

Developing a Hybrid Heuristic and Metaheuristic Scheduling Algorithm (HHMH) for Cloud-Fog Computing Environment



By

Abebe Aychiluhum Gebre

A Thesis Submitted to the Department of Computer Science and Engineering
School of Electrical Engineering and Computing

Office of Graduate Studies

Adama Science and Technology University

February 2021
Adama, Ethiopia

Developing a Hybrid Heuristic and Metaheuristic Scheduling Algorithm (HHMH) for Cloud-Fog Computing Environment



By

Abebe Aychiluhum Gebre

Advisor: Dr-Ing Frezewd Lemma

A Thesis Submitted to the Department of Computer Science and Engineering
School of Electrical Engineering and Computing

Office of Graduate Studies

Adama Science and Technology University

February 2021
Adama, Ethiopia

APPROVAL OF BOARD OF EXAMINERS

We, the undersigned, members of the Board of Examiners of the final open defense by Abebe Aychiluhum Gebre have read and evaluated his thesis entitled “***Developing a Hybrid Heuristic and Metaheuristic Scheduling Algorithm (HHMH) for Cloud-Fog Computing Environment***” and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirement of the Degree of Masters Computer Science and Engineering.

Name	Signature	Date
<u>Abebe Aychiluhum</u>	_____	_____
Name of Student		
<u>Dr-Ing Frezewd Lemma</u>	_____	_____
Advisor		
_____	_____	_____
External Examiner		
_____	_____	_____
Internal Examiner		
_____	_____	_____
Chair Person		
_____	_____	_____
Head of Department		
_____	_____	_____
School Dean		
_____	_____	_____
Postgraduate Dean		

DECLARATION

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

Name

Signature

Abebe Aychiluhum

This MSc thesis has been submitted for examination with my approval as a thesis by

Name: Dr-Ing Frezewd Lemma

Signature: _____

Date of submission: _____

ACKNOWLEDGEMENT

I am here today on the brink of eyes to declare their accomplishment, to be a nominee of Computer Science and Engineering graduate, for all of this hill climb achievement, almighty GOD that made this for me, absolute praise for GOD. Beyond this, I would like to thank my advisor Dr-Ing Frezewd Lemma for his dedicated support and guidance. Dr-Ing Frezewd was continuously encouraged and was always willing and enthusiastic to assist in any way he could throughout the research project. Finally, many thanks to all participants that took part in the study and enabled this research to be possible.

ABSTRACT

Constant Internet of Things (IoT) developments has been producing vast volumes of data in recent years, placing pressure on the infrastructure of cloud computing (CC). To meet the criteria set by the IoT devices, the Fog computing (FC) architecture is considered the next generation of CC. It consequently removes the big issue of higher delays in the cloud-IoT paradigm, but CC cannot be replaced by FC. The processing power of the Fog Node is limited and the cloud must be added to form a Cloud-Fog (CF) computing model. Inefficient scheduling of user tasks in CF computing can result in higher response time and execution time and also minimize the throughput. Therefore, it is only possible to obtain the real benefits of CF computing by applying efficient task scheduling (TS) strategies. Because of this, the thesis proposes an efficient hybrid scheduling algorithm (HHMH) for CF computing, which is a hybrid of Modified Particle Swarm Optimization (MPSO) and First Come First Serve (FCFS). These algorithms have certain drawbacks. FCFS algorithm is the simplest task scheduling technique but it has the problem of inefficient resource utilization and long waiting time. MPSO has a problem with response time, execution time, and throughput when the task and environment are heterogeneous. The key aim of this thesis is to minimize the average response time and average execution time, to improve the usage of VMs, and to increase throughput by scheduling the tasks effectively. The proposed algorithm was tested and simulated on Cloudsim by using 11 datasets of 40 up to 500 number tasks with different sizes. Experimental results show that compared to MPSO, FCFS, and SJF, the developed HHMH algorithm lowers the average response time by 54.91%, 11.5%, and 2% for 500 tasks respectively. For 500 tasks, we measured the throughput and found that the developed HHMH algorithm performs 2.19 tasks per second, while MPSO performs 2.01 tasks per second, FCFS performs about 1.88 tasks per second, and SJF performs 2.03 tasks per second. Calculated execution time and VM usage of the developed HHMH algorithm is also better.

Key Words: Cloud Computing, Cloud-Fog Computing, Fog Computing, Heuristic Algorithm, Internet of Things, Task Scheduling

TABLE OF CONTENTS

ACKNOWLEDGEMENT.....	i
ABSTRACT	ii
LIST OF FIGURES	vii
LIST OF TABLES	viii
LIST OF ABBREVIATIONS AND ACRONYMS	ix
CHAPTER 1.....	1
1 INTRODUCTION.....	1
1.1 Background of the Research	1
1.2 Motivation of the Study	5
1.3 Statement of the Problem.....	7
1.4 Hypothesis	8
1.5 Objectives	8
1.5.1 General Objective	8
1.5.2 Specific Objectives	8
1.6 Significance of the Research.....	9
1.7 Concerns of Ethical Conduct	9
1.8 Scope and Limitations of the Research.....	10
1.9 Organization of the Thesis	10

CHAPTER 2	11
2 LITERATURE REVIEW	11
2.1 Introduction.....	11
2.1.1 Overview of task scheduling in Cloud and fog computing.....	12
2.1.2 Types of Scheduling	13
2.1.3 Phase of Scheduling	14
2.1.4 Existing Task Scheduling Algorithms	14
2.1.4.1 Heuristic Task Scheduling Approaches	15
2.1.4.2 Metaheuristic Task Scheduling Approaches.....	18
2.1.4.3 Hybrid Scheduling Algorithm.....	21
2.1.5 Parameters for Comparing Algorithms of Scheduling.....	22
2.2 Related Works.....	22
CHAPTER 3.....	25
3 RESEARCH METHODOLOGY	25
3.1 Overall Methodology	25
3.2 Research Strategy	25
3.3 Research Method.....	26
3.4 Research Approach.....	26
3.5 Data Collection Method and Tools.....	26
3.6 Research Process.....	27

3.7	Simulation Environment	27
3.8	Research Metrics.....	27
3.9	Testing and Methods of Evaluation	28
3.10	Development Tools and Techniques	28
3.10.1	Implementation Techniques and Tools	28
3.10.2	Designing Tools.....	28
CHAPTER 4		29
4	DESIGNING PROPOSED SOLUTION	29
4.1	Proposed Solution Architecture.....	29
4.2	Problem Formulation	31
4.3	Scheduling of Tasks on VM Using Proposed (HHMH) Algorithm	32
CHAPTER 5		37
5	EXPERIMENTAL SETUP AND IMPLEMENTATION.....	37
5.1	Experimental Setup.....	37
5.2	Implementation	39
5.2.1	Common Simulation Configuration Activities	40
5.2.2	Algorithm Architecture and Description	40
5.2.2.2	SchedulerPaverage response timeicleUpdate.java	41
5.2.2.3	SchedulerFitnessFunction.java	42
5.2.2.4	HHMHScheduler.java.....	43

CHAPTER 6	44
6 EVALUATION AND DISCUSSION	44
6.1 Introduction.....	44
6.1.1 Utilization of VMs.....	44
6.1.2 Average Response Time	45
6.1.3 Execution Time Calculation	46
6.1.4 Throughput	48
CHAPTER 7	54
7 CONCLUSION AND FUTURE WORKS.....	54
7.1 Conclusion	54
7.2 Future Works	55
Special Acknowledgment	56
8 REFERENCES	57
Appendix A List of Java Codes.....	66
A.1 HHMH-Scheduler.....	66
A.2 DatacenterCreator	69
A.3 SchedulerFitnessFunction.....	70
A.4 SchedulerPaverage response timeicle.....	71
A.4 FCFSDatacenterBroker	72

LIST OF FIGURES

Figure 1-1 Cloud computing scheduling classification [1]	1
Figure 1-2 Distributed fog computing data processing [10].....	3
Figure 1-3 Cloud-Fog system architecture [11]	4
Figure 2-1 Classification of scheduling scheme in the cloud and fog computing [29]	15
Figure 4-1 HHMH system model	29
Figure 4-2 Sequence diagram of cloud-fog architecture [6].....	30
Figure 4-3 Flow diagram of the developed algorithm.....	34
Figure 6-1 VM utilization in terms of tasks performed by each VMs	45
Figure 6-2 Average response time comparison between HHMH Vs FCFS, SJF, and MPSO	46
Figure 6-3 Execution time comparison between HHMH Vs FCFS, SJF, and MPSO	47
Figure 6-4 Throughput comparison between HHMH Vs FCFS, SJF and MPSO.....	49

LIST OF TABLES

Table 2.1 Parameters for Comparing Scheduling Algorithm [21]	22
Table 2.2 Summary of related work finding	24
Table 5.1 Characteristics of the Cloud-Fog infrastructures.....	37
Table 5.2 Attributes of tasks.....	38
Table 5.3 Simulation setup	38
Table 5.4 Sample data set of 40 cloudlets	38
Table 6.1 Number of tasks (400) handled by the VMs	44
Table 6.2 Detail of task and calculated average response time of algorithms.....	45
Table 6.3 Detail of the task and calculated execution time of algorithms.....	47
Table 6.4 Simulation details to find the throughput of the algorithms.....	48

LIST OF ABBREVIATIONS AND ACRONYMS

ABC.....	Artificial Bee Cloney
ACO.....	Ant Colony Optimization
BFO	Bacterial Foraging Optimization
CC.....	Cloud Computing
CF	Cloud Fog
CPS	Continuous Production System
CSA	Crow Search Algorithm
CSP	Crow Service Provider
DEA.....	Differential Evolution Algorithm
FC	Fog Computing
GA	Global Best
IDEA.....	Improved Differential Evolution Algorithm
MCT	Minimum Completion Time
MET.....	Minimum Execution Time
MM.....	Min-Min
MSDE.....	Moth Search Algorithm and Differential Evolution
MxM.....	Max-Min
PBBO.....	Priority-Based Bi-Objective
PSA.....	Priority Scheduling Algorithm
PSO.....	Particle Swarm Optimization
RR.....	Round Robbin
SA	Simulated Annealing
SJF	Shortest Job First
RGB.....	Red-Green-Blue
SLA.....	Service Level Agreement

CHAPTER 1

1 INTRODUCTION

1.1 Background of the Research

The National Institute of Standards and Technology (NIST) considered cloud computing as a framework for allowing universal, easy, accessing network whenever required to a collective configurable computing resources pool. Cloud resources can be provided to users as pay-as-you services, where consumers pay for the services they use only. The main services of the cloud are split into three groups. They are Platform as a Service (PaaS), Infrastructure as a Service (IaaS), and Software as a Service (SaaS). The virtualized infrastructure allows cloud users to access resources (processing, storage, and network) on-demand using IaaS services. Amazon Web service (AWS) is the biggest competitor in the market for IaaS services. Consumers can get rid of the physical infrastructure's upfront costs by using this service. In the PaaS service, cloud users can use the cloud-supplied interface to access the PaaS service for creating new applications. Ex: Java, .NET. SaaS is a service on which we will be able to use the software provided by the cloud platform. Ex: Google docs, Gmail.

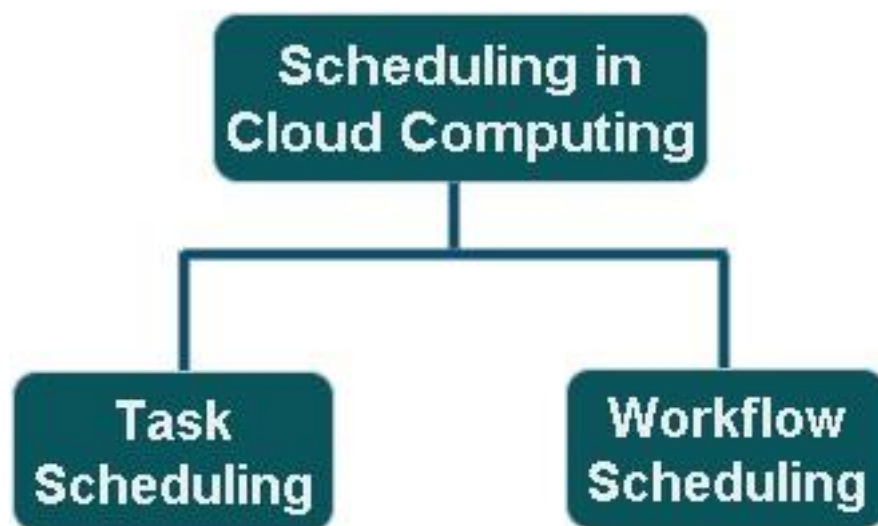


Figure 1-1 Cloud computing scheduling classification [1]

One of the major challenges in cloud computing is scheduling because the workload in the cloud is heterogeneous and complex, thus effectively providing the resources is required. In two ways, scheduling can be grouped as shown in [Figure 1-1](#), i.e., Task Scheduling and Workflow Scheduling (WS). TS can be defined as assigning a given job or task for a specified period to virtualized resources independently i.e., tasks are independent [1]. WS shall be interpreted as assigning the resources in the distributed system to the interdependent tasks [2]. VM utilization, Wait Time, Response Time, Turnaround Time, Memory, and Throughput are the metrics that are affected by task scheduling. The proposed study focused on task scheduling.

In the ICT industry, IoT has recently been one of the big revolutions. IoT [3] assures many products such as consumer electronic devices, home appliances, medical devices, cameras, and all other forms of sensors. This opens the door to technologies that promote things and people and allows smart cities, infrastructure, and services to be realized that increase the quality of life.

By 2025, analysts predict that the IoT will have an economic effect of \$11 trillion per year, including revenue produced and operating savings, which could reflect about 11% of the global economy that users would deploy one trillion IoT devices [4]. IoT environments produce extraordinary quantities of information that can be useful in several ways. However, the data volume can overwhelm today's storage systems and analytics applications.

A recent study of thirty million consumer healthcare-related IoT applications revealed data flows of even more than 25,000 records per second [5]. In addition, in smart cities with more data sources, real-time data flows could easily increase each second. The processing capacity of most devices cannot be reached due to this rapid rise in IoT [6].

Cloud computing is a centralized architecture traditionally used for information processing and storage for IoT applications. IoT devices are growing over time resulting in an explosive increase in energy consumption and a deterioration in performance [7]. In addition, for delay-sensitive IoT applications, the high delay in the cloud-IoT model renders it less realistic. Energy and performance-conscious computing and storage facilities have become highly important in this context [8]. Cloud computing expands to beat these problems and forms the edge or fog computing at the tip of the network that performs the processing.

Fog computing, which combines edge devices and cloud services seamlessly, helps to solve the constraints of centralized Cloud architecture. By exploiting cloud services and organizing the jobs of geographically dispersed edge devices, it prevents resource contention at the edge [9]. It supplies cloud services toward the corner of the network.

Fog computing is included in the elements of data processing or analytics systems operating in the cloud as well as end devices. It also promotes management and programming for storage, computing, and networking facilities between data centers and end devices. Moreover, it supports the mobility of users, resource, and device heterogeneity, and distributed data analytics to meet the need for distributed applications with low latency.

