UNIVERSITY OF SOUTHAMPTON
Faculty of Physical Sciences and Engineering
Electronics and Computer Science

Optimizing Resource Allocation using Multi-Objective Particle Swarm Optimization in Cloud Computing Systems

by
Entisar Alkayal

Supervisors: Professor Nicholas Jennings and Doctor Maysoon Abulkhair

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Physical Sciences and Engineering
Electronics and Computer Science
Agents, Interaction and Complexity Group

January 2018

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING
ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Entisar Alkayal

Allocating resources in data centers is a complex task due to their increase in size, complexity, and consumption of power. At the same time, consumers' requirements regarding execution time and cost have become more sophisticated and demanding. These requirements often conflict with the objectives of cloud providers. Set against this background, this thesis presents a model of resource allocation in cloud computing environments that focuses on developing the allocation process in three phases: (i) negotiation between consumers and providers to select the data center, (ii) scheduling tasks inside data centers, and (iii) scheduling virtual machines (VMs) to physical machines. The proposed model attempts to optimize each phase by applying multi-objective optimization (MOO) and many-objective optimization (MaOO) using a particle swarm optimization (PSO) algorithm.

In more detail, a parallel PSO (PPSO) algorithm based on multi-objective was therefore developed to improve the SLA negotiation process between consumers and providers. The main insight of this algorithm is that SLA negotiation can be automated and the PSO can be parallelized to minimize negotiation time and to maximize system throughput, thus increasing the profits of providers.

A many-objective PSO (MaOPSO) algorithm based on a modified ranking strategy was developed to improve the task scheduling problem in each data center. The novelty of this algorithm lies in using a modified ranking strategy to minimize evaluation time and improve the quality of the results. The algorithm was executed within the constraints of the tight deadline to improve performance in terms of both waiting time and completion time.

Finally, VM allocation was improved by applying a many-objective PSO to allocate VMs in physical machines after clustering the hosts. Here the novelty lies in applying PSO and K-means when clustering hosts to improve VM allocation and migration, thus maximizing resource utilization and performance whilst reducing power consumption.

Most notably, SLA Negotiation reduced waiting time and completed time by up to 20%. Additionally, it increased the throughput by about 20%. The proposed SLA negotiation reduced the rates of SLA violations by about 25%. On the other hand, the proposed MaOPSO task algorithm reduced the waiting time and completed time by 15% and 20% respectively. It increased the throughput up to 15% and the profits up to 15%.

With respect to MaOPSO VM allocation, it improved resource utilization by up to 20%. Additionally, it reduced the power consumption by 25% compared to other algorithms. Profits are indirectly increased by improving utilization up to 20%. Finally, the MaOPSO VM algorithm led to an increased throughput of 20%, a reduced waiting time of 15%, and reduced the completed time up to 15%.

Contents

List of Figures vi
List of Tables viii
List of Algorithms x
Declaration of Authorship xi
Acknowledgements xii
List of Abbreviations xiii

1 Introduction 1
1.1 Resource Allocation in Cloud Computing……..…….....1
1.1.1 Cloud computing Overview…….…..…….....1
1.1.2 Resource Allocation Overview …….…….....4
1.1.3 Resource Allocation issues in Cloud Environments…..…….....7
1.1.4 Optimization Algorithms in Resource Allocation……..…....11
1.2 Challenges in Optimizing Resource Allocation ……17
1.2.1 Service Level Agreement Negotiation……...…….……17
1.2.2 Task Scheduling in Cloud Computing…….18
1.2.3 Virtual Machine Allocation……...……..20
1.3 Research Contributions ……..…….21
1.4 Thesis Structure .....…….....25

2 Related Work 27
2.1 Meta-Heuristic Optimization Algorithms……..……..…27
2.1.1 Overview……... 27
2.1.2 Swarm Intelligence Algorithms……...…30
2.1.2.1 Particle Swarm Optimization Algorithm……..……..…31
2.1.2.2 Ant Colony Optimization Algorithms …….…..…32
2.1.3 Evolutionary Algorithms…… 33
2.1.3.1 Genetic Algorithms……..….33
2.1.4 Methods of Evaluating Multiple Objectives…...……..35
2.2 Particle Swarm Optimization……..……...…….. 37
2.2.1 Overview……..……. 37
2.2.2 Improvements on Particle Swarm Optimization Algorithms 41
2.2.2.1 Modifying the Initializing of Particles……..... 41
2.2.2.2 Modifying Based on the Number of Objectives……42
2.2.3 Parallel Particle Swarm Optimization……..... 44
2.2.4 Clustering Based Particle Swarm Optimization……...…..48
2.3 Service Level Agreement Negotiation …… 51
2.3.1 Overview.……..……..... 51
2.3.2 Automated SLA Negotiation Based on Multiple Agents……....56
2.3.3 SLA Negotiation Based on Particle Swarm Optimization…...…58
2.3.4 SLA Negotiation Based on Parallel Algorithms……..…..……...59
2.3.5 SLA Monitoring……..60
2.3.6 Discussion of SLA Negotiation work …..…….....61

2.4 Task Scheduling in Cloud Computing 62
2.4.1 Overview... 63
2.4.2 Task Scheduling Based on Particle Swarm Optimization... 66
2.4.3 Task Scheduling Based on Meta-Heuristic Algorithms 69
2.4.4 Task Scheduling Based on Heuristic Algorithms....70
2.4.5 Real-Time Task Scheduling Algorithms....71
2.4.6 Discussion of Task Scheduling Algorithms....72
2.5 Virtual Machine Allocation 74
2.5.1 Virtual Machine Scheduling 76
2.5.2 Virtual Machine Migration 79
2.5.2.1 Host Detection Strategies.... 80
2.5.2.2 Virtual Machine Selection.... 82
2.5.2.3 Virtual Machine Placement.... 83
2.5.3 Discussion of VM Allocation Related Work....84
2.6 Summary.... 85

3 A Resource Allocation Model 88
3.1Overview.... 88
3.2 Design of an Optimized Resource Allocation Model....... 91
3.3 General Resource Allocation Architecture 92
3.4 General Resource Allocation Implementation.... 97
3.4.1 Configurations and Specifications of Resources.... 100
3.4.2 Evaluation Parameters 103
3.5 Summary 109

4 SLA Negotiation Based on Parallel Particle Swarm Optimization 110
4.1 Overview.... 110
4.2 SLA Negotiation Algorithm Formation.... 113
4.3 Sequential PSO Negotiation Algorithm.... 117
4.4 Parallel PSO Negotiation Algorithms.... 120
4.5 SLA Monitoring Algorithms.... 123
4.6 Parallel PSO Implementation.... 124
4.7 Experimental Evaluation.... 125
4.7.1 Experimental Methodology 125
4.7.2 Experimental Results 128
4.8 Summary.... 137

5 Task Scheduling Based on Many Objectives Particle Swarm Optimization 139
5.1 Overview.... 139
5.2 MaOPSO Task Scheduling Algorithm 142
5.3 MaOPSO Task Scheduling Implementation 149
5.4 Experimental Evaluation 149
5.4.1 The MaOPSO Algorithm: Analysis Results.... 151
5.4.2 Evaluating Ranking Method Efficiency in MaOPSO 152
5.4.3 Evaluating MaOPSO Compared to Other Scheduling 154
5.4.4 Evaluating MaOPSO Compared to Simple Ranking 158
5.5 Summary 161

