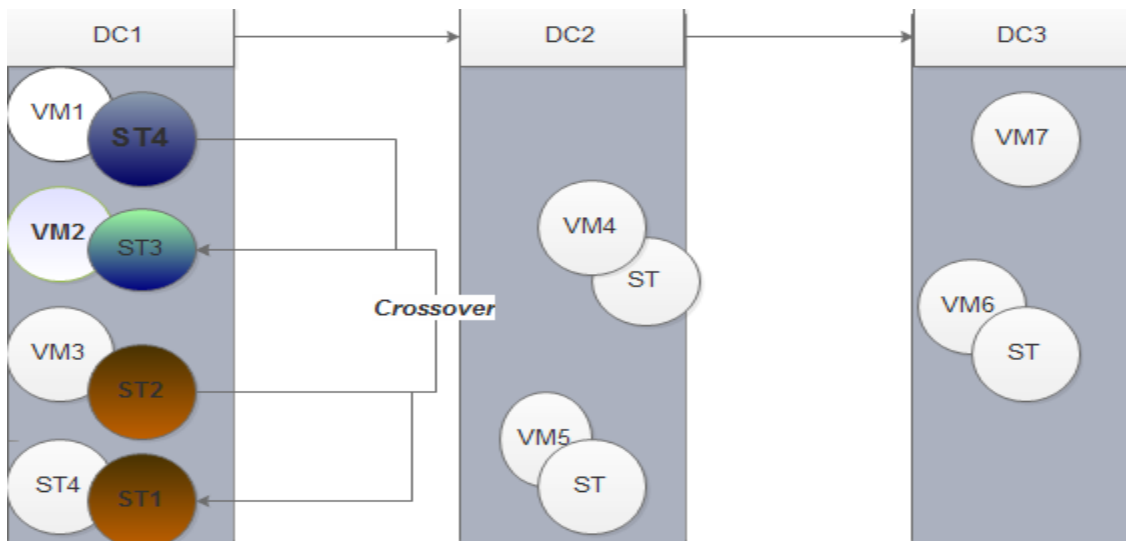


Figure 3.8: Placement Configuration Using Crossover

The above figure 3.8, shows the reconfiguration process to meet the requirement constraint and also find optimal virtual machine resources. The reconfiguration process must change the current configuration process using the next GA operator (mutation). The mutation operation is the second intelligent operator of the genetic algorithm. This operator is handling the configuration process. Figure 3.9 shows reconfiguration process using mutation operator. The mutation operation uses after crossover operation performed. This operator is handling the configuration process. Sometimes crossover operation dose not produced best placement solution. In this reconfiguration processes, the mutation operators are identifying the optimum VM location by changing the ST component position.

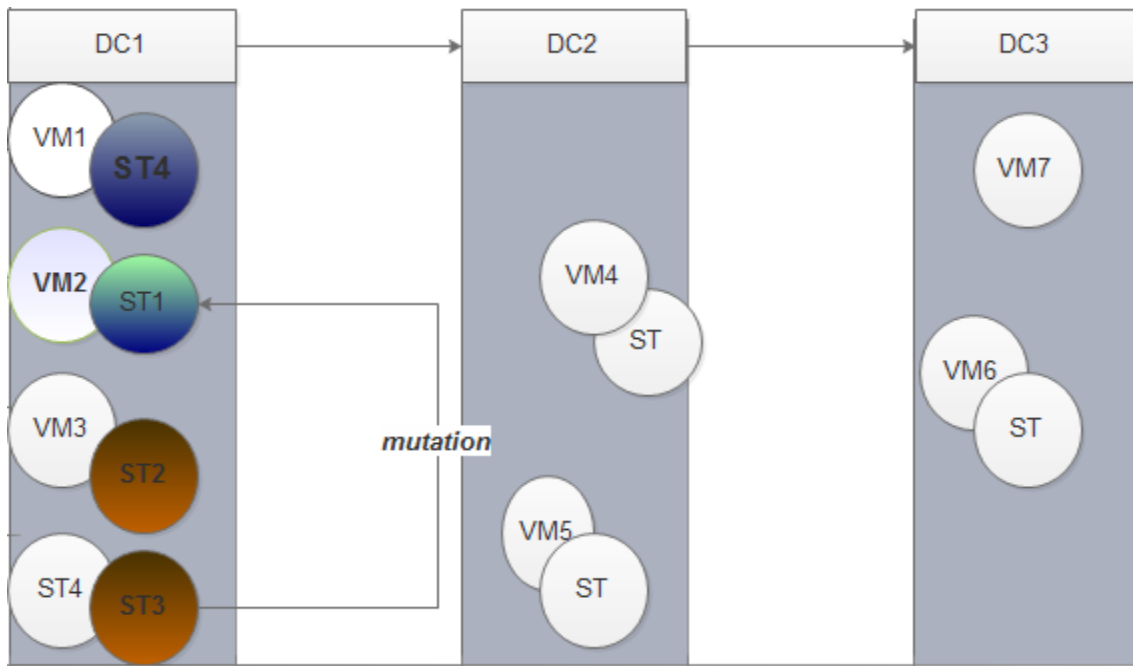


Figure 3.9: Placement Configuration Using Mutation

After completed the reconfigurations process using the GA operators (mutation and crossover), the next step is identifying the current best placement location using BA algorithm. In this step, the BA algorithm is select an optimum location by comparing TC and VC in each placement iteration. Thus, the BA attributes are updating the placement position using previous placement information. The next figure 3.10 shows, how the BA update the placement position.

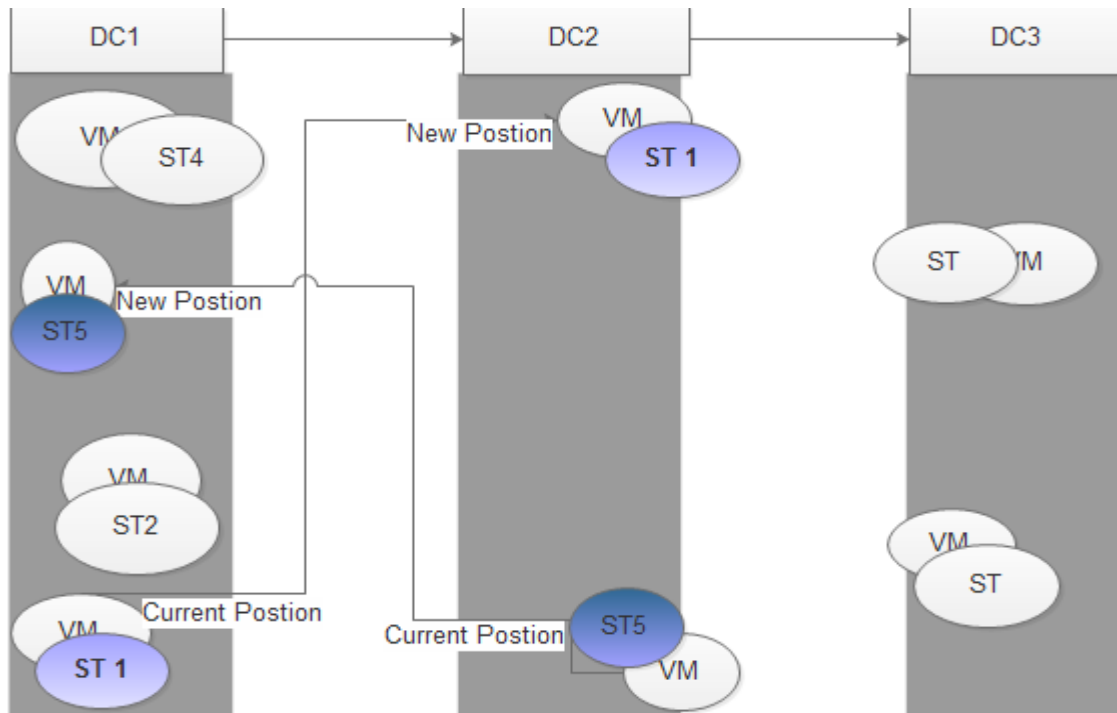


Figure 3.10: Identification of New Placement Position Using BA

As mentioned above, the BA control attribute manages the placement position in each iteration. for example, in the first placement iteration, the ST has its own placement position. This placement position is calculated using equation 3.8. In the above figure, ST1 and ST5 have changed the previous location to find another best location by updating its previous information including position and velocity.

Summary

This chapter presented, a cost-efficient initial ST component placement algorithm that is designed using hybrid approaches to optimize the resource usage. To address the placement problem, the placement strategy has been divided into two section. In the first strategy identifying the best placement environment across datacenter using the first proposed algorithm global searching capability (BA). And the second strategy (GA) was identifying optimal placement area across physical host on selected DC and handling the placement constraints using the local searching capability of the second hybrid algorithm(GA). The next chapter, the experiment simulation environment is designed using CloudSim cloud simulation tool to test the performance of proposed algorithm to achieve the objectives.

CHAPTER FOUR: EXPERIMENTAL SETUP AND IMPLEMENTATION

4.1 Introduction

In this experimentation section, the proposed solution is implemented. The cloud computing simulation tool called CloudSim simulation is used to implement the proposed algorithm. In the first section of this chapter, the detailed architecture of the tool is presented. In the second section, the simulation environment configuration is presented. In the third section, different simulation scenarios have been defined in order to implement the proposed solution.

4.2 CloudSim

Simulation tools are an important tool in order to investigate various cloud activities that are difficult to test in the real environment due to lack of cost and time, especially in cloud computing environment. Today a lot of cloud computing simulation tools are available in the market, most of those simulation tools are open source. CloudSim is one of cloud a simulation tool that performs cloud-based simulation activities. This simulation tool is a key tool to test and control the performance of simulation activity before implementing on to the actual cloud computing environment.

CloudSim is an open source cloud simulation tool and used simJava toolkit to model and simulate different cloud computing scenario. Using this simulation tool, it can simulate the cloud scenario by creating and managing different cloud computing resources. CloudSim support simulating different cloud scenarios, from simple up to complex cloud environments implementations.

CloudSim is a framework that creates virtual cloud infrastructure environment using different policy to manage the cloud resources including network topology and message communication that provide a flexible simulation environment to the user [42].

CloudSim provides different important class such as datacenter, virtual machine, application, users and scheduling and provision policy for managing different parts of the simulation tool, it is also important to assess various activity like measure performance and cost the resources [48].

4.2.1 Classes in CloudSim

CloudSim is an open source cloud simulation software. Using a library of this simulation tool, the CloudSim integrating the java JDK to compile the simulation code. CloudSim is a self-contained platform which can be used to model different data centers, service brokers, scheduling and allocation policies of a large scaled Cloud platform. Different IDEs such as Eclipse and NetBeans can easily access the CloudSim library class to use and implemented the algorithm. To perform simulation activity, CloudSim has a set of various class. This class is important for modeling the simulation environment. To model the simulation environment, the CloudSim has used a class, abstract class, and interfaces. In this section, the main class of the CloudSim simulator tools is identified that are important to perform the simulation scenario.

The basic class and the function of CloudSim modules are described below [42, 49]: -

Datacenter:

In CloudSim datacenter, datacenter resource contains different resources to the user such as hardware and software. This resource has contained compute and storage servers that are important to provide efficient simulation services.

DatacenterCharacteristics:

This class module contains detail information of data center resources characteristics.

Host:

The datacenter is a collection of the physical resource including a compute or storage server. This class module contains information about memory, storage, processing elements, allocation and provision policy of cloud physical resources.

Datacenterbroker:

This class act as a bridge between cloud user and cloud provider to communicate with each other. Using this broker, the user can get the whole datacenter information from cloud information registry center(CIS) and also extended broker class module to create our own customs broker policy.

Virtual Machine:

This class module is responsible for managing the creation of virtual machine resources that are created by cloud host.

Cloudlet:

CloudSim class module contains an application or tasks placed different cloud hosts in datacenter. Each task has its own property such as, task length, task size and input/output requirement.

BWprovisioner:

This provisionary class is an abstract class that used to provide the bandwidth based on the provisioning policy.

Memory Provisioner:

It is an abstract class that controls and identifies the cloud hosts, it is suitable for creating virtual machine resources satisfy memory, storage, and other requirement.

VM Allocation Policy:

VM Allocation Policy is an abstract class that implemented by cloud hosts and control the provisioning allocation of the virtual machine to cloud hosts. The responsibility of this class module is to search a suitable host for virtual machine placement based on the virtual machine requirement.

VmScheduler:

This virtual machine scheduler class is an abstract class that used to implemented space shared and time-share scheduling policy for allocating the processing power element to a virtual machine.

SANStorage:

This CloudSim simulation class contains the local storage area network information of the simulation tool. SANStorage class implements an interface class to simulate and assess the delay and latencies of data transmission.