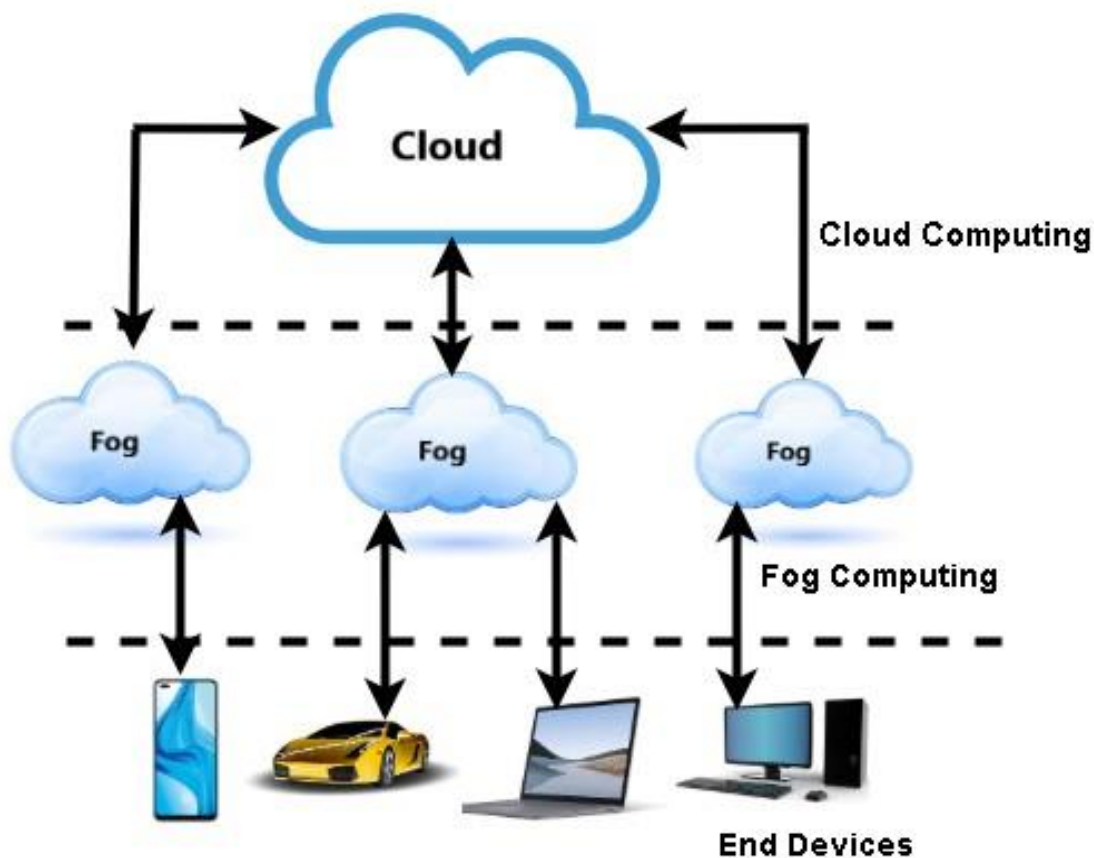


Figure 1-2 Distributed fog computing data processing [10]

Fog computing aims to handle a portion of the workloads created by applications that are called Fog servers or Fog nodes on devices near the network and users. These devices are also installed anywhere that has a network connection: warehouses, shopping centers, power poles, railways, within cars, etc. The node of a Fog can be any system that has storage, processing capability,

and networking capabilities. Figure 1-2 shows that users can deploy the IoT sensors in many environments including highways, medical centers, and farms assisted by the necessary functionality of a system. Fog systems, including nearby gateways and personal clouds, perform data analytics dynamically once the system gathers information from the sensors.

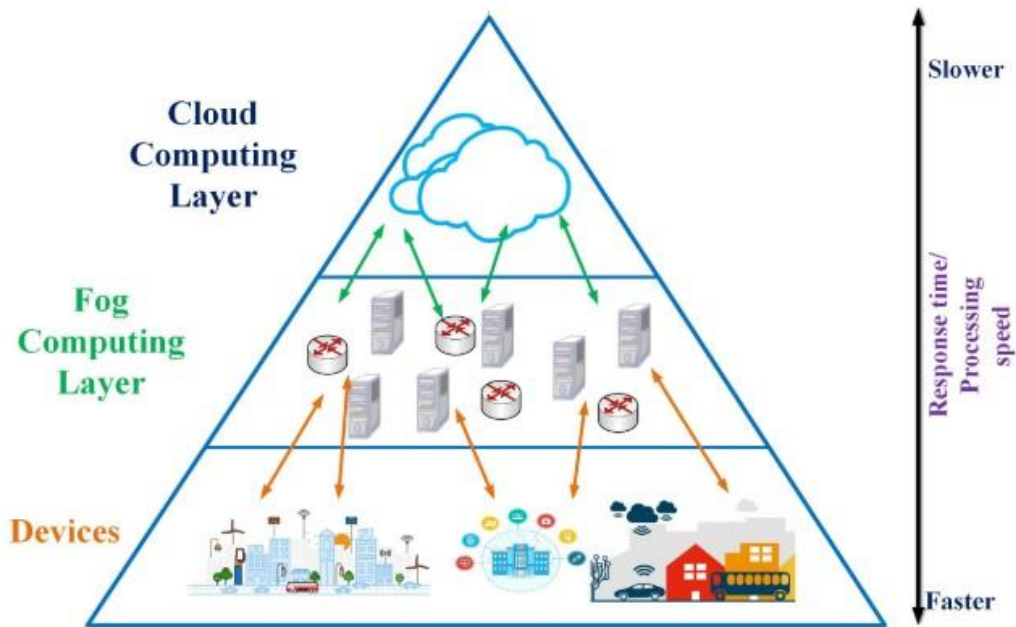


Figure 1-3 Cloud-Fog system architecture [11]

The time for data to reach a processing station can be negligible by putting resources at the network's end, so the tasks are designed for synchronized universal time. However, the Fog Node processing capacity is minimal. If the incoming tasks do not find any match from fog devices, then the tasks are directed to the clouds for processing. Fog does not replace cloud computing. Eventually, to create a brand-new computing model, Cloud-Fog computing, fog computing complements the Cloud. If cloud computing uses fog computing, it reduces latency, provides fast computing, and reduces the processing price [11]. Figure 1-3 demonstrates the Cloud-Fog structure.

There are numerous advantages to this new computing model, including decreased latency, network traffic, and increased energy efficiency, but it also has certain challenges. Task scheduling within the processing node pool, including cloud nodes (such as servers or virtual machines) and fog nodes, is the most difficult task [12]. Under the Cloud-Fog scheme, the desire of task scheduling is oriented for the good of consumers or service providers.

On the user's side makespan, budget, deadline, security, latent time, and cost are concerned. Load balancing, throughput, VM utilization, and energy efficiency are the priorities of the service providers. To guarantee QoS, response time covers an important part that directly affecting the experience of the consumer. The SLA signed with users will be met by an efficient task-scheduling algorithm, which minimizes completion time and saves monetary expense.

This thesis is concentrated on the task-scheduling problem like response time, execution time, and throughput in a cloud-fog environment, a highly distributed computing platform, for processing large scale of tasks and applications. Accordingly, the study suggests a new scheme for efficient resource management, specifically task scheduling in cloud-fog computing. This approach is a hybrid of two algorithms namely FCFS and MPSO. According to this study, the scheduler finds the most effective match of fog devices for an incoming task based on different metrics. Its demand for time of CPU and memory are major bases for measuring and estimating the incoming task. If it does not find any match from fog devices, then it sends the task to the clouds and it will be processed on that.

1.2 Motivation of the Study

Task scheduling is a system of allocating incoming requests or tasks to the available resources within the computing system. The allocation of the resource extraordinarily influences the performance, availability, and responsiveness of the given system. Therefore, a suitable scheduling mechanism is an essential issue in the situation of cloud computing and fog computing. Efficient algorithms shall be applied for enhancing the overall performance of the computing world.

Many types of research have been done in the area of task scheduling. Many different developed task scheduling algorithms were used to schedule or map incoming tasks on the available resource of cloud computing. In addition, different studies trying to develop scheduling algorithms for a new computing approach known as fog computing. However, mostly the developed algorithms for scheduling applied now in fog computing are comes from cloud computing [13].

A common scheduling algorithm is not enough to solve the problems of resource management that occurred in cloud computing as well as fog computing. Additionally, developing a task-scheduling algorithm specifically for cloud computing or fog computing surely cannot meet the requirement of the computing partner. So instead of developing an algorithm for cloud and fog computing specifically, it is a good idea to combine the two-computing approach into the cloud-fog approach and develop optimal task scheduling for the combined computing approach to be successful in fulfilling the requirement of the user of the system.

The known algorithms developed for task scheduling in cloud computing, as well as fog computing, are MM, FCFS, RR, GA, PSO, and so on [13]. From those algorithms, the proposed study focuses on the two scheduling algorithms of the cloud and fog computing architecture called FCFS and MPSO algorithm. Even though FCFS has many limitations, cloud computing uses it for scheduling purposes. However, the algorithm is straightforward and it is effortless to implement it with other algorithms. MPSO, one of the scheduling algorithms, is used in cloud computing. Many researchers were developed to advance this algorithm.

Nevertheless, the study in [13, 14] shows that the MPSO algorithm has a limitation on response time and execution time when the task is dynamic and heterogeneous. This study determined that the issue/problem could probably resolve by implementing it with other heuristic scheduling algorithms. Therefore, the exciting thing that motivates us is, FCFS enhances and improves the problems of the MPSO algorithm within the search space and this work allows implementing a better algorithm for Cloud-Fog computing. The other motivation in the area is, mostly FCFS and MPSO were used for Cloud-Fog computing but no research had proposed by combining those algorithms. However, this thesis has used these algorithms for the cloud-fog computing environment together.

For evaluating the proposed work, a simulation toolkit is used instead of a real environment, due to its less expensiveness and takes less time than experiments with a real environment. In addition to this, a simulation toolkit is easy to understand, verified and communicated [15, 16]

1.3 Statement of the Problem

The primary intention of this research is to solve problems in the existing cloud and fog-computing environment and to boost the performance of the system. This study identifies different problems of algorithms of scheduling that are used in cloud computing as well as fog computing.

The centralized cloud architecture is facing several challenges from IoT applications [6]. The main problem of the traditional architecture of cloud computing is the high latency and high response time of the system for IoT applications. Due to this problem of the centralized architecture of cloud computing, developing an optimal task-scheduling algorithm with reduced average response time is very difficult. So, the modification of existing cloud computing architecture to a new computing paradigm helps to get the full potential optimized capability of this thesis. The proposed algorithm is implemented and applied for the new computing paradigm (Cloud-Fog).

However, the focus of the study is on the scheduling algorithms used in the computing system. Task scheduling is the most crucial part of the cloud resource management system and also should execute efficiently to benefit both the service providers and customers [13]. Task scheduling aims to distribute the incoming tasks from users to the available virtual machines and concerns Load Balancing, resource allocation, execution time, performance, and response time [13]. One of the benchmarks and simplest scheduling algorithms applied in the environment of cloud computing is the FCFS algorithm [13, 17, 18, 19].

FCFS algorithm is the simplest task scheduling techniques. It uses the task arrival time to schedule the task. The task will be scheduled and executed, depending on which task has arrived first in the queue. It depends on arrival time and does not consider any other parameter. The tasks are scheduled by selecting the accurate job order. The task or user request, which comes first will be assigned to VM first for execution. This leads to the submitted task used the available resources unevenly and which makes the FCFS scheduling algorithm inefficient in resource utilization. Moreover, if a task with demanding more processing capacity comes first in the queue list, then all the other tasks with less processing time will have to wait till the task is executed, which makes the waiting time longer in FCFS [13, 20, 21].

The further available algorithm for scheduling in cloud computing is MPSO. It is a metaheuristic algorithm suitable for optimizations and load balancing. However, it has a problem with response time, execution time, and throughput when the task is dynamic and heterogeneous. In addition, this algorithm also has the above-mentioned problem when the environment is heterogeneous. It works well when the environment and task are homogeneous [13, 14, 22].

In general, the proposed approach tries to propose a modified and crossbred algorithm for scheduling the arriving tasks to a requested resource in a highly distributed computing environment called the Cloud-Fog computing system. The algorithm takes advantage of FCFS and MPSO algorithms and tries to fill the limitations of those algorithms by combining them

1.4 Hypothesis

Scheduling tasks properly can improve resource utilization, throughput, average response, and execution time. Hence, the criteria to evaluate the proposed approach would be VM utilization, throughput, average response, and execution time. The null(H_0) and alternative(H_1) hypothesis are given below:

- **H_0 :** There will be no difference in VM utilization, throughput, and average response time in the system after scheduling using the proposed algorithm.
- **H_1 :** The proposed approach will be better in average response time, throughput, and VM utilization in the cloud-fog environment.

1.5 Objectives

1.5.1 General Objective

The general objective of this thesis is to develop a hybrid task scheduling algorithm for the cloud-fog computing environment.

1.5.2 Specific Objectives

The following specific objectives are identified to achieve the main objective:

- To study the related works done in the area of cloud and fog task scheduling.

- To prepare data that can be input for the implementation of the proposed study.
- To implement the proposed algorithm
- To simulate the proposed algorithm in the simulation setting.
- To compare peer algorithms with the result of the proposed approach in terms of selected measurement metrics.

1.6 Significance of the Research

Scheduling is vital because it will have an enormous impact on resource utilization and overall system performance. The proposed study benefits all cloud computing users, particularly the cloud service provider can gain the benefit from the result. For the user, the proposed approach would allow accessing and using the available resources of Cloud-Fog computing quickly and flexibly. For the cloud provider, it allows meeting the requirements of the consumer based on the SLA. Generally, without a practical task scheduling algorithm, the service provider may not fulfill the user requirements that they want from the provider. The proposed solution has the following benefits.

- The proposition of a unique hybrid algorithm (MPSO and FCFS) for scheduling tasks in the Cloud-Fog environment.
- The proposed study would give contributions to task scheduling in Cloud-Fog computing to optimize VM utilization, throughput and to reduce the response time and execution time.

1.7 Concerns of Ethical Conduct

This work unreservedly thanks fellow researchers who dedicated their researches to the open-source community and widely available open technology. Moreover, it promotes the work done by the other researchers by giving relevant links and references to the site where it resides. The writer tried to work to amend based on the previous researchers to promote the further enhancement and contribute to the community diligently by demonstrated valid findings without falsification, abundantly, and mischief. Also, to open future ideas and development by conduct extensively.

1.8 Scope and Limitations of the Research

Management of resources is a challenging task in a distributed environment. The proposed approach is focused on proposing a task-scheduling algorithm for a highly distributed computing environment known as cloud-fog computing. This proposed work combined the two scheduling algorithms, namely FCFS and MPSO algorithm, to develop an algorithm for the cloud-fog environment.

The primary goal of this work is to optimize VM allocation and throughput and reduce the average response time in the cloud-fog environment. However, there are other concerns about scheduling algorithms that are not covered by this research. One of the limitations of this work is, the experiment and implementation of the proposed approach are performed only on Cloudsim rather than the real environment. Moreover, budget, deadline, and limitations of resources will be the constraints for greater practicality.

1.9 Organization of the Thesis

Chapter one discussed the introduction, the motivation, and the statement of problems, the hypothesis, the scope and limitation, the intention of the study, the methodology, and the significance of application results. The next chapter presents about literature review. It includes a literature review, an overview of the task scheduling algorithm, and related works. Chapter three discusses the methodology of a proposed solution, which includes a method for proposed work, data collection method, development methods and tools, techniques, simulation environment, and research metrics. Chapter four discusses designing the proposed solution which includes the proposed solution and its architecture and also the algorithm in detail. The fifth chapter describes the experiment environment setup. The sixth chapter presents the results, and analysis as well as performance evaluation. Chapter seven completes and concludes the research constructed on the findings and proposes future work.

CHAPTER 2

2 LITERATURE REVIEW

2.1 Introduction

These days, numerous advances are produced for conveying boundless quantities of processing assets to the recipient or end-client. One of the processing stages structured and created for this design is cloud computing. End clients can utilize the processing resource from anyplace and whenever whereby clients pay for the items they use. Because of this reality, both the IT portion and academics are very tolerating cloud computing. To give cloud infrastructure to clients and to keep up enormous databases and server farms, the web has a huge impact on cloud computing.

Moreover, cloud technology utilizes processing assets to give computer resources to the end-client. In particular, cloud computing is a strategy that uses virtualization technologies that can deal within the data center or server farm with various cloud properties. In other words, this applies to the management of workloads in the data center over the Internet. By using cloud computing elimination of cost for equipment, applications, controls, and data center management are possible. Furthermore, cloud computing allows for data backup, recovery from disaster, and guarantees business continuity with easier and cheaper services [23].

Various clients may need various kinds of assets found in the cloud supplier server farm or datacenter. In that scenario, the cloud supplier should address the request of the buyer in like manner. Luckily, cloud computing can offer various assets for various uses dependent on the request of the user. In the cloud, services like SaaS, PaaS, and so forth are found and conveyed to the shopper as they are mentioned their request for services. Regardless, the requirement for computational force and the number of customers in distributed computing is constantly expanding. Quickly expanding cloud servers, just as buyers, are the fundamental driver of the expansion in power consumption.