6 Virtual Machine Allocation using Particle Swarm Optimization 162
6.1 Overview……. 162
6.2 Clustering Hosts Based on PSO and K-means Algorithms…… 164
6.3 Virtual Machine Scheduling Algorithm……... 168
6.4 Virtual Machine Migration Algorithm…….……... 173
6.5 Experimental Evaluation…….. 176
6.5.1 Experimental Methodology ……... 177
6.5.2 Experimental Results …… 178
6.6 Summary …… 188

7 Conclusions and Future Work 190
7.1 Conclusions…….…… 190
7.2 Future Work …..…… 191

References 196

List of Figures

1.1: Cloud Computing Technologies 2
1.2: Resource Allocation Levels in Cloud Computing. 5
1.3: Resource Allocation Optimization Objectives. 13
1.4: Resource Allocation Modules. 22
1.5: Proposed Resource Allocation Objectives. 23

2.1: Main Classification of Optimization Techniques. 29
2.2: Flowchart illustrating Parallel Particle Swarm Optimization. 46
2.3: PPSO Topologies. 48
2.4: SLA Management Life Cycle. 53
3.1: Optimized Resource Allocation Phases 90
3.2: General Architecture of Resource Allocation in Cloud 95
3.3: Manager Module Architecture. 96
3.4: Architecture of Provider Module. 96
3.5: The Entity Relationship Diagram for the Proposed Database. 97
3.6: CloudSim Architecture (Calheiros et al. (2011)). 100
3.7: Proposed Model Layers 100

4.1: SLA Negotiation Processes. 113
4.2: Flowchart for the SPSO Algorithm. 119
4.3: Negotiation Time Results (data shown with 95% confidence intervals). 129
4.4: Average Waiting Time Results (data shown with 95% confidence intervals). 129
4.5: Throughput Results (data shown with 95% confidence intervals). 130
4.6: Average Fitness Value Results (data shown with 95% confidence intervals). 131
4.7: Negotiation Time Results (data shown with 95% confidence intervals). 132
4.8: Speedup Results (data shown with 95% confidence intervals). 132
4.9: Average Waiting Time (data shown with 95% confidence intervals). 133
4.10: Average Completed Time (data shown with 95% confidence intervals). 134
4.11: Throughput Results (data shown with 95% confidence intervals). 135
4.12: SLA Violation Rate Results (data shown with 95% confidence intervals). 136
4.13: Total Profit Results (data shown with 95% confidence intervals). 137

5.1: Task Scheduling Phase. 141
5.2: Results of MaOPSO in terms of Minimizing Objectives. 151
5.3: Results of MaOPSO in terms of Maximizing Objectives. 152
5.4: Processing Time Results (data shown with 95% confidence intervals). 153
5.5: Average Fitness Results (data shown with 95% confidence intervals). 154
5.6: Waiting Time Results (data shown with 95% confidence intervals). 155
5.7: Completed Time Results (data shown with 95% confidence intervals). 156
5.8: Average Utilization of Resources (data shown with 95% confidence intervals). 157
5.9: Total Profits (data shown with 95% confidence intervals). 158
5.10: Processing Time Results (data shown with 95% confidence intervals). 159
5.11: Waiting Time Results (data shown with 95% confidence intervals). 160
5.12: Throughput Results (data shown with 95% confidence intervals). 160

6.1: Average Completion Time (data shown with 95% confidence intervals). 179
6.2: Average Waiting Time (data shown with 95% confidence intervals). 180
6.3: Throughput Results (data shown with 95% confidence intervals). 181
6.4: Average Resource Utilization (data shown with 95% confidence intervals). 182
6.5: Profit Results (data shown with 95% confidence intervals). 183
6.6: Power Consumption Results (data shown with 95% confidence intervals). 184
6.7: Imbalance Factor Results (data shown with 95% confidence intervals). 185
6.8: SLA Violation Rate Results (data shown with 95% confidence intervals). 186
6.9: Clustering Time Results (data shown with 95% confidence intervals). 187
6.10: Average Waiting Time Results (data shown with 95% confidence intervals). 187
6.11: Throughput Results (data shown with 95% confidence intervals). 188

List of Tables

3.1. Specification of Data centers 101
3.2. Specification of Host Types. 101
3.3: Specification of VM Types. 102

4.1: Setting of the Parameters for Experiments. 126
4.2: Setting of the Parameters for PSO. 126
4.3: Negotiation Time and Average Waiting Time Results (in seconds). 128
4.4: Throughput Results. 129
4.5: Average Fitness Value Results. 130
4.6: Negotiation Time (in Seconds) and Speedup Results. 131
4.7: Average Waiting Time and Average Completed Time Results (in seconds). 133
4.8: Throughput Results. 135
4.9: SLA Violation Rate Results. 136
4.10: Total Profit Results. 137

5.1: Particle Vector Direct Representation. 143
5.2: Task Execution Time (TET). 147
5.3: Task Execution Cost (TEC). 147
5.4: Data Transfer Time (DTT). 147
5.5: Data Transfer Cost (DTC). 148
5.6: VM Capacity (VMC). 148
5.7: VMs Rank Values. 148
5.8: The Setting of the Parameters for PSO. 149
5.9: Results of MaOPSO in terms of Minimizing and Maximizing Objectives. 151
5.10: Processing Time Results. 153
5.11: Average Fitness Results. 154
5.12: Waiting Time Results. 155
5.13: Completed Time Results. 156
5.14: Average Utilization of Resources. 157
5.15: Total Profits Results. 158

6.1: Average Completion Time Results (in seconds). 178
6.2: Average Waiting Time Results (in seconds). 179
6.3: Throughput Results. 180
6.4: Average Resource Utilization Results. 181
6.5: Profit Results. 182
6.6: Power Consumption Results. 183
6.7: Imbalance Factor Results. 184
6.8: SLA Violation Rate Results. 185
6.9: Waiting time Results. 186
6.10: Average Waiting Time Results. 187
6.11: Throughput Rate Results. 188

List of Algorithms

2.1: General Meta-Heuristic Algorithm 30
2.2: Ant Colony Optimization Algorithm 33
2.3: The pseudo code of the Genetic Algorithm 34
2.4: Pseudo code of a Standard PSO Algorithm 40
2.5: Pseudo code of a MOPSO Algorithm based on Pareto set. 43
2.6: K-means Clustering Algorithm. 50

4.1: SLA Negotiation Algorithm. 118
4.2: Sequential PSO Negotiation Algorithm. 120
4.3: Parallel PSO Negotiation Algorithm. 121
4.4: PSO Negotiation Algorithm in data center. 122
4.5: SLA Monitoring Algorithm. 123

5.1: Task scheduling algorithm. 143
5.2: MaOPSO Task scheduling algorithm. 144
5.3: Modified Ranking Strategy Algorithm. 147

6.1: VM Allocation Algorithm. 164
6.2: Clustering Hosts Algorithm. 166
6.3: KPSO Clustering Algorithm. 167
6.4: PSO Clustering Algorithm. 168
6.5: VM Scheduling Algorithm. 169
6.6: MaOPSO VM Scheduling Algorithm. 172
6.7: Updating Utilization after VM Scheduling. 172
6.8: VM Migration Algorithm. 174
6.9: Migration from under-loaded host. 175
6.10: Migration from high-loaded host. 175
6.11: Migration from high-loaded host. 176
6.12: Select Host for migration VM. 176

Declaration of Authorship

I declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;

2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;

3. Where I have consulted the published work of others, this is always clearly attributed;

4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;

5. I have acknowledged all main sources of help;

6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

7. Parts of this work have been published in a number of conference (see Section 1.3 for a list).

Acknowledgements

At the beginning, praise and gratitude be to ALLAH almighty, without His gracious help, it would have been impossible to accomplish this work. Working with this thesis has been a very interesting and valuable experience to me and I have learned a lot. I want to express my thanks to the people who have been very helpful during the time it took me to finish this thesis.