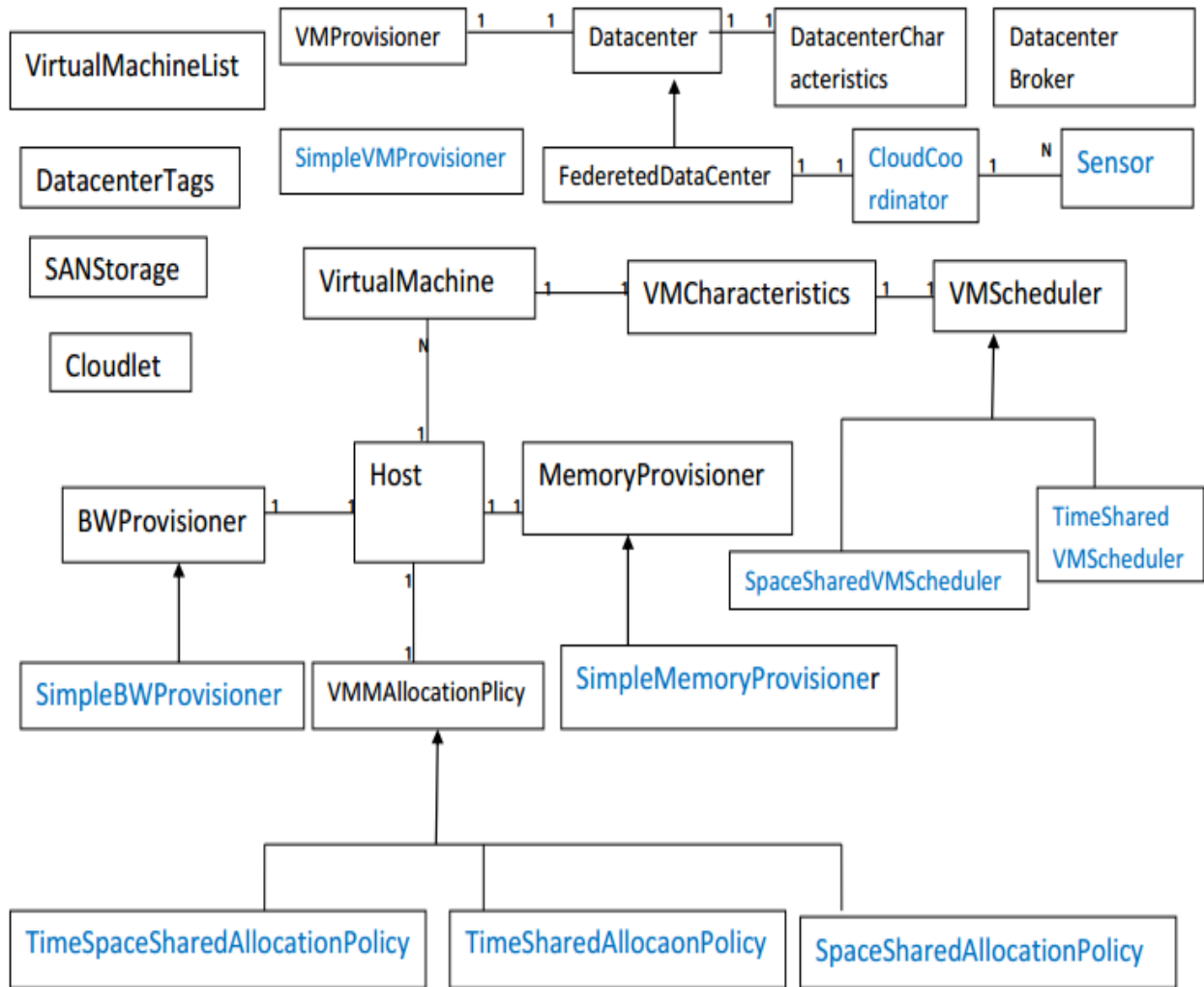


Figure 4.1: CloudSim Class Design Diagram [49].

4.2 Simulation Environment

To perform simulation activity, a simulation environment was created using CloudSim simulation tool. In this simulation activity, different data centers are created with various physical hosts. Each physical hosts are its own properties and resources. After creating a data center, virtual machine resources are created based on the physical host resources. Finally, generate ST components by assigning its attribute and properties element. Figure 4.2 shows the experiment simulation environment component using CloudSim simulation tool.

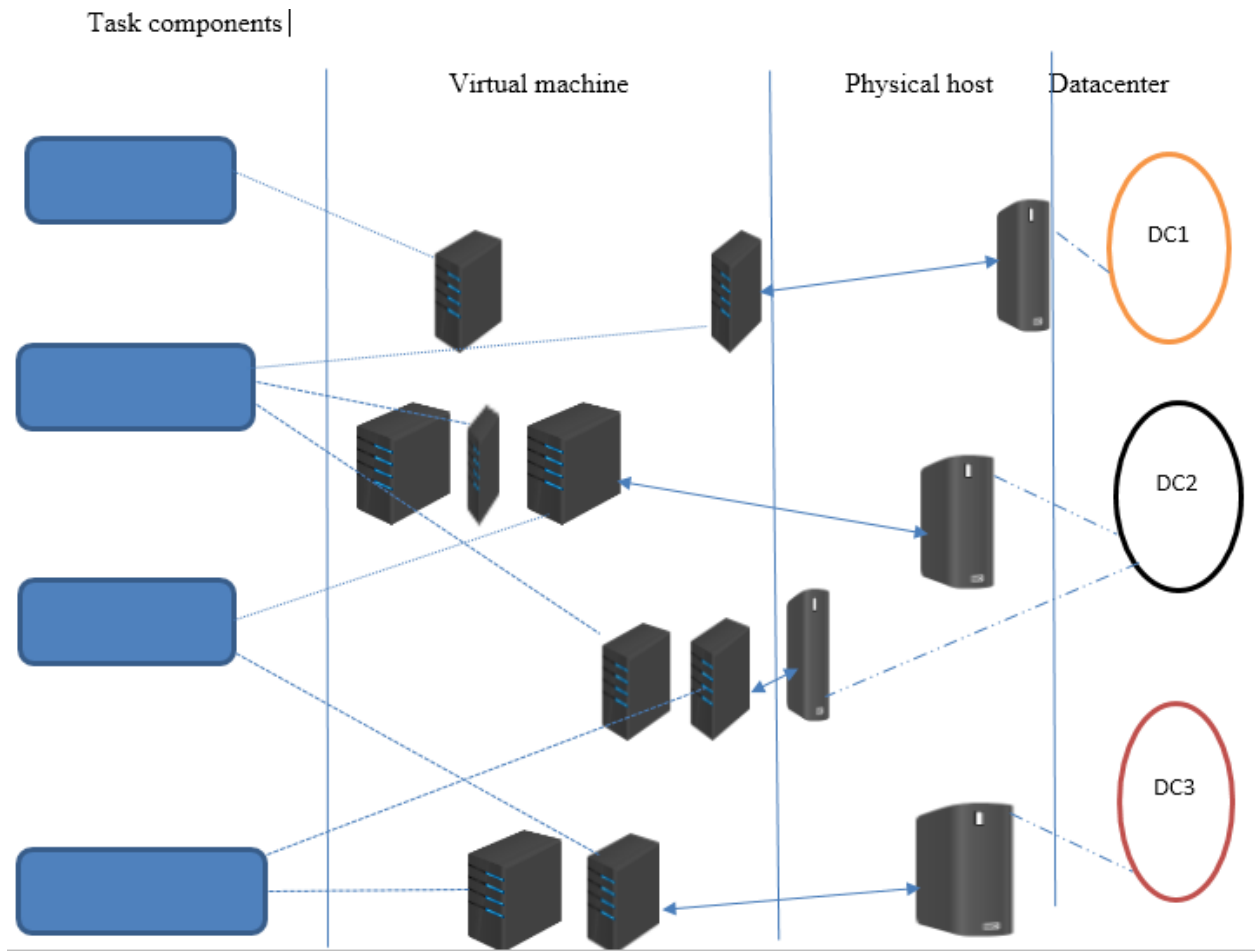


Figure 4.2 Simulation Environment of CloudSim

Physical host: Datacenter is the first component of the simulation environment. This component contains different configuration elements including processing element, storage element, and memory.

Virtual machine: After creating the datacenter infrastructure, the second main component of simulation environment is a virtual machine. In this simulation environment, virtual machine resources are created from physical host resources.

ST components: ST contain the software task applications. In these simulation components, ST applications are created including their own properties such as a size of ST, memory requirement and output requirement.

Table 4.1: Simulation Component and Configuration of the Experiment

Simulation Environment Components	Simulation Component Attribute
Physical server	Each datacenter contains different physical hosts with its properties (Host ID) ----> Id of the server (Host Name) ----->Name of server (Host storage) ----> Storage size (Host RAM) ----->Memory size
Datacenter characteristics	(Arch) -----> Physical architecture of the server (OS) -----> Operating system used by the server (VMM) -----> Hypervisor used by the server (costPerMem)-----> Cost of memory to use the resources (costPerStorage)-----> Cost of storage to use the resources (costPerBw)-----> Cost of bandwidth to use the resources
Virtual machine	(VM size) -----> Image size of VM (VM Ram) ----->VM memory (VM PNo) -----> Number of CPU
ST components	(Fsize) ----->Total file size of ST (Output size) ---->Output size of the ST

4.2.1 CloudSim Configuration

To implement the proposed algorithm, first configure datacenter resource element based on the simulation scenario. Before creating datacenter component, the first step is instantiating CloudSim library and creation of CloudSim datacenter object.

```
CloudSim.init(num_user, calendar, trace flag);
```

```
Datacenter datacenter0 = createDatacenter("Datacenter_0");
```

```
Datacenter datacenter1 = createDatacenter("Datacenter_1");
```

```
Datacenter datacenter2= createDatacenter("Datacenter_2");
```

```
.  
.
```

```
Datacenter datacenter2= createDatacenter("Datacenter_N");
```

After initialization the CloudSim library, the next step is creating a datacenter and their resources such as virtual machine and physical Host (Host). Each host and virtual machine contain resource property and its cost. Host Id and Host Name is the first property that used to identify the datacenter host resources uniquely from other datacenter resources. The second component property is the storage, RAM size, and processing element of datacenter physical host and virtual machine. Finally, the characteristics of datacenter have its own storage and RAM cost to place ST component on physical Host. After datacenters configuration, generating of ST components by assigning their properties for each component. The detail configuration of cloud datacenter elements is described below.

4.2.2 Datacenter Physical Hosts Configuration

Host configuration is the first data center configuration that after instantiating of CloudSim simulation library. In datacenter configuration, physical Host and its processing element are configured. In this experimentation, various host elements are created with RAM size, storage size, and bandwidth. After creating a host, the next step is adding processing element for each physical host. CloudSim Pe (Processing Element) class represents CPU unit, defined in terms of Millions of Instructions Per Second (MIPS) rating. This processing element is measured by million instructions per second of the CPU time and it is important to give the processing power for each datacenters Host element.

4.2.3 Datacenter Virtual Machines Configuration

To configure the VM's resources, the requirements must be known for placing the VM on the physical Host in a correct manner. The VM placement required some resources in the physical host and need a different resource that VM runs inside a Host. After configuration of physical host using the above physical Host configuration, the next step is configuring virtual machine in cloud datacenter simulation environment starting by assigning of virtual machine element such as Image size, memory size(RAM), million instructions per second(MIPS), processing element(Pe) and bandwidth(BW). These attributes are the main element of virtual machine configuration assigned to for each virtual machine configuration element and pass the value to virtual machine simulation class.

4.2.4 Datacenter Broker Configuration

In CloudSim simulation environment broker is the main simulation element. Broker element act as a bridge between user and datacenter. This broker class model, which is responsible for manage communication between users and service providers depending on users need requirements and deploys ST across DC. DC broker is used for manage which data center resources should be selected for ST placement that satisfies the requirement. Datacenterbroker represents a broker acting on behalf of a user. It manages VM resources, VM creation, submission of STs and destruction of VMs

4.2.5 Datacenter Characteristics Configuration

Datacenter characteristics contain datacenter information such as the architecture of physical server, Host operating system, hypervisor, time zone, costPerMem, costPerStorage and CostPerbandwidth. After initializing the value of each datacenter characteristics component, the datacenter characteristics CloudSim simulation class have been passed.

After configuration cloud environment, the next activity is implementing the proposed algorithm in configured CloudSim configuration environment. In order to implement the proposed algorithm, the ST components attribute are initialized such as length (length or size (in MI) of the ST to be executed in a Power Datacenter (assuming MI stands for Million Instructions), file size (size of ST before submitting), output size (size of the ST after

submitting) and `getActualCpuTime` gets the total execution time of ST in a given Cloud Resource ID. Each attribute of ST has contained its own attribute value.

After configuration of datacenter resources of CloudSim, the next procedure is implemented the proposed algorithm by creating different simulation scenario in configured CloudSim environment. Figure 4.3 shows the importing class package and CloudSim simulation environment using NetBeans IDE.