The other significant explanation behind the expansion in energy consumption is improper cloud resource management. Cloud buyers get to processing assets that are conveyed by the cloud supplier and they will be charged dependent on the use of these services. Nevertheless,

cloud computing has natural issues starting a few applications cannot work proficiently on it. Henceforth, fog computing was declared to eliminate these limitations of cloud computing [24].

Fog computing is another kind of computing platform or design that is at first proposed to stretch out services of cloud computing to the end-client. It is found between the cloud server farm and IoT applications. It gives a cloud-like help nearer to the end-client. Right now, numerous innovations are utilizing it, for example, Continuous Production System, medicinal services system, and virtual reality and gaming. Fogging is the other term to describe Fog and it was proposed by CISCO at the absolute starting point. Wireless information transfer among dispersed frameworks in the IoT applications is perhaps exceptionally simple over fog computing. It made processing assets and applications services near the end-client where information is being created. Moreover, it diminishes the amount of information coordinated to the cloud for preparing to an enormous level while decreasing the burden of cloud computing and permits to keep up a load balance between assets or resources [3, 24].

Fog computing can create a different virtual system like virtual storage, virtual processing, and so forth. But the created virtual system is near to the level where the information is being produced. This new computing paradigm appears between cloud and IoT applications and it acts as middleware. A big percentage of the request is processed in the layer of fog locally and the remaining data will be pushed and proceed in the cloud. Cloud can react to a lot of information than fog computing. Due to this fact, it has many advantages in combination with cloud computing. By using fog computing it is possible to save bandwidth, the latency, and the packet loss will be reduced to a large extent. Though fog has these types of advantages, it cannot fully function without cloud computing [25].

2.1.1 Overview of task scheduling in Cloud and fog computing

In cloud computing task scheduling is a significant task that needs to be implemented properly. However, it is the most challenging concern that affects the overall performance of cloud computing as well as fog computing. It is a method of mapping or plotting various tasks on to resources and systems at a specific time efficiently. The resources can be CPU time, bandwidth, memory, virtual machine, and so on. Many applications need to be processed and execute parallelly in cloud computing and fog computing effectively. Therefore, the data center needs

to properly utilize a cloud node and fog node while keeping up the degree of responsiveness for the jobs that are executed parallelly [26].

The internet holds an immense volume of data while the quantity of internet users is getting growing. Due to this case, it is problematic to assign various tasks to the VM manually [27]. Hence, scheduling is necessary to allocate the tasks to the available resource and vice versa effectively. Recently, many algorithms for scheduling are anticipated that can improve performance, cost, load balancing, and so on. Generally, scheduling algorithms have a great impact on the quality of services that are provided by cloud computing and fog computing. But all proposed scheduling algorithms have not the same qualities. To schedule tasks properly, the algorithms must consider different parameters.

2.1.2 Types of Scheduling

There are various types of TS algorithm for CC and fog computing that depend on different parameters and standards. These are listed below: -

- **Static Scheduling:** this type of schedule is worked by pre-Scheduling jobs. All information about the maintainable resources and tasks are well known.
- **Dynamic Scheduling:** this type of algorithm is the exact opposite of static scheduling.
- In this schedule, Jobs dynamically exist for scheduling overtime via the scheduler. It is more elastic than static scheduling.
- **Centralized Scheduling:** in this scheduling decisions are made centrally. Efficiency and ease of resource monitoring are ensured via this algorithm.
- **Distributed Scheduling:** it is the inverse of centralized scheduling and is applied in a real cloud environment. It guarantees scalability and also tolerates fault but it lacks efficiency.
- **Preemptive Scheduling:** This type allows all jobs to interject during implementation and a job can be traveled to another resource sendoff its originally owed resource, available for other jobs.
- **Non-Preemptive Scheduling:** Scheduling process, in which resources are not being allowed to be re-allocated until the consecutively and the scheduled job ended its execution.

- **Co-operative scheduling:** Here, the system has already several schedulers, all of them are responsible for the execution of certain activities in the scheduling process to the common system-wide range based on the cooperation of events, given rubrics, and present scheme users.
- **Immediate/Online Mode:** Here, the scheduler schedules any freshly arriving job as soon as it reaches with no waiting for next time intermission on available resources at that instant.
- **Batch/Offline Mode:** The scheduler supplies inward jobs as a collection of problems to solve over consecutive time intervals so that it is well to map a process for fit resources depending on its features.

2.1.3 Phase of Scheduling

CC has three stages of the scheduling procedure. These stages are described below.

- 1) **Filtering and discovering resources:** - In the CC scheduling process, it is the opening phase. The data center broker searches the resources available in the surrounding network in the process. The broker is responsible for gathering the status of related data of the available resource.
- 2) **Resource selection:** - after discovering resources the coming step will be resource selection. It is the method of choosing or selecting a target resource by considering certain parameters of the task and resource.
- 3) **Task submissions:** - Task is submitted and mapped to the resource is chosen or selected [28].

2.1.4 Existing Task Scheduling Algorithms

Significantly, there are three classifications of scheduling in distributed computing specifically in CC just as in FC. These classifications are heuristic, MH, and hybrid scheduling schemes. [Figure 2.1](#) present the comprehensive organization of the scheduling scheme.

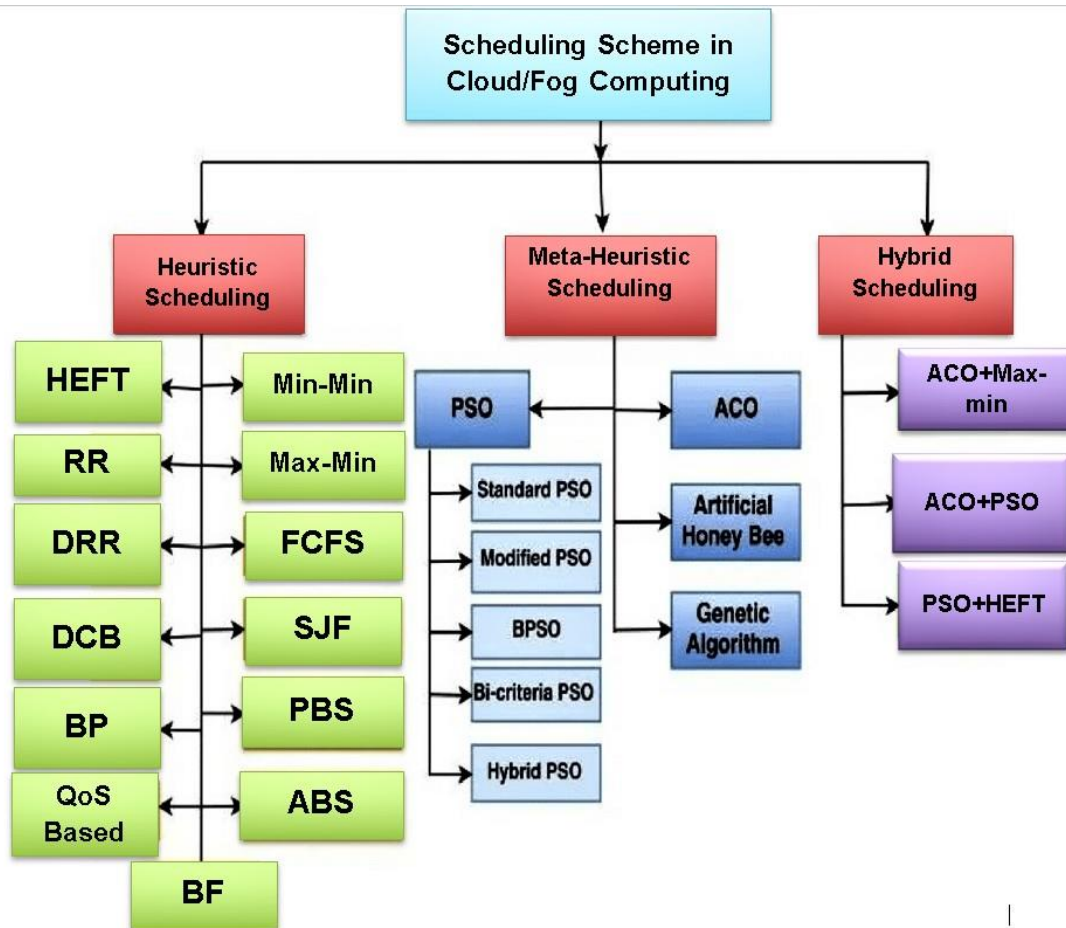


Figure 2-1 Classification of scheduling scheme in the cloud and fog computing [29]

2.1.4.1 Heuristic Task Scheduling Approaches

Heuristic scheduling is one kind of algorithm for scheduling that is broadly utilized in disseminated computing environments like CC and FC. These sorts of the algorithm are problem subordinate and give better performance results to a particular area of issues. Other than this reality, these kinds of scheduling algorithms have certain disadvantages. The presence of low execution for the complex types of problems is the problem of these types of scheduling algorithms. This scheduling algorithm can give a definite answer to certain time problems but can't take care of hard optimization issues. Distinctive heuristic calculations have been

proposed for distributed computing that tackles the scheduling problems, for example, HEFT, min-min, FCFS, RR, SJF, max-min, and so forth.

- **HEFT**: -it is a heuristic scheduling algorithm to schedule a lot of related jobs or tasks onto a system of heterogeneous resources taking correspondence time into account [30]. H. Topcuoglu et al. [31], proposed the HEFT algorithm for workflow scheduling that computes the average processing time of up-coming tasks and the average time of communication between the VM of two progressive tasks. Extra tasks are arranged in diminishing order and tasks or jobs with higher position value are given a higher need or priority. Tasks are arranged dependent on their priority or need and each task is distributed to the virtual machine that can finish the task at the earliest time.
- **FCFS** [32]: - It is one of the best and simplest Scheduling algorithms. The most fundamental and simplest technique uses the task arrival time to schedule the task on the cloud environment but the algorithm failed to balance the workload among the VMs. The task will be scheduled and executed depending on which task has arrived first in the queue.it depends on arrival time and doesn't consider any other parameter. The tasks will be scheduled by selecting the correct job order. The task or user request which comes first to the data center will be assigned to VM first for execution. The data center controller checks for a free Virtual machine and then assign a task to that VM then remove that task from the queue.
- **SJF** [33] is an algorithm of scheduling that chooses the jobs or tasks with a little execution time. This Scheduling method chooses the shortest task first for execution at available resources to minimize the time and different QoS parameters, however, the method neglected to improve parameters of the QoS because of unfair load distribution at certain nodes.
- **RR** [34] is the least complicated and comprehensively utilized task scheduling algorithm. A little unit of time, known as time slices is defined. Every runnable process is held around the queue. A RR task scheduling algorithm was once proposed to separate scheduling time among every single task equally, in which all tasks get in the queue list, and every task receives an in a similar way little unit of time.
- **Deadline-based scheduling**: Deadline constraint has to turn out to be a fundamental aspect in the subject of scheduling, it impacts the SLA if a massive vary of features unable to encounter the deadline. E. Coninck. [35], proposed a deadline-based scheduling

algorithm that dynamically provisions the available resources for scalability as per demand and improves the QoS parameters such as task meet with dead-line, time, etc.

- **MM Algorithm-** This scheduling approach works on a strategy in which a task that has minimal execution time is chosen for all tasks. This algorithm begins when a set of whole tasks/jobs are no longer assigned and proceed to execute until the whole set of the job is empty [36]. In MM jobs having a higher time or lengthy task can also now no longer be considered first and the task having larger time will usually follow the speedy job.
- **PSA [37]** - This algorithm is preemptive in which all matters are primarily based on priority. Each process in the system is based totally on the priority whereas the highest precedence job can run first whereas lower priority jobs can be made to wait. The greatest problem of this algorithm is the hunger of a process.
- **MxM Algorithm:** MxM algorithm maps every job to the VM known as sources such that every job can be processed within the minimum possible time. It first calculates the different job execution times on each accessible resource. Then the jobs or tasks that need regular most completion time are selected. This job is then assigned to the resources. This approach is repeated until the whole tasks are mapped to the resources. This algorithm allows large jobs have to wait less time [38].
- **MET:** assigns jobs to VMs or a given resource based on the agreeable predictable time completion for that task besides regard to resource availability. The core thinking of MET is to assign a task to VM or resources primarily based on minimum execution time, which every so often consequences in immoderate load imbalance since the assignment is independent of the availability [39].
- **MCT:** this scheduling method maps jobs to VMs or certain resources dependent on the optimal and less time of completion for a certain task in the arbitrary order. Each job is appointed or mapped to the VM or resources that have the most punctual finish time. With the MCT method of scheduling, a few tasks are mapped to the VMs or resources having no minimum execution time [40].

2.1.4.2 Metaheuristic Task Scheduling Approaches

Thanks to their performance in solving major computational and confusing problems, the MH scheduling algorithm has gained considerable popularity over the most recent twenty years. A metaheuristic scheduling algorithm has some useful features: -

- This scheduling technique is not problem-dependent.
- The MH scheduling methodology examines correctly the search space to discover the most efficient options or incomplete association of NP-Complete problems.
- Approximate and for the most part non-deterministic estimates.

In general, a meta-heuristic scheduling algorithm is not problem-dependent and excellent to handle issues with useful execution from exceptional areas. One of the ordinary methodologies for solving problems of NP-hard optimization is meta-heuristic techniques. Specific meta-heuristic scheduling occurred in a cloud region to occur throughout the difficult (problematic) relationship of the NP-Complete problem in a short time frame, such as PSO, ACO, ABC, GA, CSO, etc.

PSO is an instance of a population-based MH scheduling approach primarily developed from swarm experience by [41] in 1995. PSO is a method of nature-propelled development, suggested after the cultural habits of flocks of flying creatures, for example, how they locate and make use of multi-dimensional food and shelter search [42]. The PSO scheduling algorithm is similarly categorized into a variety of classes like standard PSO, bi-objective PSO, modified PSO, hybrid PSO [43, 44], which has been utilized in the environment of the cloud to schedule the upcoming applications or sensible workflow at a VM.

The first category is Standard PSO. Live VMs migration in a DE, such as CC and fog computing, is expensive and is a time-consuming process. To minimize these limitations, in [45], the author suggested a task-based PSO or work scheduling algorithm that would move tasks from overloaded VMs to underloaded VMs instead of transferring all overloaded VMs. In addition, an innovative optimization architecture was developed to move these new tasks to VM and increase the makespan time as nicely as the transfer time for the tasks.

Bi-objective PSO is the second class of PSO scheduling algorithms. It is a variant of the PSO scheduling technique which simultaneously optimizes two-goal functions in the CC. The PBBO PSO algorithm [46] that optimizes each of the parameters at the same time was proposed to enhance the execution efficiency by considering deadline and also price range constraints.

MPSO is another type of PSO scheduling algorithm for optimization. There are many PSO related algorithms viewed as modified PSO to avoid the problem and to improve the existing PSO algorithm. In the past, several PSO algorithms were developed and modeled alongside changes. the developed algorithm has the power to advance the PSO performance and maximize the range of CC QoS parameters and that of FC. A new adapted PSO-based scheduling algorithm was introduced via Tarek et al. [47] called MPSO to minimize the scheduling problem of the cloud. The updated algorithm's main intention is to scan the enormous resource for the upcoming task and assign the task to resource-based contemporary workload at VMs to reduce completion time and cost.

To overcome the problems of current PSO strategies, PSO integrates one or more scheduling methods. The resulting algorithm created by combining those algorithms is called hybrid PSO algorithms. Zhan et al. [48] proposed a useful resource scheduling algorithm that mixes SA and PSO known as IPSO to improve the CC time-and resource utilization ratio. At some point in this report, Krishnasamy and Gomathi et al. [49] to optimize the workload and diminish the makespan time of the CC programs discussed other Hybrid PSO (PSO with DE) algorithms.

ACO is an optimization algorithm that is used in CC to bargain the best result of computational problems [50]. It is induced by the seeking activity of real ants seeking the straight route between their colonies and a supply of food assisted the pheromone concentration of volatile chemical substances. To advance the time and workload reliability within the cloud cluster, a sensible set of rules for optimization called the cycle ACO (PACO) was projected by Sun W. [51].