First, I would like to thank my supervisor Professor *Dr. Nicholas Jennings* who helped me with guidance, supervision, and constructive comments until I completed this work. My PhD has been an amazing experience and I thank him wholeheartedly, not only for his tremendous academic support, but also for giving me so many wonderful opportunities.

Similar, profound gratitude and deeply thankful goes to my supervisor *Dr.Maysoon Abu Al-Khair* for her efforts with me and for her unlimited help.

My special gratitude goes to my parents whose love and affection is the source of motivation and encouragement for my studies. I would like to thank all my family, all my sisters and all my brothers for unconditional support, and for simply being there. They were always supporting me and encouraging me with their best wishes.

Finally, I would like to thank my husband, *Abdulaziz Alzanbagi*. He was always there cheering me up and stood by me through the good times and bad. I thank my daughter *Gala* and my son *Faisal*, and I would like to give them this work to express my love for them and to motivate them for success.

Thank you all!
Entisar Alkayal, November 2017

List of Abbreviations

ACO	Ant Colony Algorithms
ASPPSO	Asynchronous Parallel Particle Swarm Optimization
CPU	Central Processing Unit
CSA	Cuckoo Search Algorithm
DTC	Data Transfer Cost
DTT	Data Transfer Time
EC2	Elastic Compute Cloud
ECT	Expected Completed time
EDF	Earliest Deadline First
FCFS	First Come First Serve
FF	First Fit
FFD	First Fit Decreasing
GA	Genetic Algorithm
IaaS	Instruction as a Service
MBFD	Modified Best Fit Decreasing
MI	Machine Instruction
MIPS	Million Instruction Per Second
MOO	Multi-objective Optimization
MOPSO	Multi-objective Particle Swarm Optimization
MaOO	Many Objective Optimization
MaOPSO	Many-Objective Particle Swarm Optimization
MCT	Minimum Completion Time

MET	Minimum Execution Time
Max-min	Maximum-Minimum Completion Time
Min-min	Minimum-Minimum Completion Time
NIST	National Institute of Standards and Technology
NP	Non-Polynomial
PaaS	Platform as a Service
PSO	Particle Swarm Optimization
PPSO	Parallel Particle Swarm Optimization
QoS	Quality of Service
RAM	Random Access Memory
SaaS	Software as a Service
SLA	Service Level Agreement
SI	Swarm Intelligence
SPPSO	Synchronous Parallel Particle Swarm Optimization
SPV	Small Position Value
TEC	Task Execution Cost
TET	Task Execution Time
TPC	Task Processing Cost
VMC	Virtual Machine Capacity
VM	Virtual Machine

Chapter 1

Introduction

This chapter introduces the key problems addressed in this research: resource allocation in cloud computing and motivations for optimized solutions. An overview of this research is presented in Section 1.1, which includes brief definitions of cloud computing, benefits and structure layers. The main challenges regarding resource allocation are discussed along with general issues in cloud computing. The motivations and challenges regarding optimizing resource allocation are then addressed in Section 1.2. Section 1.3 focuses on the main contributions and objectives of the research. The last section provides a summary of the outline of the thesis.

1.1 Resource Allocation in Cloud Computing

In this section, an overview of the main concepts pertinent to cloud computing will be presented including its definition, structures and models (see section 1.1.1). In addition, Section 1.1.2 discusses the definition of resource allocation while Section 1.1.3 discusses the main issues related to resource allocation in cloud computing environments. An overview of methods for optimizing resource allocation in cloud computing along with the techniques for applying these will be presented in Section 1.1.4.

1.1.1 Cloud Computing Overview

Cloud computing is not an especially new concept; it has long been associated with other distributed systems such as grid computing, utility computing, and cluster computing (Foster et al. (2008)). It is also frequently associated with virtualized infrastructure or hardware on demand, IT outsourcing, platforms and software services,

and other IT industry technologies. Cloud computing should not be viewed as a new technology; it is a combination of many technologies and paradigms that communicate with each other to form a cloud model, as shown in Figure 1.1.

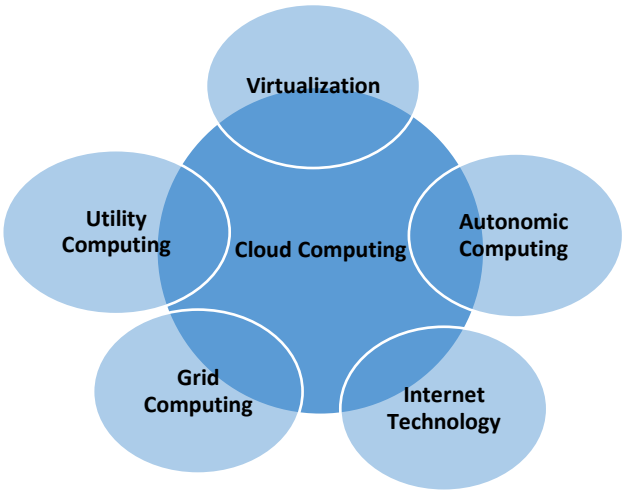

Figure 1.1: Cloud Computing Technologies.

Cloud computing is defined in a variety of ways, the most general of which, incorporating all cloud technologies and characteristics, is presented by the National Institute of Standards and Technology (NIST). NIST define the cloud computing environment as: " a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"(Mell and Grance (2011)).

Cloud service models are divided into three layers, based on the models of services providers supply to consumers. These include Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) layer, each of which is defined as follows (Zhang et al. (2010)):

- **IaaS:** This layer comprises the infrastructure of cloud environment that includes data center resources providing infrastructure services for consumers. The capability provided to the consumer is that of renting these resources to run software. Two examples of this layer are Amazon Elastic Compute Cloud

(Amazon EC2) and Amazon Simple Storage Service (Amazon S3) (Buyya et al. (2013)).

- **PaaS:** Middleware provides a runtime environment that allows developers to create applications and run them in the infrastructure provided by the provider. It responsible for creating applications and frameworks by supporting programming languages and tools hosted in the cloud.

- **SaaS:** This layer is responsible for delivering services to the consumers. SaaS provides full services to the customer rather than requiring customers to install software on their computer.

In this research, the focus is on the IaaS layer of cloud computing because managing this layer is an efficient way to improve the performance of all systems. Cloud computing has several characteristics that provide benefits for both cloud service consumers and cloud service providers. Its main characteristics are on-demand access, scalability, pay-per-use, power efficiency, reliability and virtualizing resources (Buyya et al. (2013)). Most notably, on-demand access means that the resources are available to the customers when they are needed and are in line with their requirements. Scalability to meet consumers' needs refers to the ability of the system to continue working as the number of consumers changes. A scalability feature enables cloud computing to scale resources up or down based on consumers' requirements. Pay-per-use in cloud computing means that the consumer only pays for the resources they use. Power efficiency refers to the strategies that can be applied to use energy more efficiently and reduce consumption. Reliability means that the consumer will be provided with uninterrupted services and resources in line with the agreed Quality of Services (QoS). These services include virtualization, which allows multiple instances of operating systems to run in parallel on a single physical machine (host), and thus defines the concept of virtual machines (VMs) (Ahmad et al. (2015)). This functionality is achieved through the creation of multiple VMs that host different operating systems, thus providing flexible cloud applications.