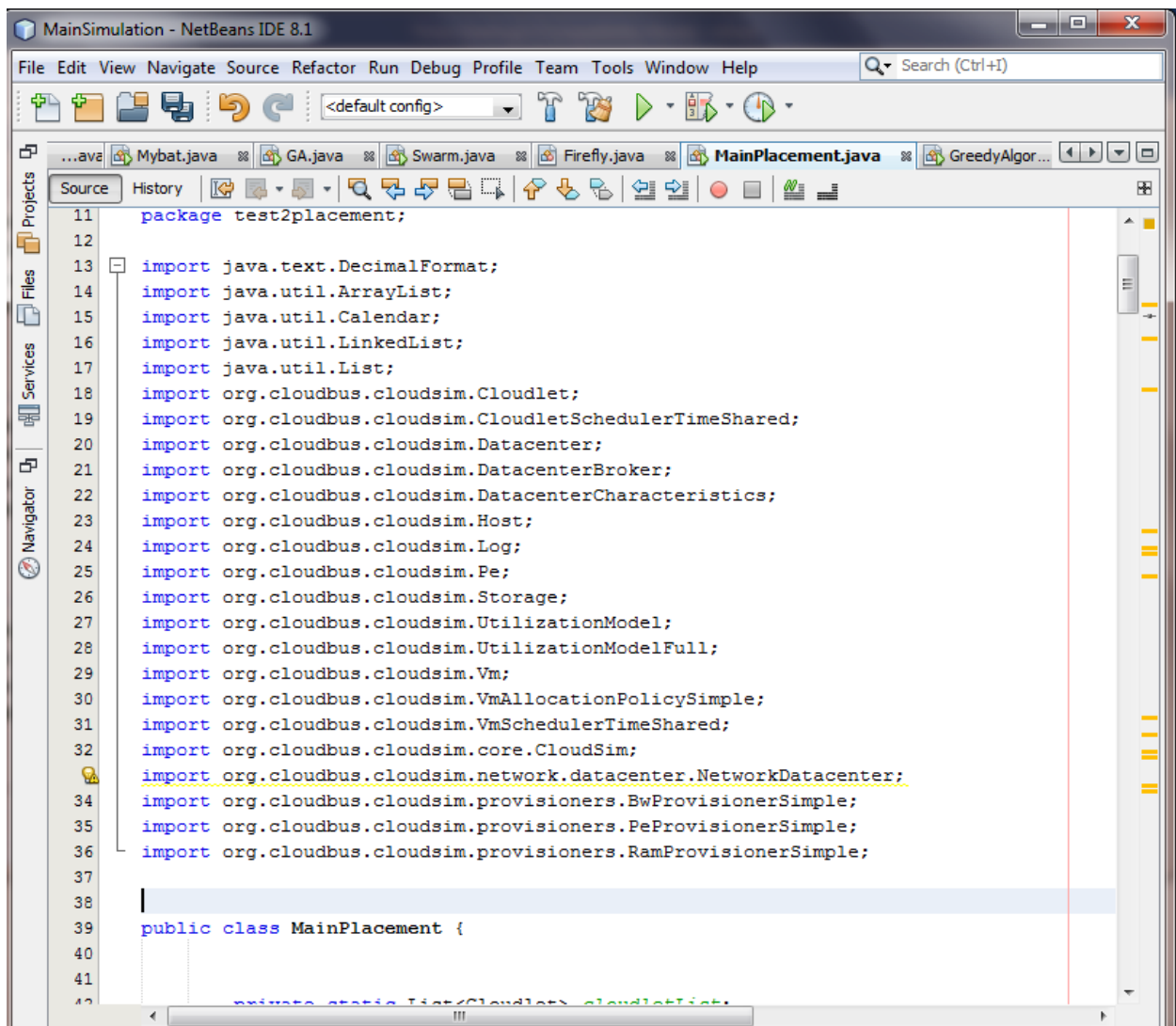


Figure 4.3: Simulation Screen

4.3 Implementation

In this section, experimental implementation has been done in the configured CloudSim environments. The proposed algorithm has been implemented using java programming language and evaluates the performance of the proposed hybrid approach. For all experimental scenarios, the proposed solutions were implemented on a large cloud environment and the attributes of datacenter information like size of the server , virtual machines, and its cost were generated from AWS [6]. All experiments were performed in Toshiba laptop computers with three GHz Intel Core 2 Duo CPU and 4GB RAM.

4.3.1 Simulation Scenario

This section, the simulation scenarios of ST placement a Cloud data center using CloudSim are carried out to meet the objective. To evaluating the placement technique five simulation activities were performed by changing a cloud datacenter resource metric to observe the impact of proposed approach in the simulation environment. In the first two experiment (experiment 1&2), the proposed algorithm is implemented on located DC by changing the number of ST and VM to evaluate the performance of hybrid approach to place the ST. The main aim of the above three experiments is to evaluate the local convergence capability of the proposed algorithm to find an optimum located VM resource. In the next three experiments (experiment 2,3 & 4) the proposed solution is implemented and the main aim of those experiments are to investigate the convergence capability those hybrid algorithms capability across DC by changing the VM and ST resources.

4.3.2 Terminology and Resource Attribute Used in the Experiment

To implement the proposed algorithm in this experiment, the following simulation terminology and resource attributes [6] were used throughout the experiment.

Table 4.2: Terminology Used in The Experiment

Placement cost	Placement cost is the total cost of virtual machine cost and transfer cost.
VM cost	Virtual machine cost is the amount cost to use this virtual machine resource .
Transfer cost	Transfer cost is the amount of cost need to transfer from one point to another.

Table 4.3: Resource Attribute Used in The Experiment

VM RamSize	600 MB- 8GB
VMSTsize	100 MB- 10GB
VMcostPerMem	0.09\$
STSize	100MB – 2GB
VMcostPerStorage	0.07\$
VMcostPerBw	0.01\$
VMcostPerPe	4.0\$

4.3.3 Experiment Scenario 1

This demonstration is evaluating the impact of changing ST on the initial placement on located DC and how the placement scenario using a hybrid approach to minimize placement cost and service efficiency by comparing the proposed algorithm with other existing placement algorithms in a cloud environment. During this experiment, the attribute and the value of DC resource component have been set and change the parameter each time to investigate the impact of the proposed solution.

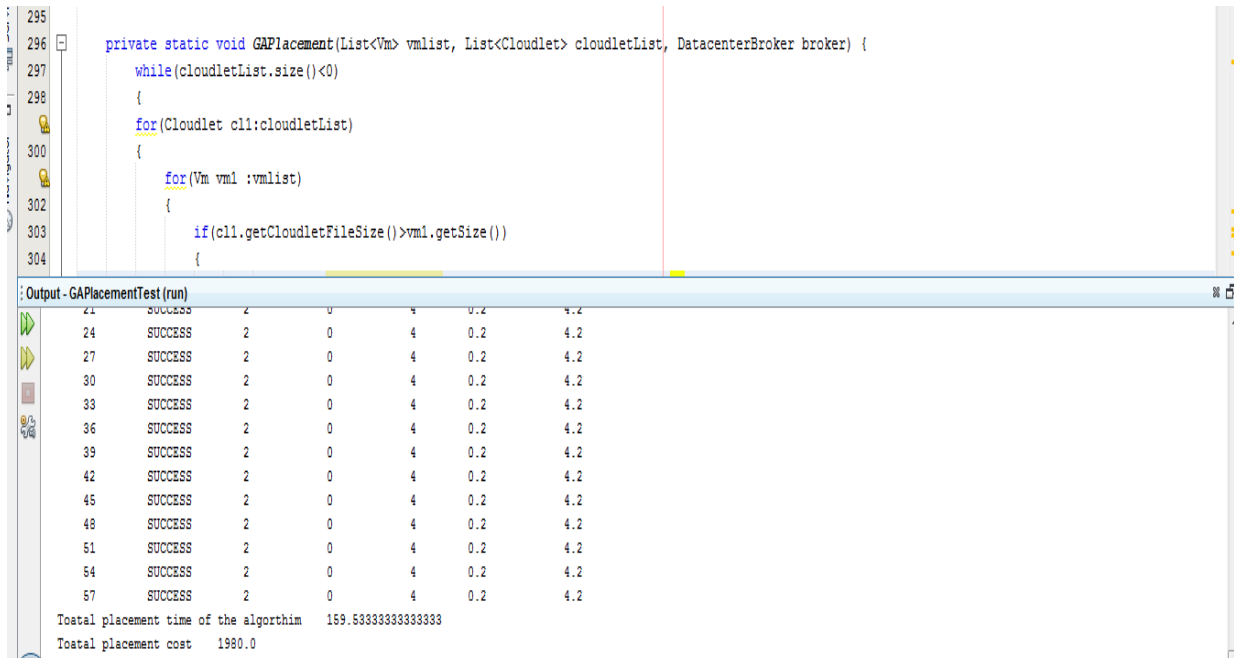


Figure 4.4: Sample Experiment Scenario 1 Simulation Results

4.3.4 Experiment Scenario 2

The second demonstration is more similar to the previous demonstration (experiment scenario 1). In the first experiment scenario, the local convergence capability of proposed hybrid approach has investigated by constant the VM and increase the Number of ST. But in this experiment, increase the number of ST while the VMs remain constant. Thus, only the value of the ST parameters is changed the attribute value in each time to observe the impact of ST in initial placement to reduce the initial placement cost. For the second experimental demonstration, the attribute and the value of DC resource component were also generated to implement the and evaluate the proposed solution.

```

183 private static void psoAlgo(List<Vm> vmList, List<Cloudlet> cloudletList, DatacenterBroker broker) {
184     while(cloudletList.size()<0)
185     {
186         for(Cloudlet c11:cloudletList)
187         {
188             for(Vm vm1 :vmList)
189             {
190                 if(c11.getCloudletFileSize()>vm1.getSize())
191                 {
192                     broker.bindCloudletToVm(c11.getCloudletId(), vm1.getId());
193                     double cbest=c11.getProcessingCost();
194                 }
195                 if(c11.getCostPerSec(vm1.getId())>c11.getProcessingCost())

```

Output - MainSimulation (run)

```

> ===== OUTPUT =====
> Cloudlet ID   STATUS   Data center ID  VM ID   Time   Start Time   Finish Time
> 2            SUCCESS   3              2       0.33   0.3          0.63
> 5            SUCCESS   3              2       0.33   0.3          0.63
> 8            SUCCESS   3              2       0.33   0.3          0.63
> 1            SUCCESS   2              1       0.5    0.3          0.8
> 4            SUCCESS   2              1       0.5    0.3          0.8
> 7            SUCCESS   2              1       0.5    0.3          0.8
> 0            SUCCESS   2              0       1.33   0.3          1.63
> 3            SUCCESS   2              0       1.33   0.3          1.63
> 6            SUCCESS   2              0       1.33   0.3          1.63
> 9            SUCCESS   2              0       1.33   0.3          1.63
> Total placement time of the algorithm  10.8175
> Total placement cost  450.0

```

Figure 4.5: Sample Experiment Scenario 2 Simulation Results

4.3.5 Experiment Scenario 3

In the previous two experiments (experiment 1 and 2), the proposed solution has implemented to evaluate the performance of hybrid approach by varying the size of VM and ST only on located datacenter. But, in this demonstration, the proposed algorithm is implemented to evaluate the impact of proposed solution by increasing DC on distributed DC and how the placement scenario using a hybrid approach to minimize placement cost by comparing the proposed algorithm with other existing placement algorithms in a cloud environment.

During this experiment, the infrastructure attribute value the attribute and the value of DC resource component has been set and change the parameter each time to investigate the impact of the algorithm in placement scenario across the DC.

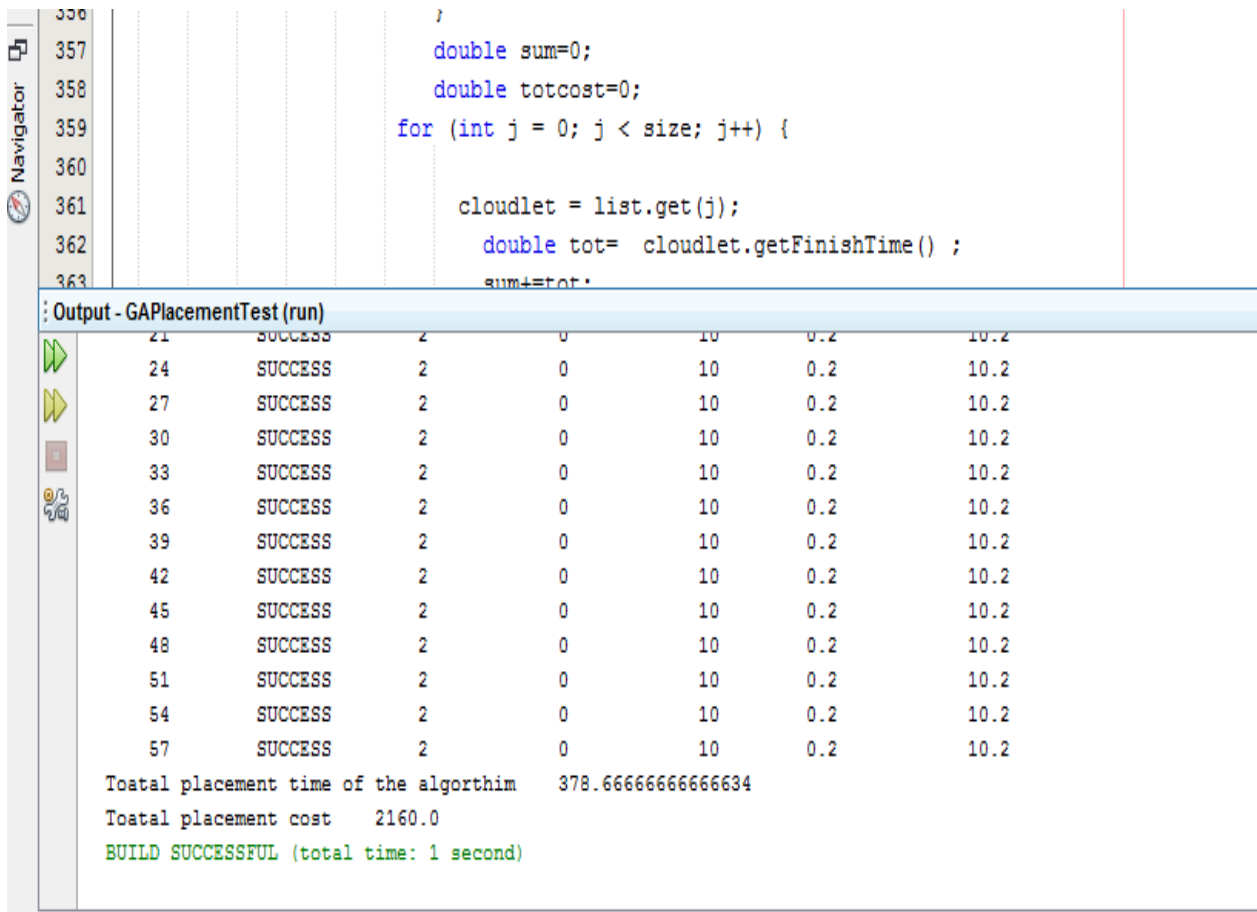


Figure 4.6: Sample Experiment Scenario 3 Simulation Results

4.3.6 Experiment Scenario 4

This demonstration is evaluating the impact of changing ST on the initial placement in different DC and how the placement scenario using a hybrid approach to minimize placement cost and service efficiency by comparing the proposed algorithm with other existing placement algorithms in a cloud environment. During this experiment, the attribute and the value of DC resource component have been set and change the parameter each time to investigate the impact of the algorithm in placement scenario across the DC.