E. Pacini et al. [52] have suggested a two-degree Cloud scheduler that uses the ACO algorithm to perform the scientific experiments and increase the various QoS parameters (response time, makespan time, and throughput) within the CC. Additionally, a set of rules based on stepped forward ACO has been developed with the aid of H. Duna et al. [29] to address the complexity of peak loads in cloud computing. This improved algorithm is called the program of Pre-Ant policy.

This algorithm consists of a predictive model that determines the first-class mapping of virtual machine tasks or jobs and improves numerous QoS parameters. ACO set of rules was explored alongside its variations using A.R. Arunarani et al. [53] discussing the scheduling problem.

ABC is stimulated by the honey bee swarm's intelligent foraging behavior. This scheduling approach is proposed to optimize many issues with the assistance of Karaboga in 2005 [54]. Artificial bees hover everywhere in different directions at ABC to locate the approximate solution to the problems of optimization. Employed bees, scouts, and onlookers are the types of bees. Scouts monitor the exploration and randomly select new food properties without having to use experience. Employed bee tells Onlooker bee about the supply of food after Onlooker bee measures the fitness value and selects the great source of food that is collected by Employed bee.

A GA is an algorithm of optimization, where it tries to imitate the way evolution takes place. John Holland developed GA in 1960, it's miles primarily based on the principle of "Survival of the Fittest" and has been tested with a reasonable rate of success on many optimization problems. At GA, we have a set of possible solutions to the given problem. The solutions undergo re-combinations to provide new offspring, such as crossover (similar to natural genetics). Member of the community is equipped with a fitness value, and the threat of mating for fitter human beings is held excessive to produce more "fitter" offspring. The process passes iteratively to multiple generations until an appropriate response is reached. The method goes over to numerous generations iteratively, until an acceptable answer is reached. GA is proving to be a tool capable of solving NP-hard issues in information technology or in computer science that takes a very long time [55].

Likewise, DEA is an optimization approach originating from genetics such as GA. Storn and Price added it for the very first time in 1995. DEA has outperformed other algorithms in solving various hard-distributed problems in computing, such as scheduling problems in a cloud environment. A novel, multi-objective approach to useful resource allocation was proposed via J. Tsai et al. [56], using IDEA.

SA is a global optimization technique that was developed in 1983 from an example of how the most beneficial atom association emerged within statistical mechanics and was used to solve the difficult computational hassle within the CC discipline. It is a memoryless algorithm i.e. no

information is stored during the search process. SA effectiveness derives from its postponement or extension of two main techniques, the scheme for iterative improvement, and the approach to divide and conquer.

BFO is a completely meta-heuristic style of scheduling to the optimization algorithm, which was invented in 2002 and will not find the approximate answer to a huge and sophisticated CC problem. Two algorithms of load balancing choices (online and offline) were proposed by way of L. Tang et coll. [57]. The primary choice (online) deals with real-time load balancing packages by providing small sources and the second choice (offline) is used for global scheduling using the BFO algorithm that gives the entire system stability and enhances the machine's overall performance.

In 2010 Xin-She Yang proposed a new algorithm to find the approximate solution to the NP-hard optimization problem (task scheduling, traveling salesman problem, graph coloring problem, etc.). This new algorithm is known to be the BAT optimization algorithm, which was inspired by Bats' echolocation behavior [58].

CSA [59] is a metaheuristic method in which individuals mimic intelligent behavior within a crew group. The published results demonstrate the ability to solve several complex issues in improving the distributed computing environment's optimization problems.

2.1.4.3 Hybrid Scheduling Algorithm

In this segment, to solve the cloud resource-scheduling problem, we need to address hybrid scheduling algorithms that combine two or more scheduling. R. Kaur & N. Ghumman combined the ant colony's MH algorithm with a heuristic max-min algorithm to improve overall processing time and cost QoS parameters, taking task priorities as a constraint [60].

S.Yassa has proposed PSO with HEFT for the preparation of multi-objective workflows. Yassa et al. [61], along with energy consumption, simultaneously raise different QoS parameters like makespan time and cost. The proposed algorithm provides a set of solutions for the time and cost of competing parameters which provide users with flexibility in their preferences. M.Elaziz. citeelaziz2019task is another researcher, suggested an algorithm for MSDE to minimize the period of the applications.

2.1.5 Parameters for Comparing Algorithms of Scheduling

In this segment, we analyze different parameters applied to schedule the tasks of different users of the cloud scenarios. Many of those requirements are strengthened in the works reviewed. [Table 2.1](#) outlines a list of parameters used to equate the scheduling to various algorithms. Any scheduling approach aims to diminish the makespan, execution time, waiting time, energy consumption, expense, and optimize processor and resource usage, throughput, and QoS.

Table 2.1 Parameters for Comparing Scheduling Algorithm [21]

No	Parameter	Description
1	Makespan or completion time	Total time taken by a task for its complete execution
2	Execution time	Processing time taken to execute the given task
3	Processing cost	The cost incurred for using the Resources
4	Throughput	Number of tasks completed per unit Time
5	Resource utilization	Resources utilized by different tasks.
6	Load balancing	Equally distribution of the workload
7	Waiting time	Amount of time a task is in queue
8	Deadline	Maximum time by which it must be completed.
	Priority	The order in which task is to be processed
10	Energy consumption	Amount of power used for doing a task
11	QoS	The overall achievement of the cloud System

2.2 Related Works

Scheduling of tasks is a challenging task in CC due to dispersion, resource volatility, and on-demand demands from application users. In the environment of the cloud, several algorithms were proposed for task scheduling. The algorithm has its advantages and disadvantages since each algorithm operates with specific QoS parameters such as execution time, makespan time,

execution cost, energy consumption, efficiency, etc., and optimizes one or more parameters based on the specified fitness function.

The authors in [62] suggest a TS algorithm by using the MH scheduling algorithms namely MPSO. The proposed model has two sections. Allocating tasks to VMs and allocating / managing resources in the cloud.

The authors in [63] proposed a novel TS algorithm by using the MH scheduling algorithms called MPSO. The proposed model is used to schedules the task among fog devices.

In [29] Mohit Kumar and S.C. Sharma prepare a comprehensive survey based upon heuristic, MH, and hybrid TS scheduling algorithm for CC.

S. Ali, and B. H. Malik [36] suggest a TS algorithm using FCFS for CC. The paper shows that the FCFS TS algorithm is used in the CC environment even if it has certain limitations. The algorithm is well suited for scheduling a small number of incoming tasks.

The study in [64] suggests a TS algorithm in a CC that manages the task to improve the cost and to minimize the energy consumption of the CC system.

Shridhar G. Domanal and G Ram Mohana Reddy [50, 51] have proposed an updated throttled algorithm that effectively schedules incoming customer tasks on the virtual machines. The author's attention has been on the task response time.

M Kumar and SC Sharma [65] suggest a resource scheduling algorithm in a cloud environment that manages the workload to maximize the use of cloud resources and the forthcoming task's makespan time.

Babu and Krishna have projected an innovative algorithm for scheduling for the environment of the cloud. The algorithm reduces the RT and makespan time by considering the task's priority as the QoS parameter [54].

Table 2.2 Summary of related work finding

<i>Approac</i>	<i>Algorith</i>	<i>Improved criteria</i>	<i>Limitations</i>
[62]	MPSO	Reliability and cloud resource utilization	Use a homogeneous task, only in CC, VM takes the long waiting time if found no match
[63]	MPSO	Energy consumption, Execution cost, and cloud resource utilization	Small dataset, homogeneous task, only in homogeneous environment/FC
[29, 36]	FCFS	First arrival request of the task was fulfilled, simplest method and lower headache in scheduling	Uneven utilization of the VM, long waiting time, long execution time, only in CC, Small dataset
[29, 64]	PSO-COGENT	Improve cost, energy consumption	long execution time, sometimes VMs are overloaded due to improper mapping of tasks with VMs, only in CC

As shown in Table 2.2 above, none of the papers simultaneously considered the parameters such as execution time or response time, energy consumption, resource utilization, and execution cost. Furthermore, numerous forms of algorithms for scheduling (CC and FC) are specifically proposed. But this paper considered some of those QoS parameters and used the proposed algorithm to optimize the parameters.

CHAPTER 3

3 RESEARCH METHODOLOGY

3.1 Overall Methodology

This part contains the thesis methodology as stated in the article. In this section, the author discusses in more detail the research strategy, method, approach, the data collection techniques, the research measures or metrics, the research procedure or technique, the simulation system, the method of testing and evaluation, the algorithm used.

3.2 Research Strategy

The research conducted in support of this thesis was applied types but not the new ones. Instead, there are several pieces of earlier academic research concerning task scheduling in cloud computing and fog computing. This research is carried out by reviewing several relevant literatures works in cloud and fog computing environments within the domain of task scheduling. To accurately collect the relevant resource, the author has analyzed written articles, research papers, a forum on the identified problem, and also visited the official technology sites to look around specifications, methodology, technology stack, conventions, and archetype. Additionally, some significant papers printed by specialists and authors are reviewed. In conduct this work, the following brief steps apply to the research strategy.

- Defined the identified problem clearly to be resolved.
- Requirements for the design of the proposed solution have been established.
- Collect the relevant resource by reading written articles, research papers, a dedicated forum on the problem found, visiting official technology sites to look at requirements, methods, application stacks, conventions, and models.
- Implement the proposed solution
- Prepare to conduct the experiment using various mechanisms that suit the problem identified.
- Document the results the analysis obtains

3.3 Research Method

Quantitative research was held to fulfill the goals of the thesis. The main characteristic of this kind of research is, it is best suited for both the small dataset and the huge dataset, whereas its results are measurable and quantifiable. Its straightforward benefit, which also establishes its fundamental difference with qualitative research, is that quantitative data is more efficient, capable of testing hypotheses, by restricting the research scope and the feature of the responses of the participants, i.e., not subjective (interpretation of events by individuals may not be important) [73].

Generally speaking, the reliability of quantitative research is based on the researchers' skills and abilities and also on the value of the data used, while the findings shall view as accurate because the results do not originate from the personal opinions and perceptions of the researcher. Because small samples, also a huge dataset, are more appropriate, it is not risky for qualitative research results to be considered reliable [74].

3.4 Research Approach

The investigative approach followed for this research was the deductive one. According to this approach, researchers start exploring an identified phenomenon and testing whether that theory is valid under certain circumstances. The reason for the deductive approach was that it takes into account the context in which the research effort is active, while it is also most suitable for small as well as large samples that produce qualitative data or results (the reliability of the research results cannot be questioned) [75].

3.5 Data Collection Method and Tools

In order to develop the proposed approach, input data was taken from another source, and also the data set has been prepared automatically using Cloudsim. The data set and data, in this case, is a group of tasks that will be scheduled on the available fog node or cloud node, and the implementations of other algorithms that are going to be compared respectively.

3.6 Research Process

The proposed solution is applied and developed using a hybrid style to the computing environment known as a cloud-fog computing system. It solves the centralized architecture problems of the cloud. Besides this, the algorithm that is developed in the proposed study uses a hybrid scheduling method for task scheduling. By combining the FCFS algorithm and the MPSO algorithm the task scheduling algorithm for the cloud-fog environment is developed. This scheduling algorithm combination or hybrid solves the limitations or problems of those algorithms and allows us to develop a good algorithm for the cloud-fog computing system.

The hybrid algorithm functions as follows. The scheduler will find the best fog node match for the incoming task, depending on its CPU time and memory demands and allocating it. If the fog device does not find any match then it sends the task to the cloud for allocation. Task allocation will take place in the first round using the FCFS algorithm, and the MPSO algorithm will be invoked.

3.7 Simulation Environment

There are many simulation toolkits for cloud and fog environment, but the proposed task-scheduling algorithm is simulated by using the Cloudsim simulator toolkit. Because this toolkit is more appropriate with our selected requirements. Also, the reason behind choosing Cloudsim is that it allows cloud providers to test their services in a repeatable and controlled environment free of cost and to tune the performance bottlenecks before deploying on real-world Clouds. On the simulation side, the simulation environment permits different kinds of resource allocation under different load distributions. Cloudsim provides a good simulation framework for emerging cloud computing applications so that researchers can investigate different design issues without getting concerned about low-level details. [34, 35].

3.8 Research Metrics

This research will analyze the advanced techniques, and algorithms in order to design and evaluate the proposed algorithm in terms of selected metrics. The criteria to evaluate the proposed approach are-

- VM utilization
- Average response time.
- Throughput and
- Average execution time.

3.9 Testing and Methods of Evaluation

The proposed work or approach is tested and evaluated by using a given evaluation metric as explained above. Moreover, the projected approach is equated with other peer algorithms, and also the original algorithm to analyze whether the defined objective of the research is succeeded or failed. Finally, the result is documented using the graph and other methods and the final document is prepared like this.

3.10 Development Tools and Techniques

For implementation and testing purposes, different types of tools are applied to implement the logic into a solution. The list is shown as follows.

3.10.1 Implementation Techniques and Tools

- **Eclipse** [76] is an interactive, computer programming development environment. Multiple language and other functionality can easily be merged into any of the default packages with eclipse. The Eclipse Marketplace enables virtually unlimited flexibility and extension.
- **Java programming language:** Cloudsim framework is designed to work with JDK.

3.10.2 Designing Tools

- Edraw Max tool [77] is a kind of design tool that allows the software designer to reliably create and publish different diagrams. This design tool provides many features such as flowchart, workflows, software, UML diagrams, business diagrams, and database.

CHAPTER 4

4 DESIGNING PROPOSED SOLUTION

4.1 Proposed Solution Architecture

The architectural solution for dynamic task scheduling using a hybrid scheduling algorithm, i.e., MPSO and FCFS, is discussed in this chapter. In this thesis, a cloud-fog system is assumed which consists of fog and cloud processing nodes. Our system architecture is comprised of four layers i.e. user module, fog devices, scheduler, and cloud datacenter. The working of our proposed HHMH architecture is that the broker will receive all the client requests through fog devices. The scheduler is used for evaluating, calculating, and arranging all activities in the Cloud-Fog system to be performed. The scheduler also searches for the best resource match for jobs based on the resource demand of the tasks. It schedules the task using FCFS in the first round and then the MPSO algorithm will be invoked. [Figure 4.1](#) shows the model of the proposed system. The scheduler is close to the nodes, so the time spent is negligible for data exchange between each other. [Figure 4.2](#) defined the operation of the system model.

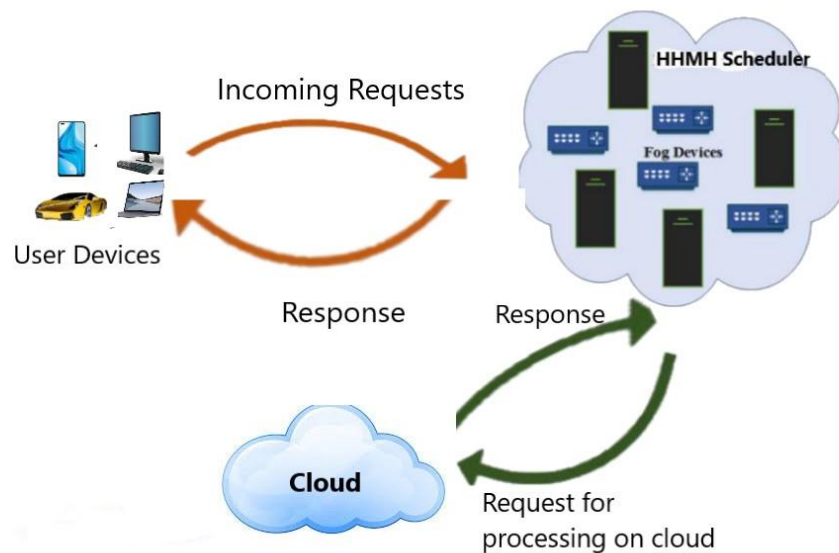


Figure 4-1 HHMH system model

- 1) The users send a request, which is handled by the Fog node it is connected to.
- 2) This request is immediately forwarded to the scheduler.
- 3) The submitted task resource usage needed is estimated using the scheduler.
- 4) Handling all information about tasks and nodes, the scheduling algorithm (HHMH) find a good task assignment.
- 5) Following the output, tasks are forward to corresponding Cloud nodes and Fog nodes.
- 6) Each node has responsibility for processing all tasks assigned.
- 7) Each node has responsibility for sending the results back to the scheduler.
- 8) Once all tasks are completed, the scheduler combines the result of the job.
- 9) Subsequently, the reply is directed to the user through the Fog node that the user connected with.