The NIST definition lists four deployment structures that describe models for delivering cloud services to consumers (Mell and Grance (2011)). These configuration models

vary in management complexity, security implications and their associated costs (Zhang et al. (2010)). The details of these models are as follows:

- **Private**: In a private cloud, one organization manages and maintains the resources. A drawback of this approach is that the benefits of multitenancy, the economy of scale and associated cost savings, are not realized. This deployment model is preferred when data privacy and security are of paramount importance.
- **Public**: In a public cloud, resources are provided to the public and a pay-per-use policy is implemented. The resources are therefore managed and monitored by an external provider. This deployment model is chosen when there are large numbers of users. Organizations can therefore use public clouds to reduce the cost of resources.
- **Hybrid**: A hybrid cloud combines both private and public clouds and is managed and controlled by one organization. This deployment model provides the benefits of enhanced scalability and reduced cost offered by public clouds, along with the security provided by private clouds, thus facilitating the deployment of certain sensitive applications internally.
- **Community**: In a Community cloud, several organizations share infrastructure to implement the same terms of service as well as access policies.

This work focuses on the private cloud model because only one provider will be used to manage and control the distributed data centers. This model is chosen because it is secure and is preferable for use in small organizations such as universities and institutions. The model can be extended to apply in public cloud and adding some procedures for authentication and security.

1.1.2 Resource Allocation Overview

Resource allocation has been an issue of concern for many areas of computing, including operating systems, grid computing, and data center management (Anuradha and Sumathi (2014)). Regarding cloud computing, resource allocation describes the process of mapping available resources to cloud services over the Internet. Specifically, the term resource allocation in a cloud context is defined as the process of finding hosts in the infrastructure of cloud providers to run the applications for consumers in a way

that utilizes resources efficiently based on predefined goals (Jayanthi et al. (2014)). It therefore describes any mechanism that aims to guarantee the requirements of applications are correctly met by the provider's infrastructure (Singh and Chana (2016)).

Data centers in the infrastructure of cloud computing are designed with the capability of applying virtualization. Virtualization is a mechanism for dividing computational resources into multiple isolated executional components known as Virtual Machines (Barham et al. (2003)). Virtualization provides several advantages such as the flexibility to configure several virtual machines on the same host. It also facilitates the dynamic initiation and termination of VMs on a host based on the task requirements and the hosts' specifications. The number of CPU cores in each host determines the number of VMs each host can run (in this research it is assumed that each VM needs only one core). In most of the methods for allocating resources, the tasks are mapped to the virtual machines before they have been assigned to hosts. Therefore, most resource allocation methods are developed in such a way that two consecutive levels are involved (see Figure 1.2) (Huang et al. (2013)). In the first level, tasks are assigned to the appropriate virtual machine based on their requirements, while in the second level the virtual machines are scheduled to appropriate physical machines. This research will aim to improve resource allocation in both levels (Bagul Dhanashri and Toris Divya (2017)).

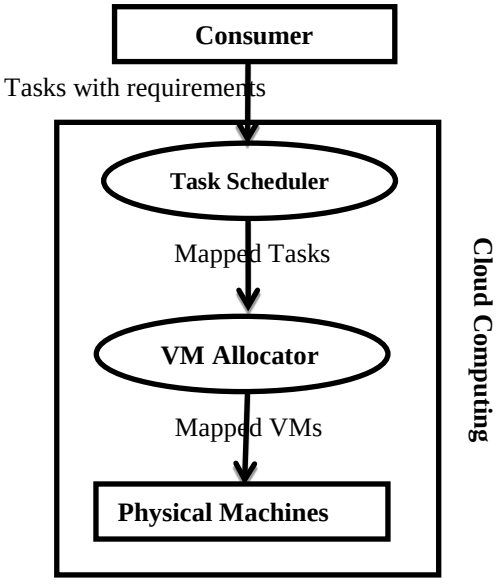

Figure 1.2: Resource Allocation Levels in Cloud Computing.

Specifically, the task scheduling problem in the first level can be defined as the process of searching for the optimal mapping of the set of tasks over the available set of virtual resources to satisfy predefined specific objectives. In the second level, virtual machine allocation refers to the process of selecting which virtual machine should be mapped to the given set of physical machines in the data center. The virtual resource allocation process involves dynamically creating and destroying virtual machines from any resource, without affecting the execution of the application (Khanna et al. (2006)). Allocating resources at the virtual machine level therefore offers many different benefits such as flexibility in allocating and migrating virtual resources. The flexibility that has been achieved using virtual machines encompasses flexibility in determining the location of hosts and dealing with resources where the type of operating system or the hardware specifications are known. Furthermore, virtual machine allocation provides the flexibility to migrate a virtual machine from one host to another. Using VM migration in this way maximizes resource utilization, improves performance, and reduces power consumption (Ahmad et al. (2015)). However, cloud providers and cloud consumers have different and conflicting requirements and objectives. Moreover, the resources in the cloud infrastructure dynamically change in terms of load and availability, making resource allocation in cloud computing a complex problem. The effective and efficient allocation of resources is one of the key essential requirements in cloud computing environments, and thus improving resource allocation is now a central concern.

However, resource allocation algorithms need to be improved to cope with the elasticity, scalability, increasing resources, and cost in the cloud environment (Madni et al. (2016)). Traditional algorithms are inadequate for allocating resource in cloud computing as the cloud resources are based on virtualization technology. Optimizing resource allocation algorithms in cloud computing has therefore attracted the attention of several researchers. The challenges for managing resources in cloud computing revolve around heterogeneity in hardware capabilities, workload estimation, and the cloud consumer's requirements regarding QoS (Madni et al. (2016)). These will be discussed in more detail in the following sub-sections.

1.1.3 **Resource Allocation issues in Cloud Environments**

There are many issues associated with resource allocation in cloud computing environments including QoS, power consumption, VM migration, provider profits, utilization cost and multi-agent systems. In this section, an overview of these issues and their relationship to the proposed research will be presented and discussed.

The QoS and Service Level Agreement (SLA) will be addressed first because it is extremely important in the management of cloud computing and providing services. QoS represents the levels of execution, dependability, and accessibility offered within services (Zheng (2014)). In the context of cloud computing, QoS is therefore a key concern for cloud consumers and providers. Negotiation between providers and consumers should thus be conducted to find agreement in terms of an SLA, which specifies the QoS attributes. Based on the SLA, the providers should guarantee the requirements and satisfy the QoS. However, if the provider cannot provide the required quality, they are penalized. This research study will therefore show how to improve the negotiation process between cloud providers and consumers by using optimization techniques to reach an agreement within a short time.