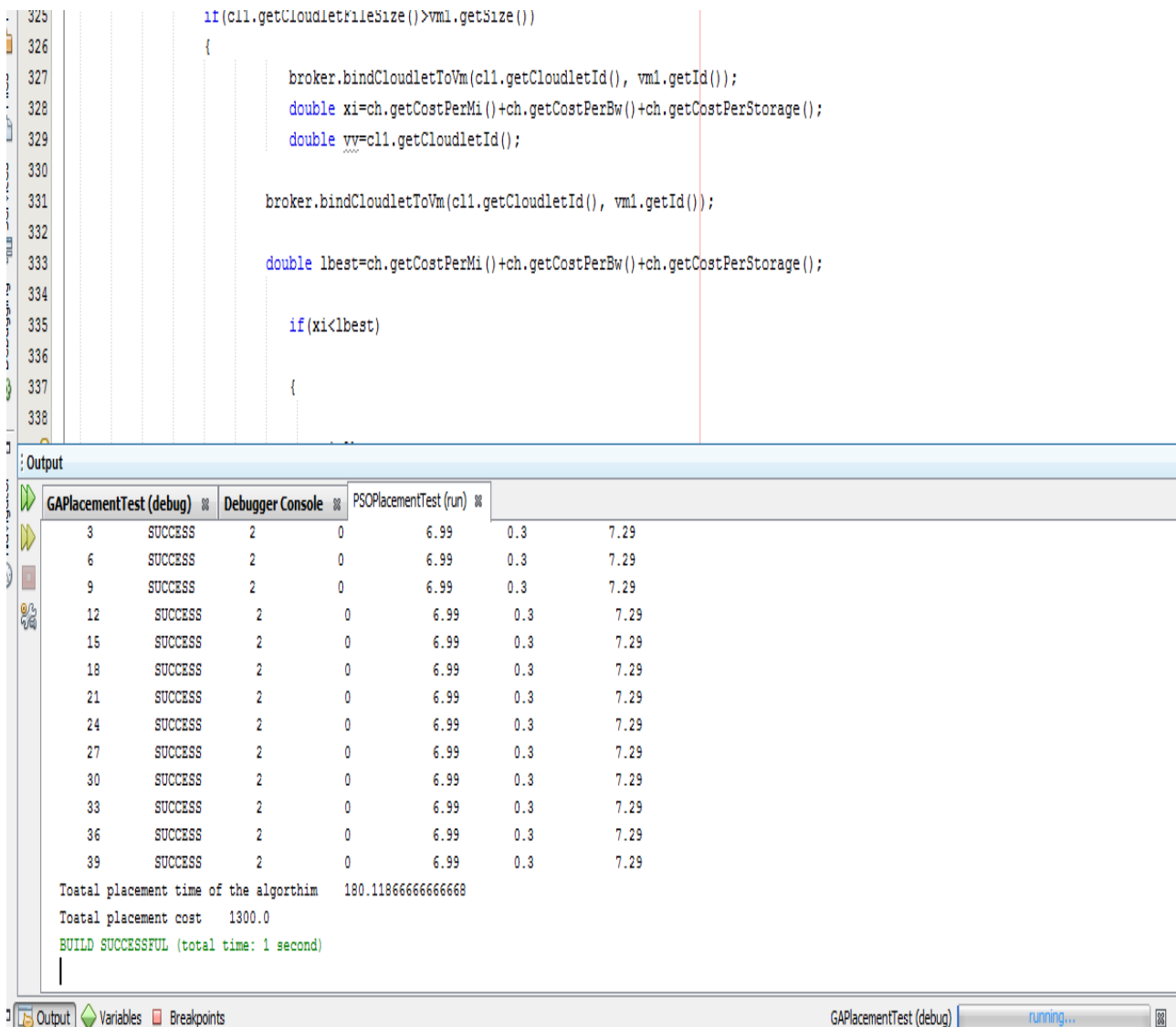
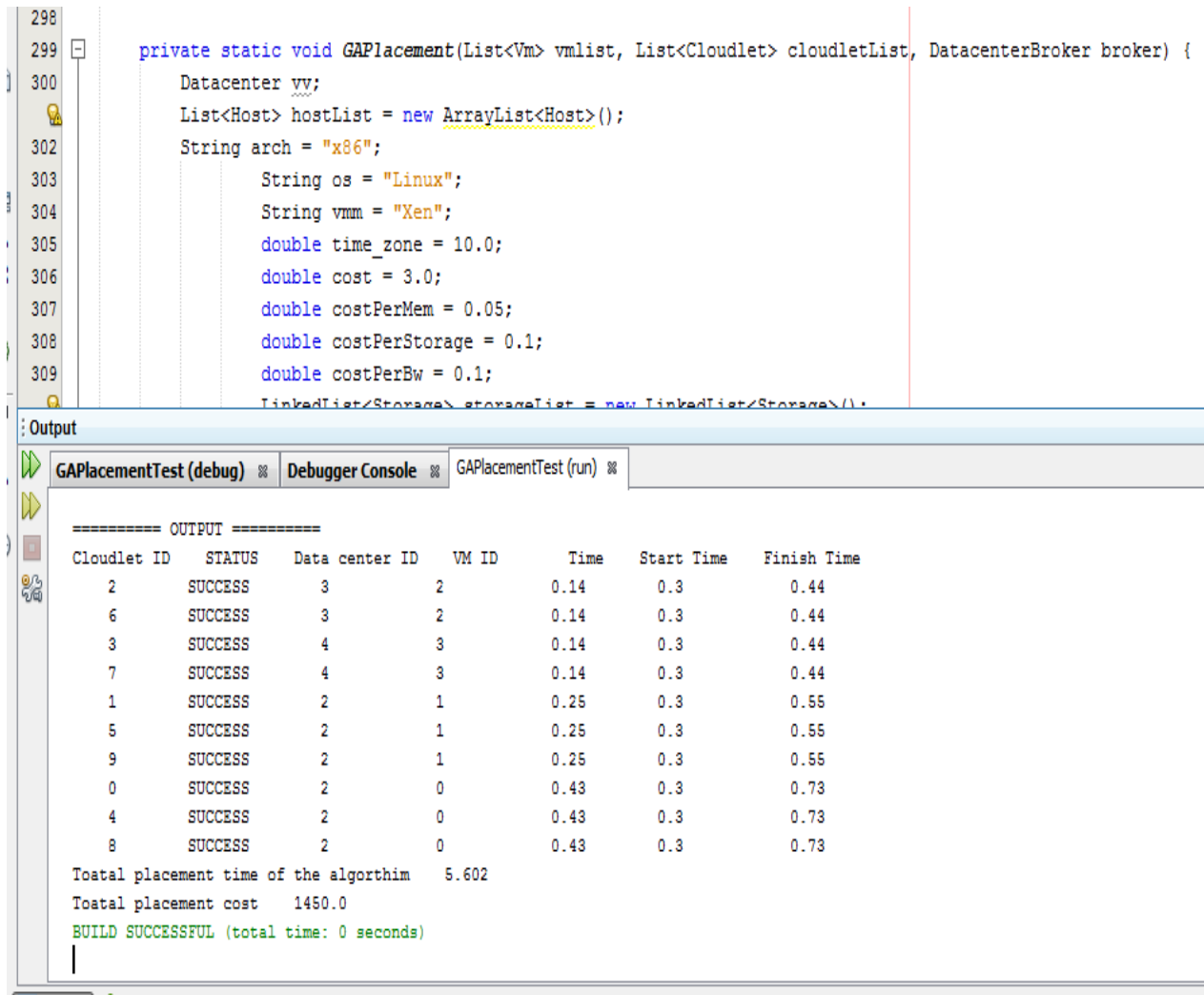


Figure 4.7: Sample Experiment Scenario 4 Simulation Results

4.3.7 Experiment Scenario 5

This experiment demonstration is similar to the above experiment 4. In the previous experiment (experiment 4), the convergence performance of proposed algorithm has evaluated on distributed DC by changing the size of VM resources. In this experiment the impact of VM resources on ST placement in various DC resources; and how the placement scenario using a hybrid approach to minimize placement cost by comparing the proposed algorithm with other existing placement algorithms in a cloud environment. During this experiment, the attribute and the value of DC resource component have been set infrastructure attribute value and change the parameter each time to investigate the impact of the algorithm in placement scenario across the DC.



```
298
299 private static void GAPlacement(List<Vm> vmList, List<Cloudlet> cloudletList, DatacenterBroker broker) {
300     Datacenter vv;
301     List<Host> hostList = new ArrayList<Host>();
302     String arch = "x86";
303     String os = "Linux";
304     String vmm = "Xen";
305     double time_zone = 10.0;
306     double cost = 3.0;
307     double costPerMem = 0.05;
308     double costPerStorage = 0.1;
309     double costPerBw = 0.1;
    LinkedList<Storage> storageList = new LinkedList<Storage>();
}
```