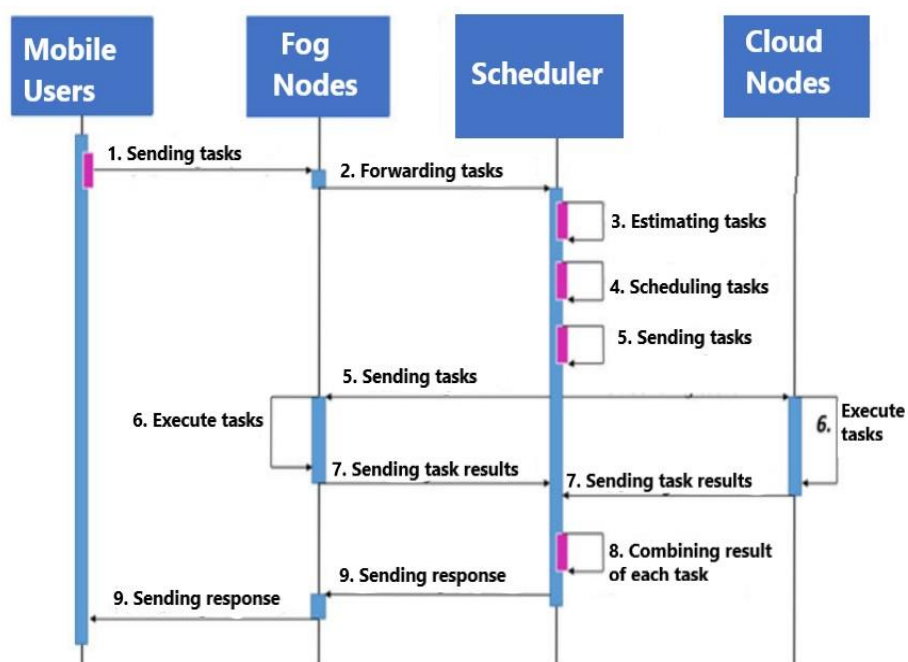


Figure 4-2 Sequence diagram of cloud-fog architecture [6]

4.2 Problem Formulation

The independent tasks to be processed over the Cloud-Fog computing system as requests from users are submitted to the Fog layer. The following properties are available for each task: the number of instructions, the memory needed, input size, and output files. Assuming that T_k indicates the k^{th} task, a collection of n independent tasks is sent to the fog layer at each moment, as follows [6]:

$$T = \{T1, T2, T3, \dots, Tn\} \quad (1)$$

The Cloud-Fog computing infrastructure consists of processors that have the same characteristics, such as CPU rate, CPU usage fee, memory usage fee, and bandwidth usage fee. Cloud nodes, however, are usually better than the Fog nodes, but the cost is higher while using them. The set of processors including cloud node and fog node in the device is expressed as [6]:

$$N = \{N1, N2, N3, \dots, \} \quad (2)$$

where N_i presents the i^{th} processing node.

Average response time is the total time taken for responding to a task. It is computed by the formula:

$$ART = t_2 \left[\sum_{x=1}^n VM(i) \right] - t_1 \left[\sum_{x=1}^n VM(i) \right] \quad (3)$$

Where t_1 presents the time at which the task enters the VM and t_2 is the time at which the task completes the execution. In the proposed algorithm, GB does not remain the same in each iteration when compared to PSO. Node or VM position is modified using the equation:

$$x(t + 1) = x(t) + v(t) \quad (4)$$

Where $x(t)$ is the current position of particle / Current load of a VM, $v(t)$ is the velocity of the node. Particles are evaluated through a fitness function, which represents the quality of the

solution that the particles expresses and shows its influence on the population. The value of fitness is computed by the utility function F . If there is a higher value of fitness, the better is the solution achieved. In the proposed work particle, refer to VMs.

$$F = \frac{(R) \sum_{i=0}^k VM(i)}{(RD) \sum_{x=1}^n T(x)} \quad (5)$$

Where R is the resource, RD is the resource demands by tasks and T is the task. The throughput of a given system is evaluated using Equation 6.

$$Throughput = \frac{\text{Number of tasks accomplished}}{\text{Total processing time}} \quad (6)$$

4.3 Scheduling of Tasks on VM Using Proposed (HHMH) Algorithm

In this section, the algorithms used in the proposed approach are clarified briefly. It is stated that in cloud computing, several researchers have used PSO to solve the scheduling problem. Fog computing is the derivative of cloud computing in which we can also use the best-performing methods of metaheuristic. Here, the proposed method is used for scheduling the VMs against incoming tasks.

Number of arriving requests i.e., x_1, x_2, x_3, x_n , have to be scheduled on VMs i.e. vm_0, vm_1, vm_2, vm_k . In allocating the external request of tasks to the VMs effectively, the HHMH algorithm provides an important role. The number of VMs taken for experimentation is the base for clustering. [Algorithm 4.1](#) contains full details of HHMH's scheduling of tasks.

The least loaded VM or personal best (PB) and the smallest among PB also called global best (GB) will be found by each cluster in each iteration. The VM related to GB is assigned to the next task. In the subsequent iteration, if the GB stays the same, then the GB is modified from the Cz cluster list with the second least PB. Until all the tasks are executed, the same process will proceed. The task is sent to the cloud for processing if the match is not identified. [Figure 4.3](#) shows the flow diagram of the suggested or proposed algorithm. [Algorithm 4.1](#) is the proposed HHMH algorithm that is used for task scheduling in cloud-fog environments. The steps for the proposed approach are listed below. Note that PB and LB have the same meaning in this paper.

- 1) With position and velocity, initiate each VM.
- 2) Allocate tasks to each VM of fog using the FCFS algorithm in the first round.
- 3) Test each VM's fitness value using the fitness function
- 4) Compare particle with the largest fitness value calculated in the above step, initiate its position and update value, and compare this particle with the smallest (optimal) fitness value and check whether its new position is suitable, if yes, change and update its personal position (PB), otherwise, assign a new position to this particle randomly in its surroundings with radius r and then update the position and velocity of another particle according to the fitness function
- 5) If the latest fitness value is good, change its fitness value and personal best (LB) to the new best, then compare the current fitness value of each node with its personal best (LB). Then find the best node/VM with the best fitness in the population and compare the current fitness value and the best global fitness value (GB). If yes, with the present location, renew its fitness value and global best (GB).
- 6) Assign the next task to the VM of fog which contains GB.
- 7) If the next allocation of VM is equal to last used GB, return step 3 for next least LB. otherwise return step 6 for next allocation of task.
- 8) Check the set criteria (fitness function) to find the desired solution, if it has been achieved, end the iteration of the algorithm.
- 9) Assign unallocated tasks to the cloud. Note that the above steps are for fog and the steps are also the same in the cloud.

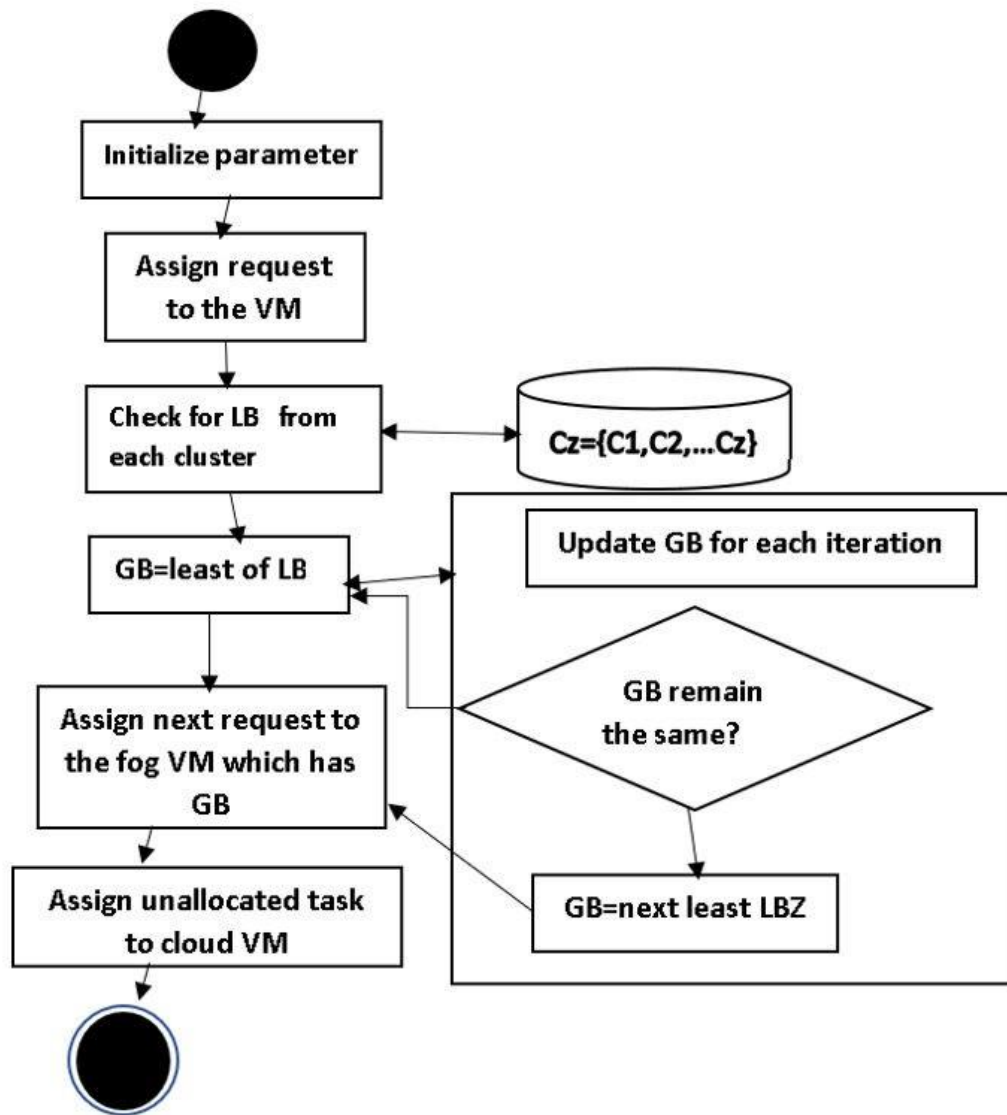


Figure 4-3 Flow diagram of the developed algorithm

Algorithm 4.1 Pseudocode of the Proposed Algorithm

```
1. initialization: count=0, LB=0, GB=0, VMs=VM0,VM1,VM2...,VMk
   Clusters Cz=C1,C2,C3...,Cz Cluster-size=K/Cz
2. for incoming requests x1, x2, x3...xn do
3.   allocate task to each VM of fog
4.   if all VMallocated then
5.     goto step 8
6.   end
7. end
8. for unallocated requests x1, x2, x3...xn do
9.   for Cz = c1, C2, C3, ...Cz do
10.    Each cluster Cz=least loaded VM
11.    Assign each one of them as LB from Cz
12.  end
13. Assign GB= least LB
14. Next task allocated to VM which contains GB
15. if nextallocation==lastusedGB then
16.   goto step 10 for next least LB
17. else
18.   goto step 14
19. end
20. end
21. for unallocated requests x1, x2, x3...xn do
22.   allocate task to each VM of cloud
23.   if all VMallocated then
24.     goto step 27
25.   end
26. end
27. for unallocated requests x1, x2, x3...xn do
28.   for Cz = c1, C2, C3, ...Cz do
29.    Each cluster Cz=least loaded VM
30.    Assign each one of them as LB from Cz
31.  end
32. Assign GB= least LB
33. Next task allocated to VM which contains GB
34. if nextallocation==lastusedGB then
35.   goto step 29 for next least LB
36. else
37.   goto step 33
38. end
39. end
```

Algorithm 4.2 Task Scheduling Using MPSO [62]

1. initialization: count=0, LB=0, GB=0, VMs=VM0,VM1,VM2...,VMk
Clusters $Cz=C_1, C_2, C_3, \dots, C_z$
2. for all incoming requests $x_1, x_2, x_3, \dots, x_n$ do
3. | Each cluster Cz =least loaded VM
4. | Assign each one of them as LB from Cz
5. end
6. Assign GB= least LB
7. Next task allocated to VM which contains GB
8. if nextallocation==lastusedGB then
9. | goto step 3 for next least LB
10. else
11. | goto step 7
12. end

CHAPTER 5

5 EXPERIMENTAL SETUP AND IMPLEMENTATION

5.1 Experimental Setup

There are distinct computing power and resource consumption costs for Cloud and FN. In addition to CPU, memory, and transmission power usage costs, we believe that each node or computer had its processing limit estimated by MIPS. The Cloud-Fog system was designed with ten processing nodes, with the characteristics shown in [Table 5.1](#). Fog nodes have minimal processing power in the fog layer, such as switches, workstations, or PCs, Servers or VM at elite server farms are responsible for taking care of requests when in the cloud layer. This way, the processing rate of the cloud node is faster than that of the fog node. Interestingly the cost of using services in cloud computing is more expensive than in the fog. In the simulation, G\$ is a currency unit used to replace real money.

Table 5.1 Characteristics of the Cloud-Fog infrastructures

<i>Parameter</i>	<i>Fog Environment</i>	<i>Cloud Environment</i>	<i>Unit</i>
Number of nodes	7	3	node
CPU rate	(500,1500)	(3000,5000)	MIPS
CPU usage cost	(0.1,0.4)	(0.7,1.0)	G\$/s
Memory usage cost	(0.01,0.03)	(0.02,0.05)	G\$/MB
Bandwidth usage cost	(0.01,0.02)	(0.05,1.0)	G\$/MB

It is the responsibility of the Cloud-Fog system to execute all requests from users. Each task was presumed to have certain attributes like the number of instructions, memory available, size of the input file, and size of the output file. The collection of tasks can differ widely in size, depending on the workload of each request. Thus, to participate in our simulation, 11 datasets with from 40 to 500 tasks were developed. Following Table 5.2, each task in a dataset was generated randomly with its attributes. The experiment could cover different scenarios of randomness because several types of tasks were developed, some of which required a large amount of computation, while others needed more use of memory or bandwidth, and so on.

Table 5.4 shows 40 cloudlets or tasks that are created randomly having different values of attributes.

Table 5.2 Attributes of tasks

<i>Property</i>	<i>Value</i>	<i>Unit</i>
Number of instructions	(1,100)	MIPS
Memory required	(50,200)	MB
Input file size	(10,100)	MB
Output file size	(10,100)	MB

Table 5.3 shows the settings for the experimental environment, and the simulation has been generated with the Java Eclipse editor and Cloudsim. It is (Cloudsim) a cloud-computing framework widely used over the years and tested in various studies.

Table 5.3 Simulation setup

<i>System</i>	<i>Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz</i>
Memory	8GB
Simulator	CloudSim
Operating system	Windows 10 Professional

Table 5.4 Sample data set of 40 cloudlets

<i>Cloudlet ID</i>	<i>Cloudlet Length</i>	<i>Number of pes</i>	<i>Cloudlet File size</i>	<i>Cloudlet Output size</i>
00	164	40	96	46
01	70	56	49	84
02	176	26	75	27
03	80	55	48	71
04	131	36	49	85
05	187	07	76	54
06	171	21	67	46
07	184	75	52	14
08	153	88	40	69
09	160	72	93	55
10	119	69	82	81
11	93	72	87	45
12	164	44	86	95
13	189	12	81	21
14	90	39	10	58
15	50	90	78	10

16	127	89	48	80
17	141	68	69	54
18	179	45	88	87
19	142	26	46	42
20	168	22	93	19
21	84	68	20	89
22	158	51	72	92
23	128	77	28	14
24	60	33	82	75
25	63	44	23	20
26	121	60	73	40
27	104	24	95	26
28	161	47	87	89
29	188	61	27	20
30	117	42	48	41
31	118	71	98	49
32	115	53	79	72
33	134	30	46	75
34	196	34	19	75
35	136	80	15	65
36	142	76	97	83
37	134	36	88	44
38	127	73	85	94
39	152	63	38	95

5.2 Implementation

Steps for creating and running a program using Cloudsim are listed below. This shows steps for creating and running the simulated infrastructure of the cloud using Cloudsim.