The second issue will be addressed is the power consumption, which is defined as the rate at which a system will perform work, while energy can be defined as the amount of work done in a certain time (Zhu et al. (2017)). The management of power consumption in cloud data centers has led to several improvements in energy efficiency (Bohra et al. (2010)). In cloud computing, there is a need to design resource allocation algorithms to reduce the total power consumption of the system (Nguyen et al. (2013)). Specifically, cloud computing involves techniques such as the virtualization of computing resources, which can be used to improve the efficiency of the power consumption in cloud environments (Buyya et al. (2010)).

Allocation based on the need for an efficient use of power is becoming increasingly important in cloud computing environments. Moreover, power management is urgently needed due to the increasing demands of computing power and the consumption of power in data center cooling resources. However, the cost of power is an important factor for any provider because a reduction in power consumption leads to a reduction in the cost of cloud infrastructure (Ahmad et al. (2015)). Currently, the main aim of

VM allocation based on power techniques is to map VMs onto a smaller number of hosts. The hosts can be utilized to maximize efficiency and the idle resources, depending on the load conditions, can be hibernated or shutdown to save power**.** In this research, power consumption will be considered in VM allocation and migration simultaneously with QoS performance and utilization of resources.

In terms of virtual machine migration, this provides advantages in the cloud by load balancing across data centers. Detecting over-loaded resources and under-loaded resources effectively is a key concern when applying VM migration from one resource to another. The execution of the migration process therefore needs to be implemented effectively to prevent a negative impact on performance. In this research, VM migration will be used to balance the load and utilize the resources in an efficient way. This research will focus on optimizing the method of detecting the status of resources by applying clustering techniques because they provide accurate and adaptive results.

In terms of profitability, the goal in many cases is to maximize the profit earned by cloud data centers, while taking into consideration the QoS. Before allocating tasks to the VMs, the cost of executing that task will be calculated using a profit model. In this model, the profit from the provider's resources depends on three major features: the execution cost of the physical resources, the utility cost of power consumption, and the penalty cost the provider should pay if there is a violation of the agreement. Several studies have examined how to maximize profit, for example by minimizing the power of active hosts (Wang et al. (2013)), rescheduling VMs (Ahmad et al. (2015)), and exploiting the penalty when exceeding the deadline, which is a predefined cost set by the provider (Lee et al. (2012)). In the latter study, Lee et al. devised two profit-based algorithms known as Max Utilization (MaxUtil) and Max Profit (MaxProfit). The former focuses on utilization and indirectly reduces the cost of running resources as well as ensuring efficient utilization. The latter focuses on profit by selecting a task with the earliest start in the queue, which is then, maintained using the MaxUtil technique. In the research reported in this thesis the focus will be on maximizing profit by combining the methods of minimizing the number of VMs and rescheduling VMs to reduce the number of running hosts. This methodology is akin to the MaxUtil algorithm, as the focus is on maximizing utilization, which increases profit indirectly

by reducing the cost of execution. However, the method proposed in this research will improve both utilization and performance in term of waiting time and throughput.

Chaisiri et al. (2009) developed a multi-objective mechanism for scheduling applications that takes into consideration various cost constraints and the availability of resources. In this research, the focus will be on maximizing utilization by using VM migration, and without using any specific techniques to optimize profit. Profits can be indirectly improved by maximizing the utilization and reducing the number of SLA violations as these involve penalties.

In terms of cost efficiency, the common method for determining cost in cloud computing is to use a pay-per-use model to ensure minimal costs and payment purely for the resources used (Pietri and Sakellariou (2016)). To maintain this feature, cloud management algorithms are responsible for offering resources that can complete the consumer's request on time, within their budget, and at the lowest cost that can be offered. This incorporates resources such as processing cost, memory cost, storage cost, and data transfer cost. Cost aware algorithms are therefore required that are cost efficient and can provide the best system performance by improving both utilization and power consumption. These types of algorithm are described as multi-objective algorithms, because many objectives need to be taken into consideration to guarantee optimal performance. In this thesis, the cost of using resources will be considered in terms of processing, memory, storage, and transfer data. The proposed model is based on distributed data centers; therefore, bandwidth cost is an important factor to consider when allocating resources for consumers.

Finally, multiple agents are considered in resource allocation. An agent is a software entity that acts on behalf of another entity to perform a specific task satisfy specific goals (Wooldridge (2009)). Agent systems are self-contained software programs with domain knowledge that can work with a degree of independence and follow specific actions to satisfy predefined goals (Jennings and Wooldridge, 1995). Agents include a set of features (Wooldridge (2009)), the main ones of which are autonomy, pro-activity, re-activity, cooperation, negotiation and learning. The concept of autonomous behavior means that agents can be satisfied their objectives are on the behalf of users. Additionally, agents can pursue their own individual goals, including taking decisions

based on internal and external events. Moreover, agents communicate with other agents to exchange information, receive instructions, give responses, negotiate, and cooperate to fulfill their own goals. Finally, agents are able to improve the decision making process when interacting with the external environment.

Multiple agents are a set of agents that interact together to resolve a complex problem using their combined knowledge (Taranti et al. (2011)). Multiple agents are conceived as a distributed computing model whereby they interact to exchange information and execute complex tasks that require dynamic, intelligence and adaptive interaction (Wooldridge (2009)).

The design of agents and their inherent features means they are well suited for creating complex systems (Jennings (2000)). Using agents can thus simplify the design and implementation of such systems, as it is not necessary to consider all possible links, interactions and states. Instead, agents can be programmed with specific behaviors, enabling them to deal with unknown states and interactions as they occur. Furthermore, agents can be used individually, either by assigning each one of them to work on a specific aspect of the problem or by cooperating to solve a problem in a distributed fashion.

Software agents can be used to model intelligence in distributed systems such as cloud environments in order to improve adaptability, flexiblility, and display autonomic charactaristics. Specifically, agents can be applied in resource allocation, providing services and in executing large-scale distributed services (Banerjee and Hecker (2017)). They can therefore be used to provide intelligent monitoring capabilities and management services. Additionally, agents can be applied to ensure the efficient use of energy in cloud computing infrastructures. They can also make the cloud smarter in its interaction with users and more efficient in providing services (Al-Ayyoub et al. (2015)). Agent-based management can be applied in cloud computing at all three layers (Al-Ayyoub et al. (2015)). In the IaaS layer, agents can be applied to manage and provide intelligent allocation of resources to consumers. In the PaaS layer, agents can be applied in the deployment process and in the execution of programming environments to implement the applications. In addition, agents are used in the PaaS to implement the management functionalities of cloud environments. Finally, in the SaaS

layer, agents can be utilized to optimize the services provided to the consumers. Furthermore, they can be used as interfaces for managing the underlying hardware/software infrastructure to ensure efficient utilization and that the QoS is satisfied. In this thesis, the focus will be on how to use agents to manage and allocate resources in the IaaS layer and, at the same time, monitor and negotiate with consumers in the PaaS layer.

Agent organizations also provide a framework of constraints and expectations regarding the agents' behavior by focusing on the decision-making and actions of specific agents (Wooldridge (2009)). The agents in this research will be designed to manage consumers' tasks, allocate VMs, balance the load of VMs, monitor the SLA agreements and utilize the resources.