Output

GAPlacementTest (debug) || Debugger Console || GAPlacementTest (run) ||

```
===== OUTPUT =====
Cloudlet ID  STATUS  Data center ID  VM ID  Time  Start Time  Finish Time
2           SUCCESS    3             2    0.14      0.3        0.44
6           SUCCESS    3             2    0.14      0.3        0.44
3           SUCCESS    4             3    0.14      0.3        0.44
7           SUCCESS    4             3    0.14      0.3        0.44
1           SUCCESS    2             1    0.25      0.3        0.55
5           SUCCESS    2             1    0.25      0.3        0.55
9           SUCCESS    2             1    0.25      0.3        0.55
0           SUCCESS    2             0    0.43      0.3        0.73
4           SUCCESS    2             0    0.43      0.3        0.73
8           SUCCESS    2             0    0.43      0.3        0.73

Total placement time of the algorithm  5.602
Total placement cost  1450.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 4.8: Sample Experiment Scenario 5 Simulation Results

Summary

This chapter presented, the simulation set up the environment for implementing the proposed solutions using CloudSim simulation environment. In this chapter, in the first activity has configured the simulation environment. In the first simulation setup configuration, the cloud environment infrastructure has configured based on the placement scenario using amazon infrastructure model. And the second simulation activity the simulation scenario was defined by classifying into three simulation scenario. In the first simulation scenario, the algorithms are implemented by changing the number of VM and fixed the total number of ST components. In the second simulation scenario, the algorithms are also implemented by changing the number ST components and constant the number of VM. Finally, scenario three, four and five has implemented the algorithms by fixed and changing the VM and ST components across DC. In the next chapter, the experiment simulations result has analyzed and evaluated the performance of the proposed solution with those existing algorithms that are discussed in the literature section.

CHAPTER FIVE: RESULT ANALYSIS AND EVALUATION

5.1 Result Analysis

5.1.1 Introduction

In this section, the proposed algorithm performances are analyzed in initial ST placement problem that implemented by changing the cloud resources such as the size of VM, ST, and DC resources. And also evaluated the result of the proposed solution in contributing the efficient placement ST to optimized the resource. To evaluate the performance of proposed solution and compare the results, we used the simulation result of the demonstration in this experiment with those current existing placement algorithms such as PSO and GA.

5.1.2 Experimental Result On changing Size of VM Resources On Located DC

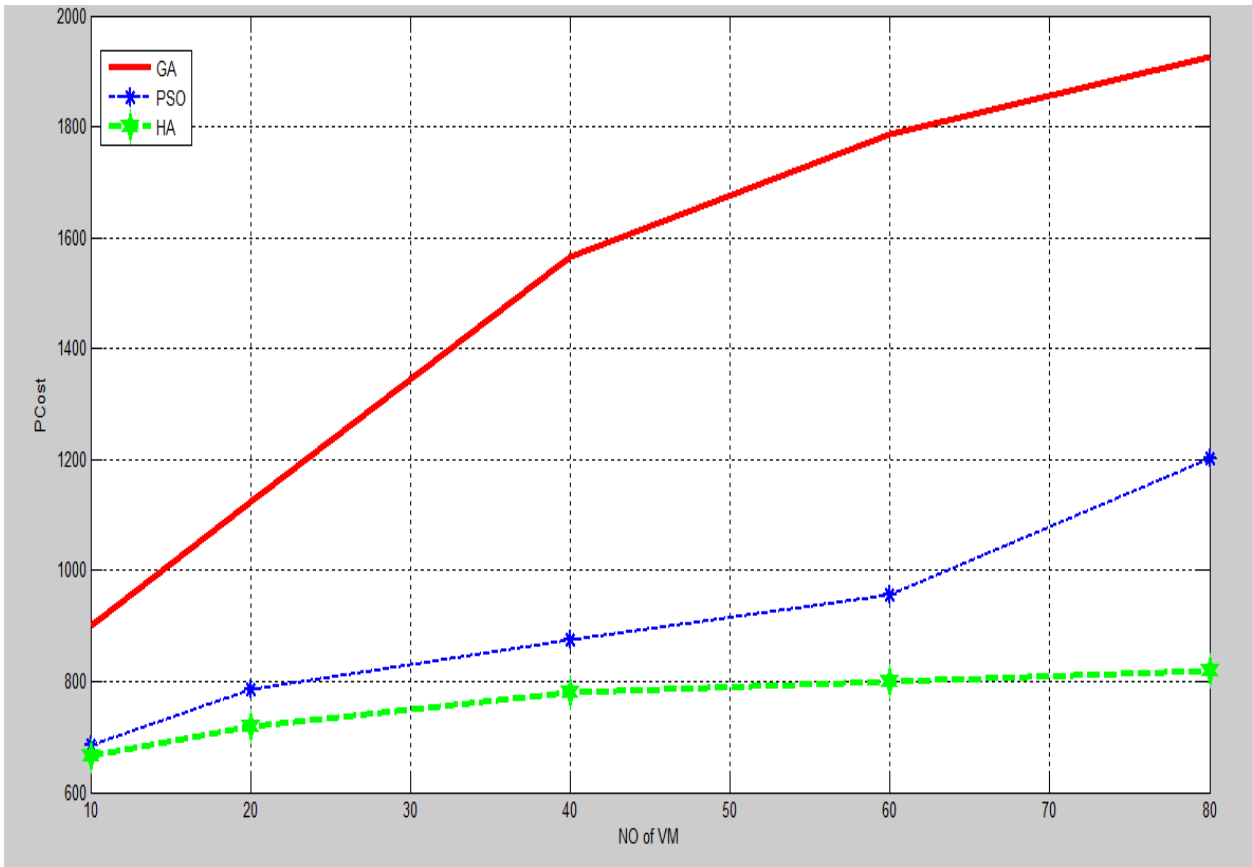


Figure 5.1 Placement of ST Components on Located DC by Varying Number of VMs

In this experiment, how the size of VM resources affects the ST placement has been evaluated on located datacenter. And also, virtual machines which are from 10 to 80 were generated and increased by 10 for each experiment. As can be revealed in figure 5.1, the experiment result shows that in the first three simulations run by changing the VMs from 10 to 40 increased each VMs by 10. In this simulation activity, the GA placement cost is more when we compared to both the PSO and HA approach algorithms. In the first GA algorithm, the placement costs are increased 40% in average and the PSO algorithm increased by 20 % in average. This implies that searching capability of The GA algorithm across virtual machine on located datacenter is not efficient when it compared to PSO algorithm. But, the HA algorithm performance is good, to search optimum VMs resources that compared to both GA and PSO solutions.

In the second simulation test case, changing the number of VM resource from 40 to 80. In each simulation run still, the GA algorithm optimization performance is not efficient. It can be seen that, with a size of VM resources increase, the GA algorithm doesn't produce an optimum placement result. It increases the placement cost dynamically. Because the nature of searching capability of the GA algorithm to converge to the optimum solution when the size of VM resources increase. Therefore, during the changes of the virtual machine, the hybrid approach found an optimal virtual machine resource. This approaches used both searching capability such as local and global to find easily located optimum resources by controlling the searching domain and placement constraints. The hybrid approach, still found optimum placement cost when the proposed solutions compared with the existing placement cost. Thus, the proposed solution saved 7 % and 13 % of placement cost that changed the size of VM that compared to PSO algorithm and GA algorithm respectively.

5.1.3 Experimental Result Changing the Size of ST on Located DC

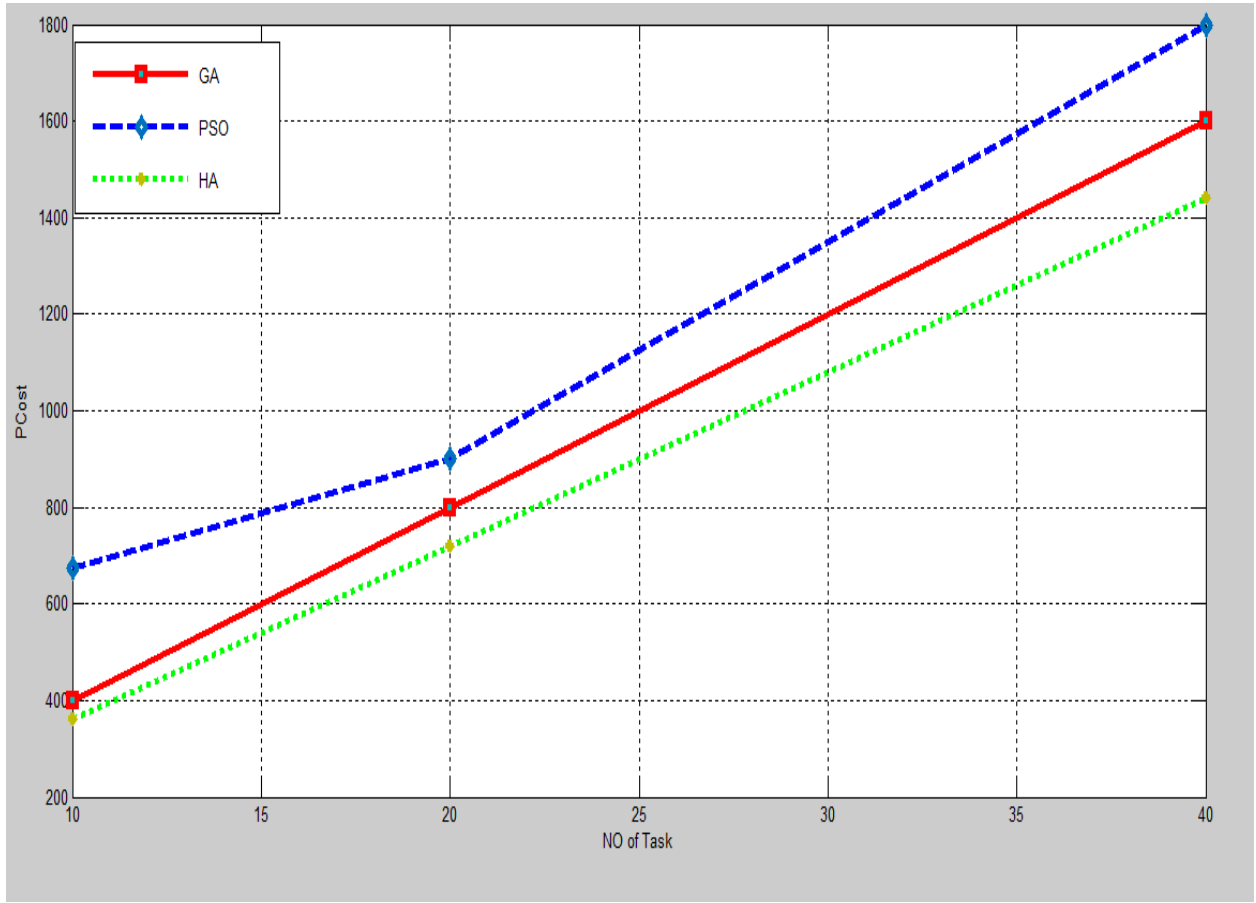


Figure 5.2 Placement of ST Components on Located DC by Varying Number of ST

In the second experiment scenario, we did not limit the size of the ST component. Because to investigate how the size of ST to place in VM resources. In this experiment, the proposed solutions are evaluated in cloud computing ST placement scenario. To evaluate the performance of the proposed solution, the size of STs is changed from 10 to 80, that increased by 10 for each experiment. Figure 5.2 shows that the x-axis represents that the number of ST while the y-axis represents the total placement cost. In the first three simulation activity, the GA and PSO algorithms placement cost are increased in a linear way. The ST component is increased from 10 to 15 and 15 to 20 both GA and PSO algorithms are increased the placement cost by 5% when compared to HA. In this simulation, we investigated that the proposed algorithm was not a great impact in placement optimization.

In the second simulation case in this demonstration, changed the ST components from 20 to 30, the GA algorithm increased the placement cost more. This means the GA algorithm convergence capability is low when the ST components increased more and more. But, the GA algorithm is slightly good when compared to PSO algorithms convergence capability. In third simulation run still, both existing placement algorithms are difficult to optimize the resources. However, the proposed algorithm change the placement cost and optimized the resource by reducing the placement cost by 13 % of the total placement cost.

5.1.4 Experimental Result on Changing Number of DC

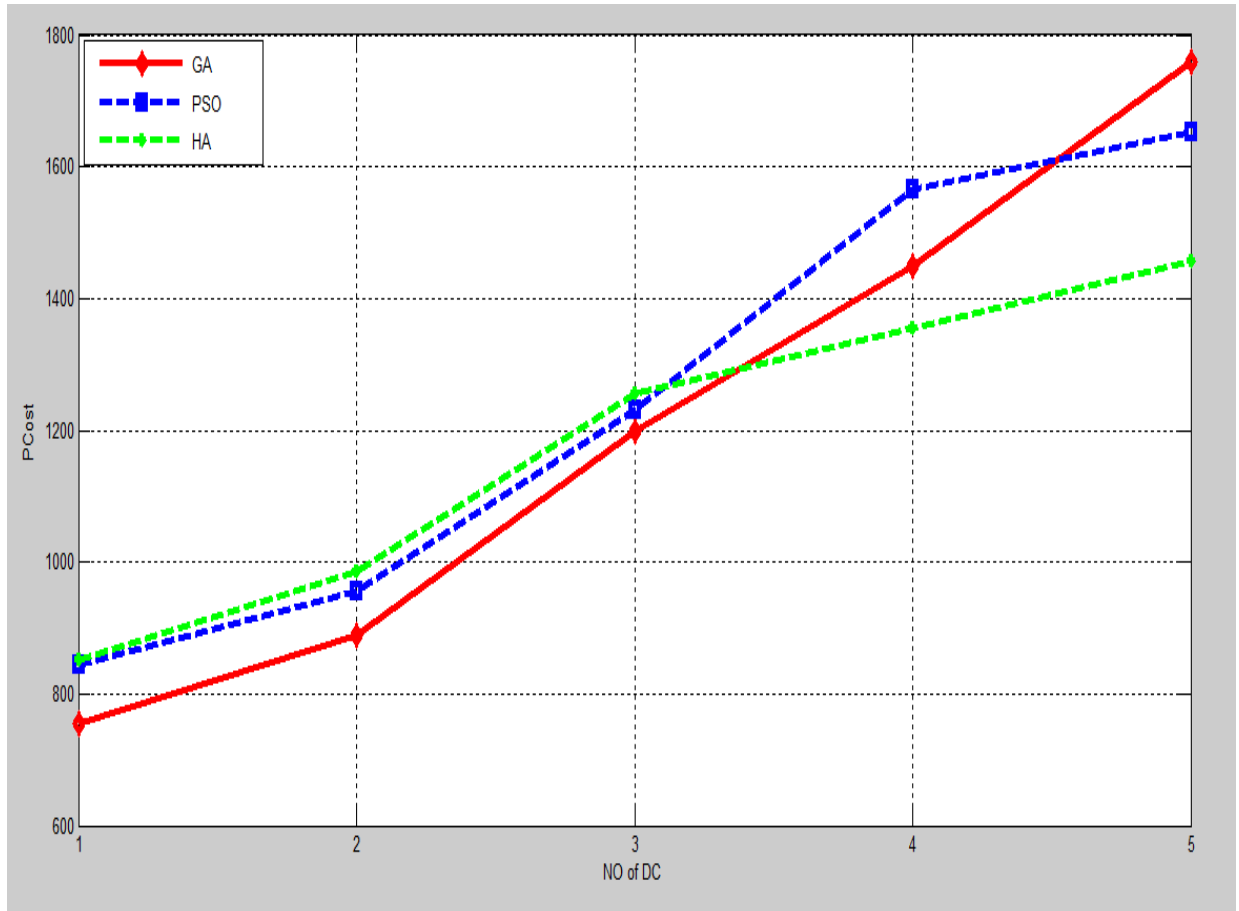


Figure 5.3 Placement of ST Across DC by fixed the VM and ST Resources

The main aim of this experiment is to evaluate the impact of DC location for optimized ST placement in a cloud environment. In a cloud environment, DC are located in the different

region. Each regional DC has its own physical resources. These resources are diverse resources capacities like memory, storage, and processing capability. When the DC locations are increased, we need to place the ST component in a strategic manner to optimize the datacenter infrastructure resources. The experiment performs by changing the size of DC and make constant VM and ST.

In the first experiment, generated similar VM and ST components on located DC. In this experiment, the performance of proposed algorithm has evaluated in single located DC. In each datacenter virtual machine and ST component resources are located. It can be seen that in figure 5.3, the existing placement algorithm such as GA algorithm produced a good result. This result shows that in the first test case the searching capability of GA algorithm in a single DC the optimization performance is efficient. In the second case, increased the DC size into two DCs still the GA placement cost optimum. But, HA algorithm is not produced an optimum placement cost when compared with PSO algorithm. The placement cost was increased by 8%. In the GA algorithm, the placement cost is reduced by 9%. This implies that the both PSO and HA algorithms are optimum when the size of cloud resource is fixed.

In the third case, increased the DC size into 3 DCs. From the experiment result, still, the searching capability of PSO and GA algorithm produced optimum solution compared with HA algorithms. However, in the fourth case, increased the DC size into 4. In this demonstration result, the GA algorithm convergence capability has reduced to find an optimum solution when the DC size increased. But, HA algorithm has reduced the placement cost by 11%. In the last demonstration case, increased the DC size into 5. Still, the HA algorithm produced optimum solutions when compared with GA and PSO algorithms. Therefore, during the placement of the ST component on located DC, the existing solution produced an optimum placement cost but the searching capability is low to find optimum located placement area when the size of DC has increased

5.1.5 Experimental Result on Changing Number of ST Across DC

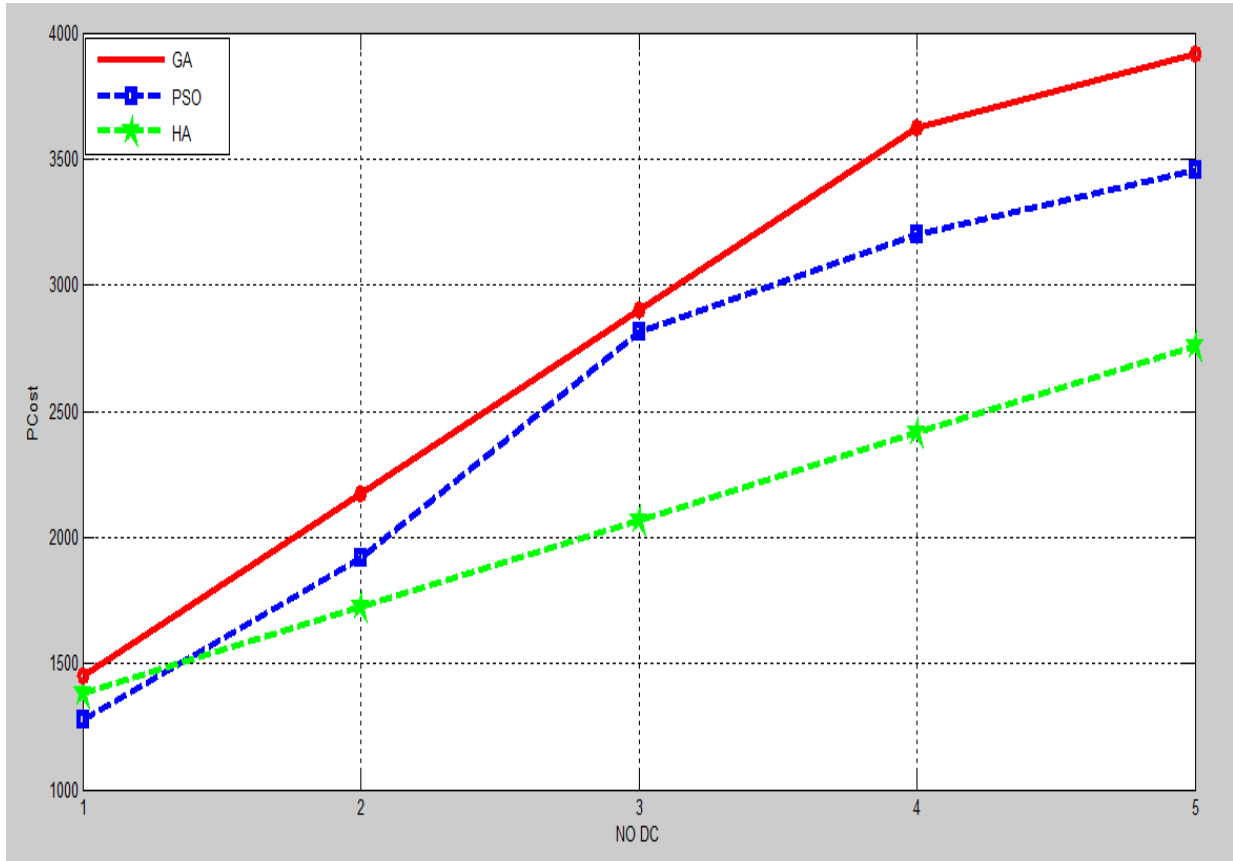


Figure 5.4 Placement of ST Across DC by Changing The size of ST Components

The main aim of this demonstration is to evaluate the impact ST size on distributed DC location for optimum placement in a cloud environment. In a cloud environment, DC are located in the different region. Each distributed DC has its own physical resources. This resource has the different capacity like memory, storage, and processing capability. The experiment performs by changing the size ST and make constant VM on distributed DC location.

In the first simulation run generated 10 VM and 15 ST components on located DC. It can be seen that in figure 5.4, the existing placement algorithm such as GA and PSO algorithm produced optimum results. These illustrations show that in the first test case the searching capability and optimization performance of GA and PSO algorithm are efficient.

In the second case, increased the DC size into 2 and ST 20. That means., still, the GA algorithm is optimum that compared to PSO algorithms. But, the PSO algorithm was more optimum when compared the both PSO and GA algorithm. In the third case, increased the DC size into 3 and ST 25. The GA algorithm has increased the placement cost by 1%. However, the PSO algorithm was decreased by 2 %. This result shows that when the cloud resources are increased the PSO algorithm is an optimum that compared to GA. In this demonstration, the proposed solution is slightly optimum that compared to PSO algorithms and reduced the placement cost by 4%. In the fourth run case, increased the DC size into 4 and ST 25. The GA algorithm is slightly optimum when compared to PSO algorithms. From this experiment results, searching capability of GA across DC is not efficient and the HA algorithm was optimized the placement cost by 4%. That means the convergence capability is high to find the optimum area when the size of cloud resource has increased. Therefore, during the placement of the ST component across datacenter by varying the ST component, the proposed solution produced an optimum placement cost and capability of searching optimum placement area of GA and PSO algorithms are not efficient compared with proposed algorithm.

5.1.6 Experimental Result on Changing Number of VM Across DC

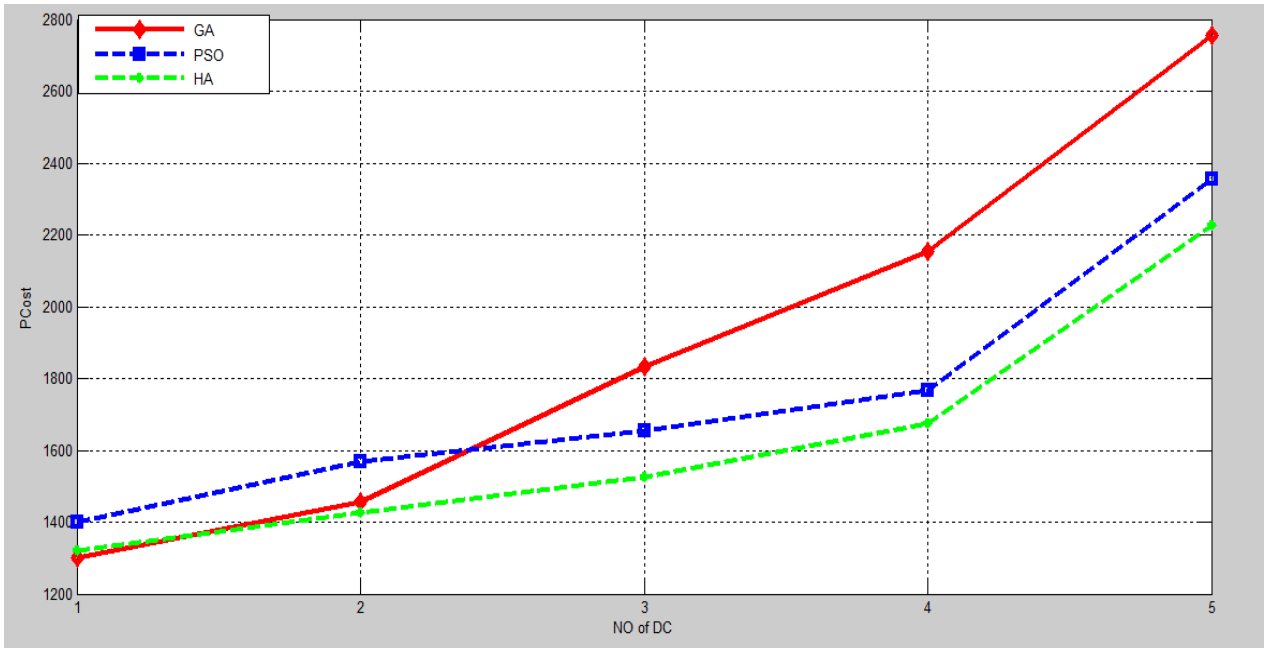


Figure 5.5 Placement of ST Across DC by Changing The VM Resources.