1. Prepare the package of Cloudsim. It should be invoked before any entities are formed.
2. Create Datacenters.
3. Create Broker
4. Create a VM
5. Create Cloudlet
6. Beginning the simulation
7. Stop the simulation
8. Print results when the test is over

5.2.1 Common Simulation Configuration Activities

The implementation of the PS is followed by some common activities. These activities are described below.

- I. **DC Configuration:** -this class is a Cloud resource whose hosts are virtualized. For our simulation, we have used five datacenters. Configure DC parameters DC (hostlist, arch, os, vmm, cost mem/bw/storage)
- II. **Configure host parameters:** -The Physical Machine inside a DC is defined in this class. It is used for the execution of activities relevant to VM management such as construction and destruction. There are resources inside the host to virtualize including MIPS running on VMs, storage, and bandwidth. Host attributes include MIPS, storage, RAM, BW.
- III. **Configure VM with its parameters:** - VM is a machine that is run inside the host. VM is sharing host lists (Host resources) with other VMs available in that host. Also, VMs are scaled up or scaled down based on tasks to shape themselves with the workload. MIPS, pesNumber (no. of CPU), Ram (MB), BW(MB/s).
- IV. **Broker configuration:** - brokers working based on users' requests. It hides VM management, as VM creation, submission of cloudlets (tasks) to the VMs, and destruction of VMs when loads are decreased.
- V. **Cloudlet configuration:** - Since we were not using the real cloud DC, we have used a simulation environment for simulating the proposed solution and take the cloudlets as a workload. Cloudlets has Length (MI), pesNumber, input Size, output Size that run on VMs.

5.2.2. Algorithm Architecture and Description

5.2.2.1 HHMHDatacenterBroker.java

This is a class accountable for an intermediate between user and service. All requests from mobile users are forwarded immediately to this broker, which is responsible for analyzing, estimating, and then scheduling all tasks to be executed in the Cloud-Fog system. It encapsulates VM management like VM creation and destruction.

```

1  public class HHMHDDatacenterBroker extends DatacenterBroker {
2      private double[] mapping;
3      PSODatacenterBroker(String name) throws Exception {
4          super(name);
5      }
6      public void setMapping(double[] mapping) {
7          this.mapping = mapping;
8      }
9      private List<Cloudlet> assignCloudletsToVms(List<Cloudlet> cloudlist) {
10         int idx = 0;
11         for (Cloudlet cl : cloudlist) {
12             cl.setVmId((int) mapping[idx++]);
13         }
14         return cloudlist;
15     }

```

5.2.2.2 SchedulerPaverage response timeicleUpdate.java

LB and GB allows VM's in order to update velocities regularly.

```

1. public class SchedulerPaverage response timeicleUpdate extends Paverage
   response timeicleUpdate {
2.     private static final double W = 0.9;
3.     private static final double C = 2.0;
4.     SchedulerPaverage response timeicleUpdate(Particle particle) {
5.         super(particle);
6.     }
7.     @Override
8.     public void update(Swarm swarm, Particle particle) {
9.         double[] v = particle.getVelocity();
10.        double[] x = particle.getPosition();
11.        double[] pbest = particle.getBestPosition();
12.        double[] gbest = swarm.getBestPosition();
13.        for (int i = 0; i < Constants.NO_OF_TASKS; ++i) {

```

```

14. v[i] = W * v[i] + C * Math.random() * (pbest[i] - x[i]) + C * Math.
    random() * (gbest[i] - x[i]);
15. x[i] = (int) (x[i] + v[i]);
16. } }
17. }

```

5.2.2.3 SchedulerFitnessFunction.java

VMs are evaluated through a fitness function; it represents the superiority of the result that the particle expresses and also shows its influence on the population. The higher is the fitness value, the better is the solution achieved.

```

1. public double evaluate(double[] position) {
2.     double alpha = 0.3;
3.     return alpha * calcTotalTime(position) + (1 - alpha) * calcMakespan
    (position);
4. }
5. private double calcTotalTime(double[] position) {
6.     double totalCost = 0;
7.     for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
8.         int dcId = (int) position[i];
9.         totalCost += execMatrix[i][dcId] + commMatrix[i][dcId];
10.    }
11.    return totalCost;
12. }
13. public double calcMakespan(double[] position) {
14.     double makespan = 0;
15. }
16. return makespan;
17. }
18. }

```

5.2.2.4 HHMHScheduler.java

This is a class where major portions of the algorithm/program are implemented and developed while the simulation is beginning from it. Generally, this class configures datacenters, VMs, Hosts, Broker, and Cloudlets. This class is also used for passing the configured parameter to the broker. Finally, it stops the simulation and prints the expected output.

```
1. public class HHMH_Scheduler {
2.     LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();
3.     long length1 = 50;
4.     long fileSize = 10;
5.     long outputSize = 10;
6.     int pesNumber = 1;
7.     UtilizationModel utilizationModel = new UtilizationModelFull();
8.     Cloudlet[] cloudlet = new Cloudlet[cloudlets];
9.     for (int i = 0; i < cloudlets; i++) {
10.         Random rojb = new Random();
11.         int dcId = (int) (Math.random() * Constants.NO_OF_DATA_CENTERS);
12.         long length = (long) (1e3 * (commMatrix[i][dcId] + execMatrix[i][dcId]));
13.         cloudlet[i] = new Cloudlet(idShift + i, length1 + rojb.nextInt(150),
            pesNumber, fileSize + rojb.nextInt(90), outputSize + rojb.nextInt(90), utilizationModel, utilizationModel, utilizationModel)
14.     }
```

CHAPTER 6

6 EVALUATION AND DISCUSSION

6.1 Introduction

Concerning effective VM use, average response time, throughput, and execution time, the effects of our proposed approach are evaluated. The execution time denotes the time consumed by the task using the resources of the cloud-fog. RT states the time the task becomes active before the execution is completed. In addition, the suggested algorithm is contrasted with scheduling benchmark algorithms such as SJF, FCFS. We have compared our proposed approach with the meta-heuristic method called MPSO as well.

6.1.1 Utilization of VMs

By considering VM utilization, we have tested our proposed scheduling algorithm. [Table 6.1](#) proves the extent of tasks for various algorithms performed by the VMs. [Figure 6.1](#) shows that when using various methods there is fluctuation in the use of VMs excluding the proposed algorithm since tasks are split equally on all VMs. This can be observed from [Figure 6.1](#) that when compared to other algorithms, MPSO and FCFS algorithms are not consistent in the use of VMs. Although the VMs are used effectively by SJF Algorithms, our suggested HHMH algorithm is more successful in using the VMs. For various sets of virtual machines and tasks, the experiment is replicated, and the results obtained are consistent.

Table 6.1 Number of tasks (400) handled by the VMs

<i>VMs</i>	<i>FCFS</i>	<i>SJF</i>	<i>MPSO</i>	<i>HHMH</i>
VM#1	76	85	220	83
VM#2	63	72	38	77
VM#3	91	92	47	79
VM#4	83	73	31	78
VM#5	87	78	64	83

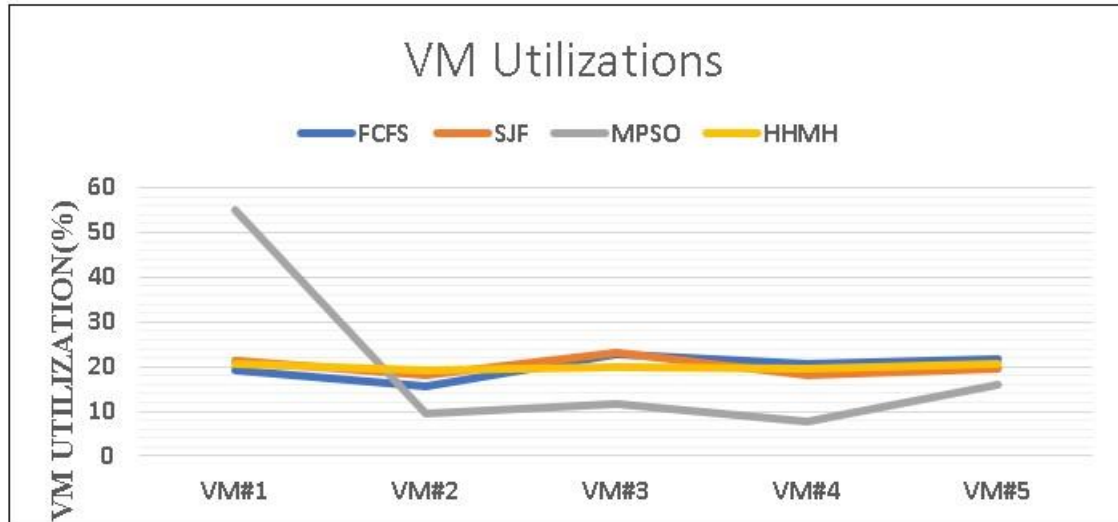


Figure 6-1 VM utilization in terms of tasks performed by each VMs

6.1.2 Average Response Time

The developed HHMH algorithm aims to schedule the tasks in such a way that task response time should be minimal. If the number of tasks on VM increases, the response time will also increase. Eleven schedules are generated to analyzed and test the execution of the developed HHMH algorithm in the cloud-fog environment as shown in Table 6.2. Figure 6.2 represents the response time comparison between the developed HHMH algorithm and other algorithms like MPSO, FCFS, and SJF for eleven schedules. It was noted that when the number of tasks is increased, the established HHMH algorithm performs the task of each schedule at the lowest response time compared to other existing algorithms.

Table 6.2 Detail of task and calculated average response time of algorithms

<i>Schedule</i>	<i>No of tasks</i>	<i>FCFS</i>	<i>SJF</i>	<i>HHMH</i>	<i>MPSO</i>
S1	40	2.18695	2.1041	2.4354	2.525
S2	80	4.27025	4.27345	3.9539	6.6776
S3	120	6.26845	5.671167	5.813733	11.20183
S4	160	8.418487	8.208875	8.123675	16.2674
S5	200	10.53398	9.64878	9.93246	20.0178
S6	250	13.63596	11.51762	12.5535	18.2361
S7	300	16.00575	15.32505	14.77843	33.01836

S8	350	18.95881	17.5348	16.57635	43.14105
S9	400	21.94394	19.57154	18.98307	33.02459
S10	450	24.48011	23.44534	22.56825	31.9221
S11	500	27.35171	24.72158	24.20458	53.68536

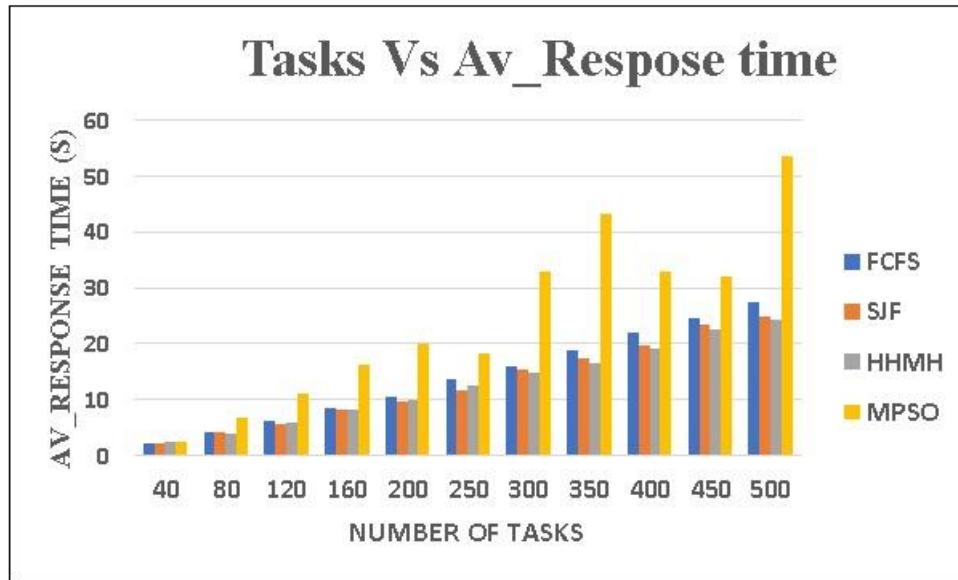


Figure 6-2 Average response time comparison between HHMH Vs FCFS, SJF, and MPSO

6.1.3 Execution Time Calculation

The goal of the HHMH algorithm developed is to minimize tasks time of execution submitted to the cloud-fog method. By varying the number of tasks in each schedule, we evaluated and investigated the established algorithm efficiency. Initially, we randomly generated 40 tasks and all the tasks are independent. A further range of tasks is expanded from 40 to 500 for the analysis (test) of the result of the HHMH algorithm developed. Cloudsim has been simulated for heterogeneous tasks and findings are analyzed. In [Table 6.3](#), simulation details are shown in terms of the number of tasks and execution time. As the number of tasks is bigger in the cloud-fog environment, execution time is increased, as publicized in [Figure 6.3](#). The execution time calculated is for all the 11 schedules.

Table 6.3 Detail of the task and calculated execution time of algorithms

<i>Schedule</i>	<i>No of tasks</i>	<i>FCFS</i>	<i>SJF</i>	<i>MPSO</i>	<i>HHMH</i>
S1	40	20.256	20.676	19.488	19.456
S2	80	45.498	38.944	42.476	37.016
S3	120	62.966	57.912	62.692	62.32
S4	160	87.124	80.848	76.612	79.304
S5	200	108.79	94.612	94.516	96.572
S6	250	132.554	128.076	125.548	122.416
S7	300	164.634	150.264	152.192	149.608
S8	350	188.518	171.7	173.516	170.38
S9	400	204.474	197.6	206.212	185.316
S10	450	239.636	216.52	225.48	210.136
S11	500	270.22	250.588	257.416	246.876

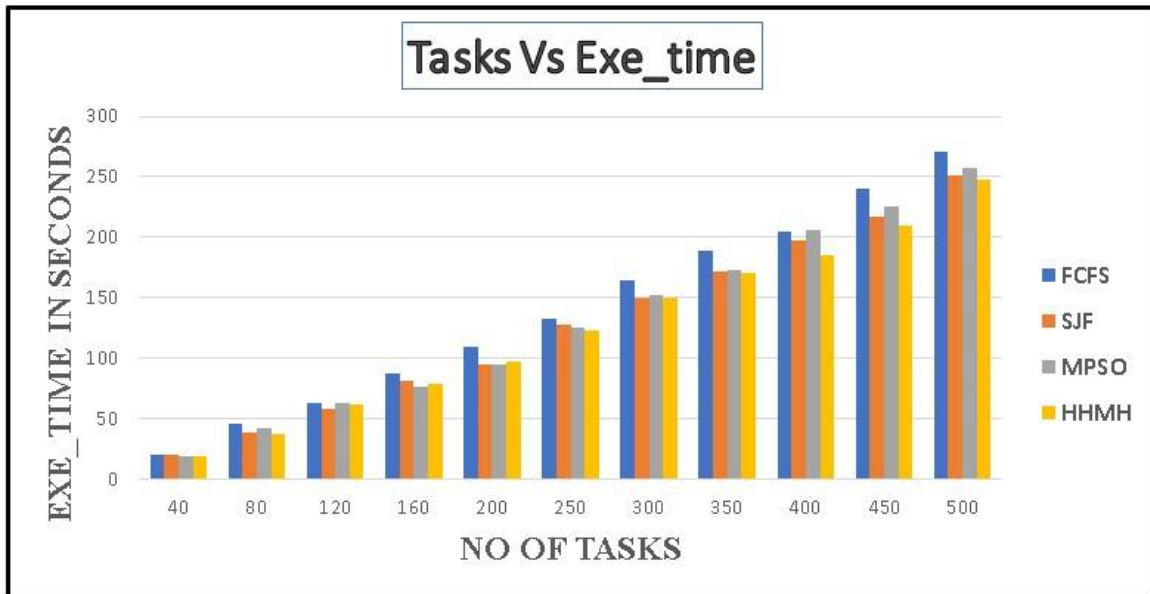


Figure 6-3 Execution time comparison between HHMH Vs FCFS, SJF, and MPSO

It is compared to the FCFS, SJF, and MPSO scheduling algorithm to test the routine or performance of the developed HHMH algorithm (for 40-500 tasks) and found that calculated execution time by the developed HHMH algorithm is better while the number of tasks increases.