1.1.4 Optimization Algorithms in Resource Allocation

Optimization is a rapidly expanding field of research that plays a vital role in addressing real world problems. As discussed previously, resource allocation is a key problem that needs to be optimized regarding the issues arising from different objectives. In this section, optimization algorithms in resource allocation will be discussed in detail. Optimization is defined as the process of finding the best possible solution amongst all those available in the search space (Gendreau and Potvin (2005)). Specifically, optimization algorithms are responsible for modeling and evaluating solutions based on an objective function and then applying search methods to find the best solution (Yang (2010)). An objective function can be computed based on single or multi-objective depending on how many objectives are involved in the evaluation methodologies. In multi-objective optimization, the best value for each objective is included in the evaluation of objective functions. Multi-objective optimization aims to seek in the search space, where each objective is represented by a vector of decision variables to satisfy specific constraints (Marler and Arora (2004)).

Optimization algorithms are needed in most modern applications due to the limited number of resources available. Furthermore, time is a significant concern because most applications work in real-time; algorithms therefore need to reduce the waiting time and allocate resources quickly. Thus, solutions need to be found in order to manage and

allocate the resources efficiently and under predefined constraints to fulfil consumers' requests and respond to providers' objectives.

However, some complex multidimensional problems cannot be solved using deterministic optimization techniques (which follow the same steps at each iteration and provide the same results), but can be solved using heuristic algorithms (Blum and Roli (2003)). Such algorithms suggest some approximations to the solution for optimization problems with low time complexity (Yang (2010)). Nowadays, meta-heuristic optimization algorithms are widely used in solving Non-Polynomial (NP) hard problems (Yang (2010)). Meta-heuristic algorithms in particular are used to find near optimal solutions in a reasonable amount of computation time (Masdari et al. (2016)). This is because meta-heuristic algorithms provide better quality results than deterministic algorithms and can find solutions faster than traditional exhaustive algorithms (Madni et al. (2016)). To achieve this, meta-heuristic algorithms use iterative strategies with randomness. Thus, meta-heuristic techniques are used in many lines of research to address problems that require fast results using reasonable solutions. For example, in terms of the resource allocation problem, they have been shown to produce better scheduling results than traditional scheduling algorithms (see sections 2.4.3 and 2.4.4 for more detail). For these reasons, this thesis will focus on meta-heuristic approaches.

In particular, we will focus on Swarm Intelligence (SI) as this is one of the more common meta-heuristic techniques. SI is a meta-heuristic that has been developed to solve optimization problems by simulating the behavior of social insects (Blum and Li (2008)). SI algorithms are used in complex distributed systems because there is no need for a central control structure. In addition, SI algorithms provide an elastic and flexible resource allocation as they add or remove resources without influencing the overall structure (Blum and Li (2008)). SI includes several algorithms such as Ant Colony Algorithms (ACO) and Particle Swarm Optimization (PSO) (Madni et al. (2016)).

With the development of cloud computing, there is now a pressing need to study and improve both the methods and the algorithms of resource allocation (as argued in Section 1.1.2). Resource allocation plays a key role in cloud computing systems because it directly affects the performance of the overall system. An efficient resource

allocation mechanism can meet consumers' requirements, and at the same time improve the utilization and profit for the provider. Given the features of cloud computing, such as flexibility, virtualization and so on, two levels of allocation mechanisms can be applied in cloud computing. The first is the task scheduling level, which schedules the tasks over available VMs, while the second level involves distributing the VMs over the host. In cloud computing, resource allocation involves the process of assigning available resources to the cloud applications that need those (Buyya et al. (2013). Resource allocation in the IaaS in cloud computing has attracted considerable attention in the research literature. In the IaaS layer, resource allocation uses VMs to execute consumers' requests. Several meta-heuristic algorithms have been developed by research scholars to optimize the allocation of resources in this level.

Resource allocation in cloud environments is, however, a complex task due to the geographical distribution of resources with varying load conditions, different user requirements and price models (Jayanthi et al. (2014)). Depending on the main objectives, resource allocation techniques in the cloud are classified as SLA and QoS, cost optimization, load balancing, resource utilization, and energy efficiency metrics (as shown in Figure 1.3).

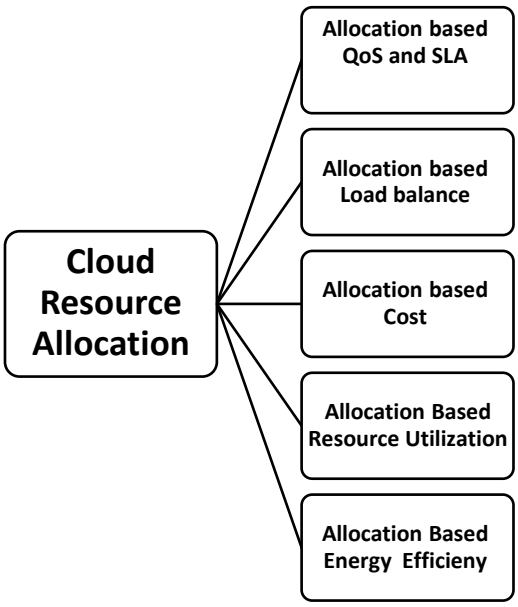

Figure 1.3: Resource Allocation Optimization Objectives.

Importantly, cloud providers need optimization algorithms to improve management and monitoring methods and maximize resource utilization and profit (Sahal et al. (2013); Omara et al., (2014)). In addition, SLA in terms of QoS parameters is another issue that needs to be considered when developing efficient resource allocation. However, some researchers have developed resource allocation to satisfy one or more optimization objectives, as will be discussed in section 2.3.2.

Resource allocation in cloud computing is therefore a field where many problems need to be addressed to optimize and increase the performance of cloud systems. In this thesis, three significant problems arising when allocating cloud computing resources will be addressed. **The first** is that of optimizing the SLA negotiation between cloud consumers and providers, which requires completing many processes to reach an agreement quickly and then monitoring this agreement to reduce the number of SLA violations. **The second** problem concerns scheduling tasks in virtual machines to select an appropriate VM that will run tasks based on several predefined objectives. **The third** problem involves optimizing the VM allocation and migration algorithms to utilize cloud resources efficiently, balance the load and reduce power consumption, which improve QoS performance and increase profits for providers.

As cloud computing resources become more distributed, QoS will become increasingly important for cloud consumers and cloud providers in satisfying several and conflicting objectives (Zheng (2014)). Therefore, to reach an SLA, a negotiation mechanism should be implemented. SLA negotiation is important in guaranteeing the performance of the cloud and developing trust between cloud consumers and cloud providers. The SLA negotiation processes need to be optimized and conducted within a short space of time by reducing the number of rounds involved in the negotiation. Additionally, automated negotiation needs to be developed to reduce communication between parties and thus reduce the time spent on negotiation. Furthermore, preventing SLA violations will avoid any costly penalties the providers need to pay to guarantee the QoS. During the negotiation process, the parties exchange information to indicate their negotiation goals and requirements. If multiple objectives are involved, this becomes a complicated problem because many factors need to be considered during the negotiation process. In this research, the aim is to improve the SLA negotiation mechanisms by reducing negotiation time between consumers and providers.

The second problem is that of task scheduling which is a critical problem in cloud computing environments; this is related to the efficiency of cloud computing facilities, which is considered an NP-hard problem (Guo et al. (2012)). Many researchers have explored ways of finding optimal mapping between tasks and virtual machines to improve scheduling algorithms and satisfy several objectives. However, there are many conflicting objectives of the consumers and providers need to be addressed. Different experiments show that, although finding an optimum solution is almost impossible, a sub-optimal solution using meta-heuristic algorithms can be reached. In this research, the task scheduling algorithm deals with five objectives, which require an optimized method to evaluate them given the time constraints. Thus, a modified ranking strategy will be developed to enhance the scheduling algorithm and reduce the mapping time to deal with many-objective where more than three objectives are involved (as will be described in Section 2.2.2.2).