This experiments are similar to the previous experiment 4. But in this experiment the size of ST is constant. The main aim of this demonstration is to evaluate the impact VM size in ST placement. In cloud environment, DC are located in different region. Each regional DC has its own VM resources and the resources have different resources capacities like memory, storage, and processing capability. This experiment performs by changing the size VM and make constant ST across the datacenter.

In the first simulation run, 10 VM and 15 ST components are generated and placed ST components on located DC. It can be seen that in figure 5.5, the existing placement algorithm such as GA and PSO algorithm produced optimum results in a single located DC. These illustrations show that, in the first test case, the searching capability and optimization performance of GA and PSO algorithm are efficient inside a single located DC.

In the second case, increased the size of ST by 20 and constant VM. In this demonstration, still, the PSO algorithm is produced an optimum solution that compared to the GA algorithms. In this demonstration, the PSO algorithm placement cost increased by 7%. But, when compared to HA algorithm, the proposed algorithm cost is increased by 4%. In the third case, increased the DC size into 3 and ST 25. In this demonstration, the GA has increased the placement cost when the cloud resources are increased. It was increased by 8 %. In the fourth run case, increased the DC size into 4 and ST 30. The GA algorithm placement cost is increased more when compared to PSO algorithms. In the final run case, increased the DC size into 5 and ST into 35. The PSO algorithm is slightly optimum when compared to GA algorithms. From this experiment result, searching capability of GA across DC is not efficient. That means the convergence capability is low to find the optimum area when the size of cloud infrastructure resource has increased. Therefore, during the placement of the ST component across datacenter by varying the number of VM, the HA algorithm produced an optimum placement cost and the proposed solution has optimized the placement cost 2- 12 % that compared with PSO algorithm and GA respectively.

5.2 Result Evaluation

In general, the results of the above experiment demonstration show that the proposed solution has optimum the placement cost when compared to other existing placement algorithms. The summary of placement algorithm to finding an optimum placement area on located and across datacenter are presented in *Table 5.1*.

Table 5.1 Placement Algorithm Performance Evaluation on Finding Optimum Placement Area on located and Across DC

Placement Performance Evaluation		
Placement Algorithms	On Located Datacenter	Across Datacenter
Genetic Algorithm	It is efficient finding the optimum solution to place a number of ST components in fixed VM resources but, it is not optimum finding optimum VM resources to place fixed number of ST.	Convergence capability is good when the size of DC and resources are fixed. It is not efficient to find an optimum solution when the size and DC resource has increased.
Particle swarm optimization	It is efficient finding optimum VM resources to place ST components but, it is not efficient to deployed ST resource on VM resources	Convergence capability is slightly good than GA when the size of STs are increased. Identifying an optimum solution is not good when the size of VM and ST is fixed
Hybrid algorithm	It is efficient both finding optimum VM resource and ST placement configuration when the resource size is increase	Convergence capability is good when the size of DC and resources are minimum. It is an efficient to find an optimum solution when the size and cloud resources are increased

In the above five main experiment demonstration results, the performance of proposed algorithms and those existing algorithms are evaluated on located data center by changing the resources and across DC Based on experiment 1 and 2, the GA algorithm performance was slightly good to placing the ST components in fixed number of cloud VM resources that compare to PSO algorithms. But, but the proposed algorithm has produced cost effective placement solutions even the size ST, as well as the number of VM components, are increased. In study [11], the researcher has conducted simulation experiment scenario to evaluate the performance of proposed solution. The results show that such the proposed solution has increased the profit of SaaS vender by decreased the placement cost up to 8 %. In this work, they considered the placement cost including memory cost, storage cost, and bandwidth cost. Whereas we evaluated the performance of the proposed solution by proposed two efficient algorithms, the result shows that optimization of the infrastructure resource cost is minimized by 13%.

In the second existing placement algorithm(PSO), the experimental demonstration 3,4 and 5 results show that the PSO algorithm has produced good placement solution when the size of ST component and VM resources are increased that compare to the GA. Although the performance of PSO algorithm is efficient to optimize the infrastructure resources in order to place ST components on located datacenter and convergence capability of the algorithm is slightly good to identify an optimum placement location across datacenter when compared to GA algorithms. But, the PSO algorithm has inefficient to find an optimum solution when the size of ST and VM that compared to HA. In study [7], the researchers are implemented the PSO algorithm to optimize the placement cost and conducting simulation experiment scenario to evaluate the performance of proposed solution. The results illustrate that such the proposed solution is decreased the placement cost up to 6 %. In this work, they considered the placement cost including memory cost, storage cost, and bandwidth cost. Whereas we evaluated the performance of proposed solutions, the result shows that the proposed solution has minimized the resource up to 12%.

Summary

This chapter presented the demonstration results and evaluation of those existing algorithm and the proposed solution. To compare the performance of the efficiency of the algorithms, five simulation scenario environment were configured. The experiment result in this section shows that, the impact of the proposed hybrid approach for cloud resource management in ST placement activity. In all the above result, the proposed solution has optimized the resource efficiently to provide cost effective service to the user. In the first experiment, the proposed solution identified an optimum DC location by decreasing placement cost by 6 %. It is also minimized the placement cost in the second and third experiment by 7 -13 % by managing the size of ST and VM resources respectively. In the last experiment 4 and 5, the proposed hybrid algorithms were reduced the placement cost by 2- 12%. In general, when we see the above experiment results, the hybrid approach has been optimized the total placement cost across DC in a cloud environment.

CHAPTER 6: CONCLUSION AND FUTURE DIRECTION

This section summarizes the research process throughout the thesis and indicate future directions.

6.1 Conclusion

Placement is the main activity in a cloud environment. In this activity, the provider has used an efficient placement strategy to provide an efficient and a cost effective service to the user by optimizing the infrastructure resources. Optimal ST placement is the process of identifying the best placement area in cloud infrastructure environment. The main aim of this optimal placement is to optimize the placement resource during in the placement scenario in order to provide cost effective service to the user. To achieve the aim of this work, different optimization algorithms (genetic, greedy, bat, particle swarm) has identified and investigated their application area. Based on convergences and robustness capability, the performance of those optimization algorithms was evaluated to implemented and tackle placement problems in a cloud environment and learned these optimization algorithms are more powerful algorithm to solve the different optimization problem in real world scenario. Initial ST placement problem is one of optimization ST placement problem in a cloud environment. To handle this problem, an efficient algorithm is needed.

In order to optimize the placement cost, in this work an efficient ST placement strategy has proposed using hybrid approach and give the detail idea, implementation of proposed algorithm and evaluate the solution. The proposed solution has used to optimize the placement problem especially in the initial stage of ST placement in a cloud environment. To optimize the infrastructure resources, in the initial placement stage, BA and GA hybrid optimization algorithms are proposed. In the first hybrid placement algorithm(BA), it is easy to find an optimum located DC using the BA searching capability. This algorithm was high performance to global searching capability performance. The ST placement has its own constraint that is an obstacle to search an optimum placement area. To handle this constraint and optimize VM resources, the second placement strategy(GA) were proposed. The second algorithm is also the high capability to find an optimum solution using their control attributes.

The initial placement problem is constrained based optimization problem, so the proposed the hybrid approaches was optimized both local and global optimum placement solution across the datacenter.

This study investigated different cloud simulation tools such as CloudSim, Greencloud and DCSim to implemented the algorithms and analyzed the performance of solution to achieve the objectives. In order to identify the best simulation tool different criteria has seated like availability, language independent and platform support. Finally, the CloudSim simulation tools were selected in order to evaluated the performance of the proposed solution. The algorithms are implemented by configuring the experiment setup environment using CloudSim simulation environment.

To evaluated the performance of proposed solution, first the experiment setup environment has configured and generated the resource using amazon models. In this study five main simulation experiment scenario has conducted by changing the cloud infrastructure resource such as DC, VM and ST components. In the first two experiment, the proposed solution was tested on located DC. In this experiment, the proposed algorithm convergence capability to find the best placement location were optimum. The proposed algorithm produced a good placement solution by minimizing the placement cost. In the third experiment, the proposed solutions are evaluated by fixed the size of ST and VM resources on distributed DC. In this demonstration, the searching capability of GA and PSO algorithms has produced optimum solution when compared to the proposed solution on located DC. But, both existing algorithms has not produced efficient solution when the located DC is increased. In the fourth and fifth experiment demonstration, the experiments performed by changing the size of ST components and VM resources. In this demonstration, the experimental result shows that the proposed solution has optimized the placement cost efficiently. Finally, the hybrid approach produced an optimum solution that compared with the existing placement algorithms.