As far as the number of tasks is considered, the proposed HHMH solution is therefore considered more successful than FCFS, SJF, and MPSO in the cloud-fog environment.

6.1.4 Throughput

The result of the developed HHMH algorithm for parameter throughput has been evaluated. Eleven different schedules are created, as shown in Table 6.4, to test the task execution rate per second of the algorithms. The results in Figure 6.4 show that the established HHMH algorithm outperforms the existing algorithm like FCFS.SJF, MPSO. The number of tasks extending from 40 to 500 is included in each schedule.

Compared to other algorithms, the developed HHMH algorithm performs more tasks in a second (average 2.3 tasks per second); MPSO performs 2.02 tasks per second; FCFS performs 1.97 tasks per second, and SJF performs 2.19 tasks per second for 40 tasks. For all the tasks, we measured the throughput and found that the developed HHMH algorithm performs about 2.19 tasks per second, while MPSO performs 2.01 tasks per second, FCFS performs about 1.88 tasks per second, and SJF performs 1.99 tasks per second. As the quantity of tasks increases, the throughput of the cloud-fog environment decreases slightly because the processing time increases. Both Table 6.4 and Figure 6.4 show that in the entire schedule, the developed HHMH algorithm has better throughput than another existing algorithm.

Table 6.4 Simulation details to find the throughput of the algorithms

<i>Schedule</i>	<i>No of tasks</i>	<i>FCFS</i>	<i>SJF</i>	<i>HHMH</i>	<i>MPSO</i>
S1	40	1.967342	2.188663	2.306805	2.021427
S2	80	1.86785	2.131969	2.262955	2.1777
S3	120	1.838686	1.982685	2.055921	2.004812
S4	160	1.833769	1.971123	2.046769	1.992727
S5	200	1.909563	1.98114	2.008032	1.972231
S6	250	1.943922	2.015609	2.003847	1.947234
S7	300	1.857838	1.982239	2.069594	2.040705
S8	350	1.893468	2.016547	2.096863	1.970898
S9	400	1.87737	2.009969	2.002523	1.982671
S10	450	1.892729	2.007781	2.090845	1.990798

S11	500	1.869257	1.987408	2.19225	2.025538
-----	-----	----------	----------	---------	----------

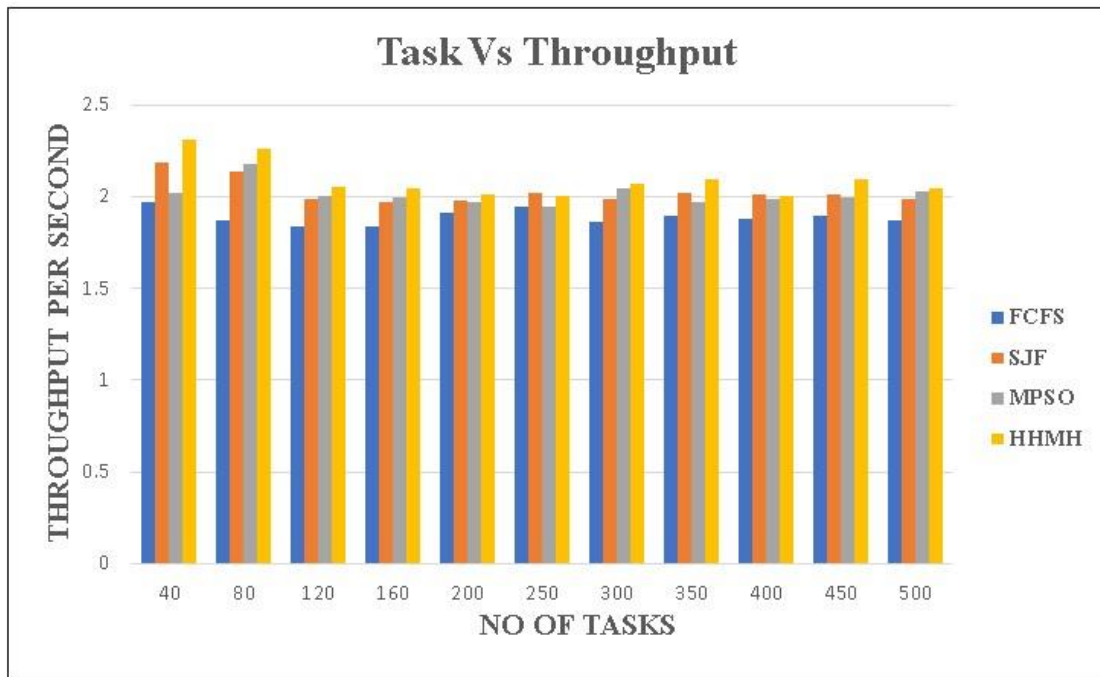


Figure 6-4 Throughput comparison between HHMH Vs FCFS, SJF and MPSO

CHAPTER 7

7 CONCLUSION AND FUTURE WORKS

7.1 Conclusion

An effective scheduling algorithm can provide users with more desirable services and improve the performance offered by the cloud or fog system. The intention of scheduling tasks in a cloud and fog environment is to decrease task response time, task execution time, and optimize the usage of VM. Different related papers with the existing algorithm for task scheduling in a cloud and fog environment are discussed briefly. A short overview of each methodology for the algorithm has been provided and most algorithms consider one or two parameters. By adding more metrics to existing algorithms, more satisfactory results can be obtained. [Table 2.1](#) presents various parameters of scheduling, such as load balance, execution time, service quality, performance, response time, and makespan. Response time, resource usage, and throughput are the main important parameters in task scheduling. By combining various parameters with existing algorithms that will boost their overall computing environment efficiency, an effective scheduling algorithm can be achieved.

In this work, we focused on task scheduling problems within the Cloud–Fog computing environment. To deal with the limitation of traditional cloud (delay), fog computing is used, which provides the next level of decentralization and heterogeneity. Some task scheduling algorithms within the cloud and fog environment considered certain parameters but none of the reported algorithms considered effective VM use, average response time, throughput, and execution time simultaneously. We developed FCFS and MPSO based scheduling algorithm named HHMH that schedules the tasks at cloud-fog resources in an efficient way and optimizes the parameters (average execution time, throughput, average response time, throughput, and VM utilization). HHMH algorithm is verified and tested at Cloudsim simulator and experimental results are shown in [Figure 6.1](#) to [Figure 6.4](#). The result proved that the developed algorithm decreases the execution time and response time, optimizes the throughput, and utilizes the VM

in an efficient way instead of the existing MPSO, FCFS, and SJF algorithm in all over 11 sets of tasks.

7.2 Future Works

We will research, improve, and apply more methods to solve other scheduling problems in the future. We recommended for another motivated researcher to develop a machine-learning approach to predict the upcoming workload in the Cloud-Fog environment. It is also better to develop and evaluate the proposed algorithm in the real environment rather than a simulation toolkit.

Special Acknowledgment

This research project is funded by Adama Science and Technology University under the grant number:

ASTU/SM-R/ 081/19

Adama, Ethiopia

8 REFERENCES

- [1] S. MS., “an effective analysis on various scheduling algorithms in cloud computing.”, *Inventive Computing and Informatics*, vol. 98, p. 931–6, 2018.
- [2] R. K. Yu J, Buyya R, “workflow scheduling algorithms for grid computing”, vol. 98, p. 173–214, 2018.
- [3] A. V. Dastjerdi and R. Buyya, “Fog Computing: Helping the Internet of Things Realize Its Potential,” pp. 112–116, 2016.
- [4] J. M. et al., “Unlocking the potential of the internet of things,” pp. 112–116, 2015.
- [5] R. C. et al., “stream processing of healthcare sensor data: Studying user traces to identify challenges from a big data perspective,” vol. 52, p. 1004–1009, 2015.
- [6] B. M. Nguyen, H. T. T. Binh, T. T. Anh, and D. B. Son, “Evolutionary algorithms to optimize task scheduling problem for the IoT based Bag-of-Tasks application in Cloud-Fog computing environment,” *Applied Sciences (Switzerland)*, vol. 9, no. 9, 2019.
- [7] G. A. S. J. S. K. A. Toor, S. ul Islam and A. M. Sharif, “energy-efficient edge-of-things”, *EURASIP Journal on Wireless Communications and Networking*, vol. 49, no. 5, p. 82, 2019.
- [8] S. C. S. Sarkar and S. Misra, “assessment of the suitability of fog computing in the context of internet of things”, *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, p. 46–59, 2018.
- [9] H. Rafique, M. A. Shah, S. U. Islam, T. Maqsood, S. Khan, and C. Maple, “A Novel Bio-Inspired Hybrid Algorithm (NBIHA) for Efficient Resource Management in Fog Computing,” *IEEE Access*, vol. 7, pp. 115 760–115 773, 2019.
- [10] “fog computing architecture diagram - google search,” https://www.google.com/search?q=fog+computing+architecture+diagram&tbm=isch&ved=2ahUKEwiCrY_S76TuAhUE8IUkHRNeD2AQ2-cCegQIABAA&oq=fog+computing+&gs_lcp=

CgNpbWcQARgCMgQIIxAnMgIIADICCAAyAggAMgIIADICCAAyAggAMgIIADI
CCAAyAggAUJsclient=img&ei=Bi0FYIKSDITglwSTvL2ABg&bih=606&biw=1396,
(Accessed on 01/17/2021).

[11] L. S. M. Mukherjee and D. Wang, “survey of fog computing: Fundamental, network applications, and research challenges,” *IEEE Communications Surveys*, vol. 20, no. 3, p. 1826–1857, 2018.

[12] H. N. P. Hu, S. Dhelim and T. Qiu, “survey on fog computing: architecture, key technologies, applications, and open issues,” *Journal of network and computer applications*, vol. 98, p. 27–42, 2018.

[13] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, “A comprehensive survey for scheduling techniques in cloud computing,” *Journal of Network and Computer Applications*, vol. 143, no. January, pp. 1–33, 2019. [Online]. Available: <https://doi.org/10.1016/j.jnca.2019.06.006>

[14] M. S. Sudheer, M. Vamsi Krishna², and A. Anurag, “A Review on Metaheuristic Scheduling Algorithms in Cloud Computing,” *Indian Journal of Science and Technology*, vol. 12, no. 23, pp. 1–8, 2019.

[15] Y. H. Wang and I. C. Wu, “Achieving high and consistent rendering performance of java AWT/Swing on multiple platforms,” *Software - Practice and Experience*, vol. 39, no. 7, pp. 701–736, 2009.

[16] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments,” *Software - Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[17] L. Tom and V. R. Bindu, “Task Scheduling Algorithms in Cloud Computing: A Survey,” *Lecture Notes in Networks and Systems*, vol. 98, pp. 342–350, 2020.

- [18] A. K. Mashuqur Rahman Mazumder, K. M. Aslam Uddin, N. Arbe, L. Jahan, and M. Whaiduzzaman, "Dynamic task scheduling algorithms in cloud computing," *Proceedings of the 3rd International Conference on Electronics and Communication and Aerospace Technology, ICECA 2019*, pp. 1280–1286, 2019.
- [19] N. Almansour and N. M. Allah, "A survey of scheduling algorithms in cloud computing," *2019 International Conference on Computer and Information Sciences, ICCIS 2019*, pp.1–6, 2019.
- [20] H. Kaur, "Available Online at www.ijarcs.info A Review on Security Issues in Cloud Computing," vol. 6, no. 2, pp. 178–183, 2015.
- [21] N. Arora, "Review on Task Scheduling Algorithms in Cloud Computing Environment," *International Journal*, vol. 8, no. 4, pp. 462–468, 2017.
- [22] B. H. Malik, M. Amir, B. Mazhar, S. Ali, R. Jalil, and J. Khalid, "Comparison of task scheduling algorithms in the cloud environment," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 5, pp. 384–390, 2018.
- [23] A. Jyoti, M. Shrimali, and R. Mishra, "Cloud computing and load balancing in cloud computing-survey," in *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, 2019, pp. 51–55.
- [24] H. Alshareef, M. Almasri, A. Albeshier, and D. Grigoras, "Towards an effective management of iot by integrating cloud and fog computing," in *2019 IEEE International Conference on Smaverage response time Internet of Things (Smaverage response timeIoT)*. IEEE, 2019, pp. 197–204.
- [25] "Fog computing vs. Cloud computing: Key differences | sam solutions," <https://www.sam-solutions.com/blog/fog-computing-vs-cloud-computing-for-iot-projects/>, (Accessed on 02/09/2020).
- [26] L. Tripathy and R. R. Patra, "Scheduling in cloud computing," *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, vol. 4, no. 5, pp. 21–7, 2014.

- [27] M. Sajid and Z. Raza, "Cloud computing: Issues & challenges," in *International Conference on Cloud, Big Data and Trust*, vol. 20, no. 13, 2013, pp. 13–15.
- [28] A. Yadav and S. Rathod, "Study of scheduling techniques in cloud computing environment," *Int. J. Comput. Trends Technol*, vol. 29, pp. 69–73, 2015.
- [29] M. Kumar, S. Sharma, A. Goel, and S. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, 2019.
- [30] "Heterogeneous earliest finish time - Wikipedia," https://en.wikipedia.org/wiki/Heterogeneous_Earliest_Finis_Time, (Accessed on 02/02/2020).
- [31] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.
- [32] W. Li and H. Shi, "Dynamic load balancing algorithm based on fcfs," in *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*. IEEE, 2016, pp. 1528–1531.
- [33] E. Nandi, R. K. Mondal, P. Ray, B. Biswas, M. K. Sanyal, and D. Sarddar, "Improved cost-effective technique for resource allocation in mobile cloud computing," in *Progress in Computing, Analytics and Networking*. Springer, 2018, pp. 551–558.
- [34] P. Samal and P. Mishra, "Analysis of variants in round robin algorithms for load balancing in cloud computing," *International Journal of computer science and Information Technologies*, vol. 4, no. 3, pp. 416–419, 2013.
- [35] E. De Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens, and B. Dhoedt, "Dynamic auto-scaling and scheduling of deadline constrained service workloads on iaas clouds," *Journal of Systems and Software*, vol. 118, pp. 101–114, 2016.

- [36] B. Mazhar, R. Jalil, J. Khalid, M. Amir, S. Ali, and B. H. Malik, "Comparison of task scheduling algorithms in cloud environment," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 5, pp. 384–390, 2018.
- [37] N. Arora, "Review on task scheduling algorithms in cloud computing environment," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 4, 2017.
- [38] P. Kumar and P. K. D. Pramanik, "Host selection methodology in cloud computing environment," *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, vol. 1, no. 8, pp. 1–5, 2012.
- [39] M.-C. Shan and M. C. Murphy, "Method of automatically controlling the allocation of resources of a parallel processor computer system by calculating a minimum execution time of a task and scheduling subtasks against resources to execute the task in the minimum time," Jun. 28 1994, uS Patent 5,325,525.
- [40] X. He, X. Sun, and G. Von Laszewski, "Qos guided min-min heuristic for grid task scheduling," *Journal of computer science and technology*, vol. 18, no. 4, pp. 442–451, 2003.
- [41] R. Eberhaverage response time and J. Kennedy, "particle swarm optimization," in *Proceedings of the IEEE international conference on neural networks*, vol. 4. Citeseer, 1995, pp. 1942–1948.
- [42] R. V. Kulkarni and G. K. Venayagamoorthy, "particle swarm optimization in wireless sensor networks: A brief survey," *IEEE Transactions on Systems, Man, and Cybernetics, part C (Applications and Reviews)*, vol. 41, no. 2, pp. 262–267, 2010.
- [43] M. R. Mozafar, M. H. Moradi, and M. H. Amini, "A simultaneous approach for optimal allocation of renewable energy sources and electric vehicle charging stations in smart grids based on an improved-PSO algorithm," *Sustainable cities and society*, vol. 32, pp. 627–637, 2017.