The third problem is that of allocating virtual machines to hosts and migrating them in a way that improves utilization and reduces the power consumption. Different strategies for allocating resources (virtual or physical) result in different levels of efficiency (Chaisiri et al. (2009)). This thesis will explore how to use clustering technologies to improve the method for detecting the status of hosts in the cloud data centers, and thus schedule and migrate VMs to provide dynamic and effective methods of balancing the load among hosts. The aims of optimizing VM allocation are therefore to reduce the migration time of VMs, improve the utilization of the resources, reduce power consumption, and balance the load among hosts.

In cloud computing, it is important to develop algorithms for allocating resources quickly to reduce waiting time and the number of missed deadlines of tasks. Meta-heuristic methods have occupied a strong position in research on optimizing cloud computing over the past few years due to their effectiveness in solving some of the most important problems that arise. They have been shown to provide immediate and quick solutions to these problems compared to deterministic algorithms. Several intelligent techniques have been developed to improve resource allocation in cloud systems such as Genetic Algorithms (GA) (Singh and Chana (2016)), PSO (Salman et al. (2002)), and ACO (Xue et al. (2014)). In this thesis, the focus will be on applying PSO

algorithms to improve resource allocation, and then comparing results with GA and ACO algorithms.

Specifically, a PSO has proved to be effective in finding near-optimal solutions by simulating the movement of a flock of birds when they search for food (Salman et al. (2002)). In addition to the benefits of most meta-heuristic algorithms, such as flexibility and acceptable calculations, PSO has additional advantages such as easy implementation and consistent performance. In the PSO algorithm, a search for the solution is carried out at low computational cost for a wide range of complex applications including combinatorial optimization problems, finding optimal routes, scheduling, structural optimization, image analysis, data mining, bioinformatics, and finance and business (Eberhart and Shi (2000); Alkayal et al. (2016)).

The implementation of the PSO procedure is straightforward and generally requires relatively few lines of code because it based on simple operations and takes a short time compared to other algorithms such as ACO and Genetic Algorithms (Eberhart and Kennedy (1995)). It has only one operation to update velocity and position to coordinate and control the movements of particles. The calculation in PSO is simple because no overlapping and mutation calculations are involved, unlike Genetic algorithms. Therefore, PSO takes less time to find solutions than Genetic algorithms and time is a key factor in most applications (Mirzayi and Rafe (2013)). Thus, PSO is more popular than other SI algorithms in solving problems that require quick search results. The current research therefore focuses on PSO and will study the benefits of applying variants of PSO algorithms in cloud computing environments.

Specifically, PSO is a meta-heuristic algorithm often used to optimize cloud computing (Tsai and Rodrigues (2014)). Such algorithms have been developed to solve optimization problems when allocating resources in cloud computing infrastructure to satisfy certain goals (Feng et al. (2012)). Several modifications to the original concept of the PSO algorithm have been made to improve the standard PSO and ensure it can cope with the requirements of cloud environments. Given this, three types of PSO will be focused upon in this research: Parallel PSO, multi-objective PSO, and Cluster based PSO. In the next section, the reasons for using each type of PSO to optimize the problem addressed in this research will be discussed.

1.2 Challenges in Optimizing Resource Allocation

The PSO algorithm can be applied to solve different types of problems in several fields such as multi-objective optimization (Cagnina et al. (2005)), data clustering (Neshat et al. (2012)), and scheduling (Salman et al. (2002)). There are several reasons for using PSO to improve resource allocation in cloud computing in the task scheduling and virtual machine allocation levels (Madni et al. (2016)). In this section, the motivation for applying the PSO algorithm to improve SLA negotiation (see Section 1.2.1), task scheduling (see Section 1.2.2), and clustering techniques in cloud computing (in Section 1.2.3) will be discussed in detail.

1.2.1 Service Level Agreement Negotiation

Cloud providers and cloud consumers have different and conflicting objectives, so they need to negotiate to reach a certain agreement. Unfortunately, in cloud computing the SLA negotiation is a difficult process because resources are very diversified, distributed and managed by different entities **(**Dastjerdi and Buyya (2012)). To cope with these difficulties, a multiple agent approach to SLA negotiation and management in cloud environments is often used (Chen et al. (2014)). Reducing the waiting time is another requirement for cloud consumers, so it is essential to reach an agreement quickly to reduce mapping time. Thus, concentrating on enhancing the negotiation process can benefit both parties. It can reduce the waiting time for consumers and increase the number of completed tasks, which will increase both the system throughput and the provider's profits.

In this research, the SLA negotiation will be considered at the IaaS level, which includes the virtual and physical resources. Negotiation at this level is more significant in terms of enhancing resource allocation. From another perspective, SLA negotiation at this level is more complex and requires greater optimization because there are multitude of distributed data centers with many resources in the infrastructure. The main requirements are to speed up the process of negotiation to reduce waiting time and decrease the number of negotiation steps to save time for consumers and providers.

The provider will also benefit from an increase in the number of successful tasks, which expands the system throughput.

Parallel computing can also be applied to speed up the process of the algorithm, as this executes the algorithm over several resources simultaneously (Grama (2003)). Recently, parallel computing has been used extensively in cloud computing field to develop tools enabling efficient solutions for resource allocation problems to be found (Warneke and Kao (2011)). This is because parallel computing reduces the processing time needed for performing complex computational tasks. The goal of parallelization is therefore to reduce the time spent on computation and to solve a problem by using many nodes simultaneously and dividing the work between them (Grama, (2003)).

In optimization problems, processing large amounts of data using individual function evaluations may take a considerable amount of time. Therefore, some of the meta-heuristic algorithms are parallelized to improve their speed. PSO is one of these, as it has been parallelized (Parallel PSO) to reduce search times. This research will therefore use Parallel PSO to optimize the negotiation process and reduce communication overheads, thus enhancing the speed of negotiation processes. The purpose of using meta-heuristic optimization in the SLA negotiation in cloud computing is to reduce the cost and time complexity of the negotiation process. PSO is a simple and effective algorithm, but it may be time consuming to use in a large search space (Koh et al. (2006)). The performance of sequential PSO is negatively affected when applied to complex optimization problems, which is a strong motivation for the development of parallel optimization (Chang et al. (2005)).

1.2.2 Task Scheduling in Cloud Computing

Task scheduling is one of the key issues in cloud computing environments and thus has garnered attention from several researchers (Masdari et al. (2016)). In cloud systems, task scheduling algorithms aim to spread the workload among the computing nodes to maximize utilization while minimizing the overall task execution time. Moreover, optimizing scheduling in cloud environments taking both performance and resource utilization into consideration is a significant goal (Al-Olimat et al. (2014)). In this

context, performance is viewed in terms of throughput, which is the total number of tasks executed to completion in a unit of time. The utilization of resources represents the overall utilized capacity of the cloud resources (Al-Olimat et al. (2014)). However, utilization and performance have an inverse relationship, i.e. increasing utilization may decrease the performance in terms of waiting time. Therefore, to achieve maximum utilization, resources should be allocated efficiently and simultaneously compromise conflicting objectives for consumers and providers. For instance, allocation will decrease the waiting time for consumers because they pay per time spent on the cloud, and it minimizes the costs of reserving resources while decreasing the utilization time.