Generally, placement in cloud environment especially ST placement is a big issue and interesting research area based on its significant for those players in a cloud environment such as infrastructure provider, application vender and users. The strategic ST placement may save cloud infrastructure resource and provide cost effective services to the user.

6.2 Future Direction

The main objective of this research is to find an optimum placement area for initial ST placement in cloud environment. During this research, the impact of proposed solution in the first stage of ST component placement has investigated and online placement software component are out of our consideration. Placement of ST is the challenging problem in cloud environment. The following are the recommended future works:

- Test the performance of the proposed solution in to real infrastructure environment.
- It needs detail investigation of the problem by integrating the online placement and scalability issue.
- Design an efficient algorithm that is more dynamic to easily handle the placement constraint.

REFERENCES

- [1] C. McLellan, "ZDNet," 1 Augest 2016. [Online]. Available: www.saas in 2016:the keytrends/ZDNet. [Accessed 12 October 2016].
- [2] Zhenzhang Liu ,Zhihui Hu ,Leek K.Jonepun, "Composite saas Placement Problem Based on Anti Colony Optimization Algorithm," *Journal of Digital Information Management* , vol. 12, no. 4, pp. 225-234, 2014.
- [3] Zeratul Izzah Mohd Yusoh and Maolin Tang, "Composite SaaS Placement and Resource optimization in Cloud Computing Using Evolutionary Algorithm," in *EEE International Conference* , Hawaii, 2012.
- [4] Z. I. M. Yusoh, "Composite SaaS Resource Management in Cloud Computing using Evolutionary Computation," Queensland University of Technology , Austeralia, 2013.
- [5] Sumit Bhardwaj and Bibhudatta Sahoo, "A particle swarm optimization Approach For Cost Effective SaaS Placement on cloud," in *international conference on Computing,Communication and Automation* , India , 2015.
- [6] I. Amazon Web Service, "Amazone Web Service," Amazon Web service ,Inc , 2016. [Online]. Available: <https://aww.amazone.com/S3/>. [Accessed 12 Dec 2016].
- [7] Bhardwaj, S., & Sahoo, B(, "Particle Swarm Optimization Approach for Cost Effective SaaS Placement on Cloud.," INDIA, 2015.
- [8] M. Erascu, "A scalable Hybrid Approache for Application Placement in Cloud Enviroment," pp. 1-4, 3 April 2015.
- [9] Rajkumar Buyya, *Mastering Cloud Computing Foundations and Applications Programming*, USA: Morgan Kaufmann(Elsevier), 2013.
- [10] D. Reviews, "DeZyre for Business," DeZyre, 11 Apr 2015. [Online]. Available: Cloud Computing vs. Distributed Computing.htm. [Accessed 1 May 2016].
- [11] A. Kumar, "Placement of software-as-a-service components in cloud computing environment," INDIA, June, 2014.
- [12] A. Krishnasway, "Cloud Reference Architecture," India, 2012.
- [13] Andy Mulholland, "The Cloud and SOA," Capgemini, 2008.

- [14] E. Gorelik, "Cloud Computing Models," Massachusetts Institute of Technology , Cambridge , January 2013.
- [15] Wei-Tek, "Service Orinted Cloud Computing Architecture," in *Interenational Conference on Information Technology*, China , 2010.
- [16] P.-O. O. stberg, "Virtual Infrastructures for Computational Science: Software and Architectures forDistributed Job and Resource Management," Umea university , Sweden, 2011.
- [17] S. Somepalle, "linked in," 23 Apr 2015. [Online]. Available: <https://www.linkedin.com/pulse/3-service-4-deployment-models-cloud-computing-sankar-somepalle>. [Accessed 25 Apr 2016].
- [18] VictorVictories, "IMB Web site IBM blog," IBM, 19 Aug 2015. [Online]. Available: www.ibm.com/developerworks/community/blogs/722f6200-f4ca-4eb3-9d648d2b58b2d4e8/entry/4_Types_of_Cloud_Computing_Deployment_Model_You_Need_to_Know1?lang=en ,. [Accessed 25 Mar 2016].
- [19] CloudTweaks, " CloudTweaks," 2 July 2012. [Online]. Available: <http://cloudtweaks.com/2012/07/4-primary-cloud-deployment-models/>. [Accessed 12 Jun 2016].
- [20] [Online]. Available: www.google.com.et/?gws_rd=ssl#q=deployment+models+in+cloud+pic+. [Accessed 3 Jun 2016].
- [21] V. Beal, "Webopedia," IT busness Edge , 2016. [Online]. Available: <http://www.webopedia.com/TERM/V/virtualization.html>. [Accessed 12 Nov 2016].
- [22] S. Kang et al, "A General Maturity Model and Reference Architecture for SaaS Service," *Springer-Verlag*, vol. II, no. II, p. 337–346, 2010.
- [23] M. Levinson, "CIO," CIO, 15 May 2007. [Online]. Available: www.SaaS.htm. [Accessed 23 Jun 2016].
- [24] S.Satyanarayana, "CLOUD COMPUTING : SAAS," *Computer Science and Telecommunications*, vol. 36, no. 4, 2012 .
- [25] O. Erel, "Saas Adcit," 16 May 2015. [Online]. Available: [SaaS Delivery Model in Cloud Computing - SaaS Addict.htm](http://SaaS-Delivery-Model-in-Cloud-Computing-SaaS-Addict.htm). [Accessed 10 June 2016].
- [26] M. T. Professionals, " IT knowlgde portal," 2016. [Online]. Available: [Software as a Service \(SaaS\).htm](http://Software-as-a-Service-(SaaS).htm). [Accessed 12 April 2016].

- [27] A. Company, "Amazon Web Service," Amazon Inc, 2017. [Online]. Available: <https://aws.amazon.com/application-hosting/>. [Accessed 12 January 2017].
- [28] A. Inc, "Software- as- a- Service (SaaS) on AWS," Amazon Inc., 2010.
- [29] M. Morley, "Insights 1st Edition," 2013. [Online]. Available: [How do SaaS Environments Work.htm](http://www.morley.com/insights/How-do-SaaS-Environments-Work.htm). [Accessed 26 Feb 2016].
- [30] K. K. M. Kumar, "SOFTWARE AS A SERVICE FOR EFFICIENT CLOUD COMPUTING," *International Journal of Research in Engineering and Technology*, vol. 03, no. 01, pp. 178-181, Jan-2014.
- [31] Gianpaolo, " Microsoft developer site date of access," Microsoft , 6 March 2006. [Online]. Available: <https://blogs.msdn.microsoft.com/gianpaolo/2006/03/06/saas-simple-maturity-model>. [Accessed 25 April 2016].
- [32] M. F. Lohrer, "A Comparison Between the Firefly Algorithm and Particle Swarm Optimization," Oakland University, 2013.
- [33] He, Xin-She Yang and Xingshi, "Firefly Algorithm: Recent Advances and Applications," *Int. J. Swarm Intelligence*, vol. I, no. 1, pp. 36-50, 2013.
- [34] K. Danesh, "Slide Share," In Techonlogy, 27 Jun 2013. [Online]. Available: [https://www.Practical Swarm Optimization \(PSO\).htm](https://www.PracticalSwarmOptimization.com). [Accessed 16 May 2016].
- [35] Selim Yilmaza And Ecir U. Küc, üksille, "A new modification approach on bat algorithm for solving optimization problems," *Applied Soft Computing*, pp. 259-275, 25 Nov. 2014.
- [36] D. W. Dyer, 2010. [Online]. Available: <http://watchmaker.uncommons.org/manual/ch01.html>. [Accessed 12 August 2016].
- [37] Er. A.K.Mishra, Dr. M. N. Das, Dr. T. C. Panda, "Swarm Intelligence Optimization: Editorial Survey," *International Journal of Emerging Technology and Advanced Engineering*, vol. III, no. 1, pp. 217-230, 2013.
- [38] M. Skinner, "Genetic Algorithms Warehouse," [Online]. Available: <http://geneticalgorithms.ai-depot.com/>. [Accessed 12 Sep 2016].
- [39] M. University of carlos, "Carlos University," [Online]. Available: <http://tracer.uc3m.es/tws/ps/>. [Accessed 12 Oct 2016].
- [40] M. Dorigo, "scholarpeda," 2007. [Online]. Available: http://www.scholarpedia.org/article/Ant_colony_optimization. [Accessed monday Sep 2016].

- [41] J. Brownlee, "Clever Algorithms: Nature-Inspired Programming Recipes," 2015. [Online]. Available: www.IEEEepapaers/greedy%20approche.htm. [Accessed 12 Sep 2016].
- [42] A. C. a. A. Smith, 09 Jan 2007. [Online]. Available: <http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume28/coles07a-html/node11.html>. [Accessed 20 Jan 2017].
- [43] Arpit Sengar & Aarju Gangwar B, "cloudsimpresntation," 2014 . [Online]. Available: www.cloudSimautherStream.com. [Accessed 3 july 2015].
- [44] A. Steven, "docplayer.net," 10 Feb 2016. [Online]. Available: <http://docplayer.net/1412031-Comparison-of-various-cloud-simulation-tools-available-in-cloud-computing.html>. [Accessed 2 August 2016].
- [45] A. Oujani, 24 Aril 2013. [Online]. Available: <http://www.cse.wustl.edu/~jain/cse567-13/ftp/cloud/index.html>. [Accessed 4 August 2016].
- [46] Al-Sakib,Muhammad,Shafiullah Khan , "Simulation technique in networking and communication," USA, CRS Press , 2015, pp. 316-330.
- [47] Tang, Zeratul Izzah Mohd Yusoh and Maolin, "A Penalty-based Genetic Algorithm for the Composite SaaS," in *World Congress on Computational Intelligence*, Barcelona, Spain, 2010.
- [48] Ting Yu,Limin Wang,Ying Liu and Li Zhang, "Swarm Intelligence Optimization and Their Application," in *Wuhan International Conference on e-Business*, China, 2015.
- [49] Mitesh_Soni, "OpenSource," Creative Commons Attribution-NonCommercial , 3 March 2014 . [Online]. Available: [he CloudSim Framework Modelling and Simulating the Cloud.htm](http://heCloudSimFrameworkModellingandSimulatingtheCloud.htm). [Accessed Monday June 2016].
- [50] Jayshri Damodar Pagare and Dr. Nitin A Koli, "Design and simulate cloud computing environment using cloudsim," *Computer Technology & Applications*, vol. IV, no. 1, pp. 35-42, 2015.
- [51] Tsai, W. T., Huang, Y., Bai, X., & Gao, J., "Scalable architectures for SaaS," in *In Object/component/service-oriented real-time distributed computing workshops (isorcw)*, 2012, Apri.
- [52] M. Vasileiadi, "cloud compruting and software-as-a-service weaknesses strengths and proposed architectures," Kingston, JULY 2010.
- [53] A. Shields, "Overview: Cloud computing services and deployment models," Market Realist, Inc. , 23 Jul 2014. [Online]. Available: [Why 'SaaS' is an integral component of cloud computing - Market Realist.htm](http://Why'SaaS'is-an-integral-component-of-cloud-computing-Market-Realist.htm). [Accessed 2 Mar 2016].

- [54] S.-a.-a.-S. E. Council, "Software-as-a-Service: A Comprehensive Look at the Total Cost of Ownership of Software Applications," Sep 2006.
- [55] Judith M. Myerson, "IBM DevelopersWorks," IBM, 07 April 2009. [Online]. Available: The role of Software as a Service in cloud computing.htm. [Accessed 03 Feb 2016].
- [56] Juho, "Ask Question," 15 Jul 2015. [Online]. Available: www.nphardproplem.htm. [Accessed 12 Mar 2016].
- [57] Dong Yuan, Yun Yang, Xiao Liu, Jinjun Chen, "A Data Placement Strategy in Scientific Cloud Workflows," Swinburne University of Technology, Australia , 2012.
- [58] Qing Zhao, Congcong Xiong, Kunyu Zhang, Yang Yue and Jucheng Yang, "A Data Placement Algorithm for Data Intensive," *International Journal of Grid and Distributed Computing*, vol. 9, no. 2, pp. 145-156, 2016.
- [59] Qiang Xu, Zhengquan Xu, Tao Wang, "A Data-Placement Strategy Based on Genetic Algorithm Cloud Computing," *International Journal of Intelligence Science*, vol. 5, no. 1, pp. 145-157, 2015.
- [60] S. (. Ltd., "Spacetal," 26 Mar 2015. [Online]. Available: <http://spacetal.co.uk/spacetal-insights/finding-the-right-cloud-infrastructure-vendor-for-your-saas-application/>. [Accessed 10 Nov 2016].
- [61] A. w. services, "Amazone global infrastructure," Amazone web services,inc, 2016. [Online]. Available: <https://aws.amazone.com/about-aws/global-infrastructure/>. [Accessed 12 Dec 2016].
- [62] C. McLellan, "ZDNet," ZDNet, 1--15:25 GMT (08:25 PDT) August 2016. [Online]. Available: https://SaaS%20in%202016%20%20The%20key%20trends%20_%20ZDNet.htm. [Accessed 2 Dec 2016].
- [63] Falatah, M. M., & Batarfi, O. A., "Cloud scalability considerations," *International Journal of Computer Science and Engineering Survey*, vol. 5, no. 4, 2014.

APPENDICES I: Simulation Environment Configuration

DC Configuration