- [44] L. Yi, “Study on an improved PSO algorithm and its application for solving function problem,” *Int. J. Smart Home*, vol. 10, no. 3, pp. 51–62, 2016.
- [45] F. Ramezani, J. Lu, and F. K. Hussain, “Task-based system load balancing in cloud computing using particle swarm optimization,” *International journal of parallel programming*, vol. 42, no. 5, pp. 739–754, 2014.
- [46] A. Verma and S. Kaushal, “Bi-criteria priority-based particle swarm optimization workflow scheduling algorithm for cloud,” in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*. IEEE, 2014, pp. 1–6.
- [47] M. Adhikari and S. N. Srirama, “Multi-objective accelerated particle swarm optimization with a container-based scheduling for internet-of-things in cloud environment,” *Journal of Network and Computer Applications*, vol. 137, pp. 35–61, 2019.
- [48] S. Zhan and H. Huo, “Improved pso-based task scheduling algorithm in cloud computing,” *Journal of Information & Computational Science*, vol. 9, no. 13, pp. 3821–3829, 2012.
- [49] M. Elhoseny, A. S. Salama, A. Abdelaziz, and A. M. Riad, “Intelligent systems based on cloud computing for healthcare services: a survey,” *IJCISudies*, vol. 6, no. 2/3, pp. 157–188, 2017.
- [50] D. Zheng, “Research on hazardous chemical road transportation route optimization based on the new ant colony algorithm,” *Chemical Engineering Transactions*, vol. 62, pp. 1375–1380, 2017.
- [51] W. Sun, N. Zhang, H. Wang, W. Yin, and T. Qiu, “Paco: A period aco based scheduling algorithm in cloud computing,” in *2013 International Conference on Cloud Computing and Big Data*. IEEE, 2013, pp. 482–486.
- [52] E. Pacini, C. Mateos, and C. G. Garino, “Balancing throughput and response time in online scientific clouds via ant colony optimization (sp2013/2013/00006),” *Advances in Engineering Software*, vol. 84, pp. 31–47, 2015.

- [53] A. Arunarani, D. Manjula, and V. Sugumaran, "Task scheduling techniques in cloud computing: A literature survey," *Future Generation Computer Systems*, vol. 91, pp.407–415, 2019.
- [54] D. B. LD and P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292–2303, 2013.
- [55] S. Singh and M. Kalra, "Scheduling of independent tasks in cloud computing using modified genetic algorithm," in *2014 International Conference on Computational Intelligence and Communication Networks*. IEEE, 2014, pp. 565–569.
- [56] M. Ali, C. W. Ahn, and P. Siarry, "Differential evolution algorithm for the selection of optimal scaling factors in image watermarking," *Engineering Applications of Artificial Intelligence*, vol. 31, pp. 15–26, 2014.
- [57] Y.-P. Chen, Y. Li, G. Wang, Y.-F. Zheng, Q. Xu, J.-H. Fan, and X.-T. Cui, "A novel bacterial foraging optimization algorithm for feature selection," *Expert Systems with Applications*, vol. 83, pp. 1–17, 2017.
- [58] X.-S. Yang, "A new metaheuristic bat-inspired algorithm," in *Nature inspired cooperative strategies for optimization (NICSO 2010)*. Springer, 2010, pp. 65–74.
- [59] A. Askarzadeh, "A novel metaheuristic method for solving constrained engineering optimization problems: crow search algorithm," *Computers & Structures*, vol. 169, pp.1–12, 2016.
- [60] R. Kaur and N. Ghumman, "Hybrid improved max min ant algorithm for load balancing in cloud," in *Proceedings of the International Conference on Communication, Computing and Systems (CCS'14)*, 2014, pp. 188–191.
- [61] S. Yassa, R. Chelouah, H. Kadima, and B. Granado, "Multi-objective approach for energy-aware workflow scheduling in cloud computing environments," *The Scientific World Journal*, vol. 2013, 2013.

- [62] R. M. Guddeti, R. Buyya *et al.*, “A hybrid bio-inspired algorithm for scheduling and resource management in cloud environment,” *IEEE Transactions on Services Computing*, 2017.
- [63] H. Rafique, M. A. Shah, S. U. Islam, T. Maqsood, S. Khan, and C. Maple, “A Novel Bio-Inspired Hybrid Algorithm (NBIHA) for Efficient Resource Management in Fog Computing,” *IEEE Access*, vol. 7, pp. 115 760–115 773, 2019.
- [64] M. Kumar and S. Sharma, “Pso-cogent: Cost and energy efficient scheduling in cloud environment with deadline constraint,” *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 147–164, 2018.
- [65] M. Kumar and S. C. Sharma, “Priority aware longest job first (pa-ljf) algorithm for utilization of the resource in cloud environment,” in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2016, pp.415–420.
- [66] P. Azad and N. J. Navimipour, “An energy-aware task scheduling in the cloud computing using a hybrid cultural and ant colony optimization algorithm,” *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 7, no. 4, pp. 20–40, 2017.
- [67] J.-T. Tsi, J.-C. Fang, and J.-H. Chou, “Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm,” *Computers & Operations Research*, vol. 40, no. 12, pp. 3045–3055, 2013.
- [68] M. Masdari, F. Salehi, M. Jalali, and M. Bidaki, “A survey of pso-based scheduling algorithms in cloud computing,” *Journal of Network and Systems Management*, vol. 25, no. 1, pp. 122–158, 2017.
- [69] N. Netjinda, B. Sirinaovakul, and T. Achalakul, “Cost optimal scheduling in iaas for dependent workload with particle swarm optimization,” *The Journal of Supercomputing*, vol. 68, no. 3, pp. 1579–1603, 2014.
- [70] S. Yassa, R. Chelouah, H. Kadima, and B. Granado, “Multi-objective approach for energy-aware workflow scheduling in cloud computing environments,” *The Scientific World Journal*, vol. 2013, 2013.

- [71] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp.26–35, 2017.
- [72] S. S. Patra, "Energy-efficient task consolidation for cloud data center," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 8, no. 1, pp. 117–142, 2018.
- [73] J. Collis and R. Hussey, "Business research (ed.)," 2003.
- [74] R. Bell, J. Dahl, R. Moore, and E. Kalso, "Peri-operative ketamine for acute post-operative pain: a quantitative and qualitative systematic review (cochrane review)," *Acta anaesthesiologica scandinavica*, vol. 49, no. 10, pp. 1405–1428, 2005.
- [75] A. J. Onwuegbuzie and N. L. Leech, "Sampling designs in qualitative research: Making the sampling process more public." *Qualitative Report*, vol. 12, no. 2, pp. 238–254, 2007.
- [76] "Eclipse foundation | the eclipse foundation," <https://www.eclipse.org/org/foundation/>, (Accessed on 02/26/2020).
- [77] "Edraw max - wikipedia," https://en.wikipedia.org/wiki/Edraw_Max, (Accessed on 02/26/2020).
- [78] "Latex - a document preparation system," <https://www.latex-project.org/>, (Accessed on 02/26/2020).

Appendix A List of Java Codes

A.1 HHMH-Scheduler

```
1  public class HHMH_Scheduler {
2  private static List<Vm> createVM(int userId, int vms) {
3  LinkedList<Vm> list = new LinkedList<Vm>();
4  long size = 10000;
5  int ram = 512;
6  int mips = 250;
7  long bw = 1000;
8  int pesNumber = 1;
9  String vmm = "Xen";
10 Vm[] vm = new Vm[vms];
11 for (int i = 0; i < vms; i++) {
12 vm[i] = new Vm(datacenter[i].getId(), userId, mips, pesNumber, ram, bw,
    size, vmm,
13 new CloudletSchedulerSpaceShared());
14 list.add(vm[i]);
15 }
16 return list;
17 }
18 private static List<Cloudlet> createCloudlet(int userId, int cloudlets,
    int idShift) {
19 LinkedList<Cloudlet> list = new LinkedList<Cloudlet>();
20 long length1 = 50;
21 long fileSize = 10;
22 long outputSize = 10;
23 int pesNumber = 1;
24 UtilizationModel utilizationModel = new UtilizationModelFull();
25 Cloudlet[] cloudlet = new Cloudlet[cloudlets];
26 for (int i = 0; i < cloudlets; i++) {
```

```

27 Random rojb = new Random();
28 int dcId = (int) (Math.random() * Constants.NO_OF_DATA_CENTERS);
29 long length = (long) (1e3 * (commMatrix[i][dcId] + execMatrix[i][dcId]));
30 cloudlet[i] = new Cloudlet(idShift + i, length1 +
31 rojb.nextInt(150), pesNumber, fileSize + rojb.nextInt(90),
32 outputSize + rojb.nextInt(90), utilizationModel, utilizationModel,
    utilizationModel);
33 cloudlet[i].setUserId(userId)
34 cloudlet[i].setVmId(dcId + 2);
35 list.add(cloudlet[i]);
36 }
37 return list;
38 }
39 public static void main(String[] args) {
40 Log.println("Staverage response timeing HHMH-Scheduler...");
41 new GenerateMatrices();
42 commMatrix = GenerateMatrices.getCommMatrix();
43 execMatrix = GenerateMatrices.getExecMatrix();
44 PSOSchedularInstance = new PSO();
45 mapping = PSOSchedularInstance.run();
46 try {
47 int num_user = 1;
48 Calendar calendar = Calendar.getInstance();
49 boolean trace_flag = false;
50 CloudSim.init(num_user, calendar, trace_flag);
51 datacenter = new Datacenter[Constants.NO_OF_DATA_CENTERS];
52 for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
53 datacenter[i] = DatacenterCreator.createDatacenter("Datacenter_" + i);
54 }
55 HHMHDatacenterBroker broker = createBroker("Broker_0");
56 int brokerId = broker.getId();
57 vmList = createVM(brokerId, Constants.NO_OF_DATA_CENTERS);
58 cloudletList = createCloudlet(brokerId, Constants.NO_OF_TASKS, 0);
59 HashSet<Integer> dcIds = new HashSet<>();

```

```

60 HashMap<Integer, Integer> hm = new HashMap<>();
61 for (Datacenter dc : datacenter) {
62 if (!dcIds.contains(dc.getId()))
63 dcIds.add(dc.getId());
64 }
65 Iterator<Integer> it = dcIds.iterator();
66 for (int i = 0; i < mapping.length; i++) {
67 if (hm.containsKey((int) mapping[i]))
68 continue;
69 hm.put((int) mapping[i], it.next());
70 }
71 for (int i = 0; i < mapping.length; i++)
72 mapping[i] = hm.containsKey((int) mapping[i]) ? hm.get((int) mapping[i]) :
    mapping[i];
73 broker.submitVmList(vmList);
74 broker.submitCloudletList(cloudletList);
75 CloudSim.staverage response timeSimulation();
76 List<Cloudlet> newList = broker.getCloudletReceivedList();
77 CloudSim.stopSimulation();
78 printCloudletList(newList);
79 Log.println(HHMH_Scheduler.class.getName() + " finished!");
80 } catch (Exception e) {
81 e.printStackTrace();
82 Log.println("The simulation has been terminated due to an unexpected
    error")
83 }}private static HHMHDatacenterBroker createBroker(String name) throws
    Exception {
84 return new HHMHDatacenterBroker(name);}

```

A.2 DatacenterCreator

```
1  public class DatacenterCreator {
2      public static Datacenter createDatacenter(String name) {
3          List<Host> hostList = new ArrayList<Host>();
4          List<Pe> peList = new ArrayList<Pe>();
5          int mips = 500;
6          for (int i = 0; i < 10; i++) {
7              Random rojb = new Random();
8              PeProvisionerSimple pProvisioner = new PeProvisionerSimple(mips + rojb.
                  nextInt(4500));
9              Pe core = new Pe(i, pProvisioner);
10             peList.add(core);
11         }
12         int ram = 2048;
13         long storage = 1000000;
14         int bw = 10000;
15         for (int i = 0; i < 10; i++) {
16             Random rojb1 = new Random();
17             hostList.add(new Host(i, new RamProvisionerSimple(ram + rojb1.nextInt
                  (100)), new BwProvisionerSimple(bw + rojb1.nextInt(100)), storage +
                  rojb1.nextInt(100),
18             new VmSchedulerSpaceShared(peList))); // This is our first machine
19         }
20         String arch = "x86";
21         String os = "Windows";
22         String vmm = "Xen";
23         double time_zone = 10.0;
24         double cost = 3.0;
25         double costPerMem = 0.05;
26         double costPerStorage = 0.1;
27         double costPerBw = 0.1;
28         LinkedList<Storage> storageList = new LinkedList<Storage>();
```

```

29 DatacenterCharacteristics characteristics = new
    DatacenterCharacteristics(arch, os, vmm, hostList, time_zone, cost,
        costPerMem, costPerStorage, costPerBw);
30 Datacenter datacenter = null;
31 try {
32 datacenter = new Datacenter(name, characteristics, new
    VmAllocationPolicySimple(hostList), storageList, 0);
33 } catch (Exception e) {
34 e.printStackTrace();}
35 return datacenter;} }

```

A.3 SchedulerFitnessFunction

```

1 public class SchedulerFitnessFunction extends FitnessFunction {
2 private static double[][] execMatrix, commMatrix;
3 SchedulerFitnessFunction() {
4 super(false);
5 commMatrix = GenerateMatrices.getCommMatrix();
6 execMatrix = GenerateMatrices.getExecMatrix();
7 }
8 @Override
9 public double evaluate(double[] position) {
10 double alpha = 0.3;
11 return alpha * calcTotalTime(position) + (1 - alpha) * calcMakespan
    (position);
12 }
13 private double calcTotalTime(double[] position) {
14 double totalCost = 0;
15 for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
16 int dcId = (int) position[i];
17 int dcId = (int) position[i];
18 totalCost += execMatrix[i][dcId] + commMatrix[i][dcId];
19 }
20 return totalCost;}

```

```

21 public double calcMakespan(double[] position) {
22     double makespan = 0;
23     double[] dcWorkingTime = new double[Constants.NO_OF_DATA_CENTERS];
24     for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
25         int dcId = (int) position[i];
26         if (dcWorkingTime[dcId] != 0)
27             —dcWorkingTime[dcId];
28         dcWorkingTime[dcId] += execMatrix[i][dcId] + commMatrix[i][dcId];
29         makespan = Math.max(makespan, dcWorkingTime[dcId]);
30     }
31     return makespan;}}

```

A.4 SchedulerPaverage response timeicle

```

1  public class SchedulerPaverage response timeicle extends Particle {
2      SchedulerPaverage response timeicle() {
3          super(Constants.NO_OF_TASKS);
4          double[] position = new double[Constants.NO_OF_TASKS];
5          double[] velocity = new double[Constants.NO_OF_TASKS];
6          for (int i = 0; i < Constants.NO_OF_TASKS; i++) {
7              Random randObj = new Random();
8              position[i] = randObj.nextInt(Constants.NO_OF_DATA_CENTERS);
9              velocity[i] = Math.random();}
10         setPosition(position);
11         setVelocity(velocity);}
12     @Override
13     public String toString() {
14         String output = "";
15         for (int i = 0; i < Constants.NO_OF_DATA_CENTERS; i++) {
16             String tasks = "";
17             int no_of_tasks = 0;

```

```

18 for (int j = 0; j < Constants.NO_OF_TASKS; j++) {
19 if (i == (int) getPosition()[j]) { tasks += (tasks.isEmpty() ? "" : "
    ") + j; ++no_of_tasks; }}
20 if (tasks.isEmpty()) output += "There is no tasks associated to Data
    Center " + i + "\n";
21 else
22 output += "There are " + no_of_tasks + " tasks associated to Data Center
    " + i + " and they are " + tasks + "\n";}
23 return output;}

```

A.4 FCFSDatacenterBroker

```

1 public class FCFSDatacenterBroker extends DatacenterBroker {
2 public FCFSDatacenterBroker(String name) throws Exception {
3 super(name);
4 }
5 public void scheduleTaskstoVms() {
6 ArrayList<Cloudlet> clist = new ArrayList<Cloudlet>();
7 for (Cloudlet cloudlet : getCloudletSubmittedList()) {
8 clist.add(cloudlet);
9 }
10 setCloudletReceivedList(clist);
11 @Override
12 protected void processCloudletReturn(SimEvent ev) {
13 Cloudlet cloudlet = (Cloudlet) ev.getData();
14 getCloudletReceivedList().add(cloudlet);
15 Log.println(CloudSim.clock() + ": " + getName() + ": Cloudlet "
    + cloudlet.getCloudletId() + " received");
16 cloudletsSubmitted--;
17 if (getCloudletList().size() == 0 && cloudletsSubmitted == 0) {
18 scheduleTaskstoVms();
19 cloudletExecution(cloudlet);}}

```