```
Datacenter datacenter0 = createDatacenter("Datacenter_0");
Datacenter datacenter1 = createDatacenter("Datacenter_1");
Datacenter datacenter2 = createDatacenter("Datacenter_2");
private static Datacenter createDatacenter (String name) {
    List<Host> hostList = new ArrayList<Host> ();
    List<Pe> peList1 = new ArrayList<Pe> ();
    int mips;
    peList1.add (new Pe (0, new PeProvisionerSimple(mips)));
    List<Pe> peList2 = new ArrayList<Pe> ();
    int hostId, ram;
    long storage;
    int bw;
    hostList.add (new Host (hostId, new RamProvisionerSimple(ram),new BwProvisionerSimple(bw),
    storage peList1, new VmSchedulerTimeShared(peList1)));
    hostId++;
}
```

Datacenter Characteristics Configuration

```
String arch = "x86", os = "Linux", vmm = "Xen";
double time_zone = 10.0;
double cost, costPerMem, costPerStorage, costPerBw;
LinkedList<Storage> storageList = new LinkedList<Storage> ();
DatacenterCharacteristics characteristics = new DatacenterCharacteristics (
    arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);
Datacenter datacenter = null;
private static List<Vm> vmList;
private static List<Vm> createVM (int userId, int vms) {
```

```

LinkedList<Vm> list = new LinkedList<Vm> ()

for (int i=0; i<vms; i++) {

    vm[i] = new Vm (i, userId, mips, pesNumber, ram, bw, size, vmm, new
    CloudletSchedulerTimeShared ());

```

ST configuration

```

private static List<Cloudlet> createCloudlet (int userId, int cloudlets

LinkedList<Cloudlet> list = new LinkedList<Cloudlet> (

UtilizationModel utilizationModel = new UtilizationModelFull ();

Cloudlet [] cloudlet = new Cloudlet[cloudlets];

for (int i=0; i<cloudlets; i++) {

cloudlet[i] = new Cloudlet (i, length, pesNumber, fileSize, outputSize, utilizationModel,
utilizationModel, utilizationModel);

cloudlet[i]. setUserId(userId);

list.add(cloudlet[i]);

} return list;
}

```

Broker configuration

```

DatacenterBroker broker = createBroker ();

int brokerId = broker. getId ();

private static DatacenterBroker createBroker () {

DatacenterBroker broker = null;
try {
broker = new DatacenterBroker("Broker");
} catch (Exception e) {

e. printStackTrace ();

return null;
}
return broker;
}

```

APPENDICES II: Algorithms

GA algorithm

```
Random md= new Random ();
while (cloudletList.size () <0)
{
for (Cloudlet cl1: cloudletList)
{
for (Vm vm1: vmList)
{
if (cl1.getCloudletFileSize()> vm1.getSize())

broker. bindCloudletToVm (cl1.getCloudletId(), vm1.getId());
double cbest=cl1.getCostPerSec();
if (cl1.getCloudletLength()>0)
{

for (Cloudlet cl2: cloudletList)
{
for (Vm vm2: vmList)
{
broker. bindCloudletToVm (md. nextInt(), cl2.getVmId());

}
}

for (Cloudlet cl2: cloudletList)
{
for (Vm vm2: vmList)
{
int vmid;
vmid= md. nextInt(vm2.getId());
broker. bindCloudletToVm (vmid, cl2.getVmId());
double nn=cl2.getCloudletLength() * df. getCostPerBw ();
System.out.println(nn);

}
}
}
```

PSO algorithm

```
for (Cloudlet cl1: cloudletList)
{
    for (Vm vm1: vmList)
    {
        if (cl1.getCloudletFileSize() > vm1.getSize())
        {
            broker. bindCloudletToVm (cl1.getCloudletId(), vm1.getId());
            double xi=ch. getCostPerMi () + ch. getCostPerBw () + ch. getCostPerStorage ();
            double vv=cl1.getCloudletId();
            broker. bindCloudletToVm (cl1.getCloudletId(), vm1.getId())
            double lbest=ch. getCostPerMi () +ch. getCostPerBw () +ch. getCostPerStorage ();
            if(xi<lbest)
            {
                xi=lbest;
            }
            broker. bindCloudletToVm (cl1.getCloudletId(), vm1.getId());
            double gbest=ch. getCostPerMi () +ch. getCostPerBw () +ch. getCostPerStorage ();
            if(lbest<gbest)
            {
                gbest=lbest;
            }
            // broker. bindCloudletToVm (cl1.getCloudletId(), vm1.getId());
            // double gbest=cl1.getProcessingCost();
            // p.nextInt (vmin);

            for (int i=0; i<10; i++)
                for (Cloudlet cl3: cloudletList)
                {
                    double v[]={2};
                    double x[]={3};
                    v[i]= v[i]+ p. nextDouble () * (v[i]-v[i])+p.nextDouble()*(v[i]-v[i]);
                    x[i]=x[i]+v[i];

                    //vValue = Particle.velocity() + 2 * new Random().nextDouble() * (aParticle.pBest() -
                    //testResults) + 2 * new Random().nextDouble() * (bestResults - testResults);
                }
        }
    }
}
```

BA-GA algorithm

```
Random p= new Random ();
int bats;
int generations;
double loud=0.5;
double pulse=1;
double Qmin = 0;
double Qmax = 2;
int N_iter = 1;
Datacenter dd = null;
double xf=p.nextInt();
while(N_iter >12)
{
    for (int i=1; i<N_iter; i++)
    {
        int Q[]={};
        double x[]={2};
        double v[]={3};
        Q[i] = (int) (Qmin+((Qmin-Qmax)*p.nextDouble()));
        v[i] = v[i]+((x[i]-x[i])*Q[i]);
        x[i]=x[i-1]+v[i];
    }
    double d1=ch.getCostPerBw()+ch.getCostPerMem()+ch.getCostPerStorage();
    if(d1> ch.getCostPerBw()+ch.getCostPerMi()+ch.getCostPerMem())
    {
        broker.getId();
    }
    for(Cloudlet cl1:cloudletList)
    {
        for(Vm vm1 :vmList)
        {
            if(cl1.getCloudletFileSize()>vm1.getSize())
            {
                broker.bindCloudletToVm(cl1.getCloudletId(), vm1.getId());
                double xi=ch.getCostPerMi()+ch.getCostPerBw()+ch.getCostPerStorage();
                double vv=cl1.getCloudletId();
            }
            else
            for (Cloudlet cl2: cloudletList)
            {
                for (Vm vm2 :vmList)
                {
                    broker.bindCloudletToVm(p.nextInt(),cl2.getVmId());
                }
            }
        }
    }
}
```

```

for (Cloudlet cl2:cloudletList)
{
for (Vm vm2 :vmList)
{
int vmid;
vmid= p.nextInt(vm2.getId());
broker. bindCloudletToVm (vmid, cl2.getVmId());
broker. bindCloudletToVm (cl1.getCloudletId(), vm1.getId());
double vc=ch. getCostPerMi ()+ ch.getCostPerBw() + ch.getCostPerStorage();
double vmcost=ch.getCostPerMi() +ch.getCostPerMem()+ch.getCostPerStorage();
double tccost=cl2.getCloudletOutputSize()*ch.getCostPerBw();
broker.bindCloudletToVm(cl1.getCloudletId(), vm1.getId());
double lbest=ch.getCostPerMi()+ch.getCostPerBw()+ch.getCostPerStorage();
if(vc<lbest)
{
vc=lbest;
}
if(tccost<lbest)
{
tccost=lbest;
}
if(p.nextDouble()>pulse && vmid <vc && tccost < lbest )
{
pulse++;
loud++;
}
}
}
}

```