To address the problem of evaluating and satisfying multiple and conflicting objectives, several researchers have developed techniques using multi-objective optimization (MOO) which handles two or three objectives (Reyes-Sierra and Coello (2006)). Many-objective optimization methods are developed to deal with more than three objectives. However as argued previously, there are often multiple and conflicting objectives that need to be satisfied in cloud systems.

Optimization problems with more than three objectives are a very attractive topic for researchers due to their widespread applicability. Previous research efforts in the optimization field have resulted in the development of algorithms that are able to achieve good results by handling problems with two or three conflicting objectives (i.e. multi-objective optimization). However, these techniques do not present the same quality when the number of objectives increases to greater than three (Li et al. (2015)). Therefore, several attempts have been made to reduce the computational requirement of evaluating many-objective optimization. Currently, meta-heuristic algorithms have attracted the most research attention in this respect (Figueiredo et al. (2016)). Thus, the current research will tackle the problem by using many-objective PSO algorithms to improve task scheduling problems.

Resource allocation problems have many objectives that need to handle and thus evaluate to adapt with an increased number of objectives. Thus, in the current research, the Pareto-optimal method will not be used, as the aim is to combine the ranking method with a weighted sum approach to handle many objectives in a short space of time and a simple process. These objectives include improving performance by reducing waiting

time, the cost of execution, and maximizing throughput, which leads to an increase in profits.

1.2.3 Virtual Machine Allocation

The third major goal of this work is to increase resource utilization and reduce power consumption by improving VM allocation and migration at the level of VMs. Specifically, VM allocation can be carried out in two phases: VM scheduling and VM migration. In this research, they will be dealt with as an integrated process to utilize the resources and reduce the number of migrations by selecting the best mapping. Specifically, the interaction between VM scheduling and VM migration is developed to take into consideration the tradeoffs between power consumption, QoS performance and resource utilization.

In most VM scheduling algorithms, hosts are divided into two sets: a set that meets some of the criteria and a set that does not fulfil the criteria. Consequently, the set of hosts that meet the criteria is ordered to start VM scheduling with the first hosts on the list, and continues until all VMs have been placed or until the set of qualified hosts is exhausted. Based on different criteria, several algorithms have been developed to allocate VMs to hosts. To determine the qualified hosts, various allocation criteria are considered such as host load, execution cost and CPU processing speed.

In particular, VM migration, for instance, is used to improve resource utilization and reduce power consumption (Masdari et al. (2016)). The process of determining when the machine is over-loaded or under-loaded is a critical consideration when allocating resources, different aspects of which have been addressed by many researchers such as (Beloglazov and Buyya (2010); Lin et al. (2011)). In general, the methods for classifying hosts inside data centers can be divided into two main approaches. In the first approach, the threshold level is used to detect the over-loaded hosts. These thresholds can be static or dynamic (Beloglazov and Buyya (2012)). In most cases, dynamic thresholds are used because static thresholds do not reflect the current change in the system load and thus unbalance the load. Dynamic thresholds are used to determine the machines' load status; thus, a threshold is the value related to the system

state that determines changes in system behavior (e.g. system load, waiting queue length, and storage size) (Huang et al. (2013)). VM migration algorithms can be implemented based on the utilization threshold, to allocate VMs and migrate them among hosts. However, defining thresholds with fixed values are unsuitable for an environment with dynamic and unpredictable workloads, where different numbers and types of application may share physical resources (Pietri and Sakellariou (2016)). Using the dynamic threshold is very important in determining and classifying VMs in data centers because the threshold values must be able to adjust according to changes in the system utilization.

In the second approach, VM allocation is treated as a combinational optimization problem with specific constraints such as CPU utilization, power, performance, or a combination of these (Shabeera et al. (2017)). A meta-heuristic optimization algorithm is then executed to solve the problem. Using a meta-heuristic to solve VM allocation and migration provides an effective solution compared to the use of thresholds (Madni et al. (2016)). The research described in this thesis will determine how to use PSO in the clustering of hosts inside data centers to improve the migration process and enhance the load balance techniques. The novelty of this field lies in the process of clustering hosts based on PSO before applying VM migration, which considers the performance factor during the migration process. Thus, VM migration is performed from over-loaded and under-loaded machines simultaneously to reduce power consumption.

1.3 Research Contributions

This thesis addresses how to apply PSO algorithms to optimize resource allocation in cloud computing regarding the negotiation process, scheduling tasks, and clustering hosts. The general aim of optimizing resource allocation is to satisfy the requirements of both cloud providers and consumers. More specifically, improvement in performance is measured by minimizing the waiting time, SLA violations, and power consumption, and by maximizing the utilization of resources, throughput and profits.

The proposed resource allocation model will be developed by improving three modules for allocating resources in cloud computing as shown in Figure 1.4. The objectives of this work can be summarized as follows:

- Improving SLA negotiation based on Parallel PSO to reduce the waiting time for finding the data center that has the most suitable resources for executing tasks and increasing the throughput. In addition, the SLA violation rates will be reduced to increase the profits.
- Enhancing the method of handling many-objective PSO task scheduling in cloud computing by using a modified ranking strategy to reduce the waiting time for scheduling tasks inside each data center and increase the throughput.
- Improving VM allocation based on many-objective PSO to improve resource utilization, power consumption, and QoS performance.
- Optimizing the utilization and balancing the load among cloud hosts by clustering the hosts inside each data center based on PSO and K-means algorithms.

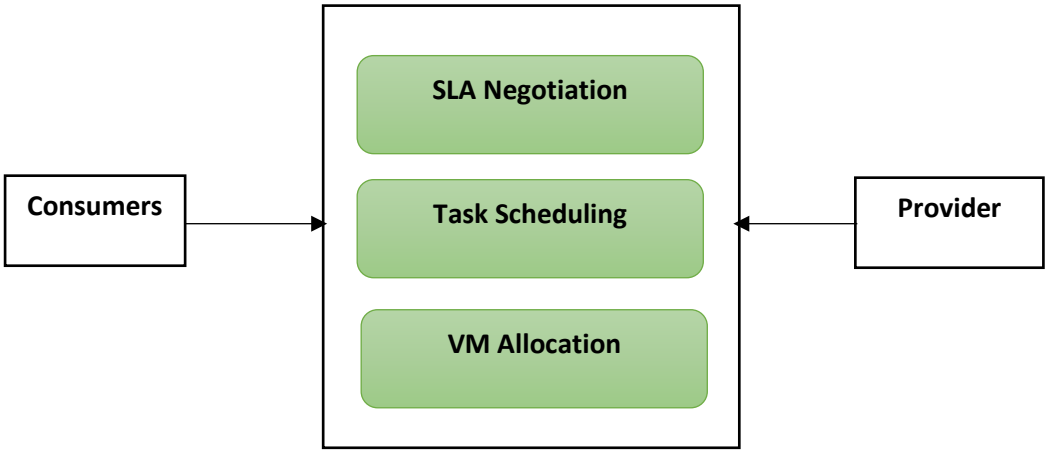

Figure 1.4: Resource Allocation Modules.

Figure 1.5 summarizes the main objectives of each module in the proposed model based on the optimization objectives of resource allocation (see Figure 1.3).

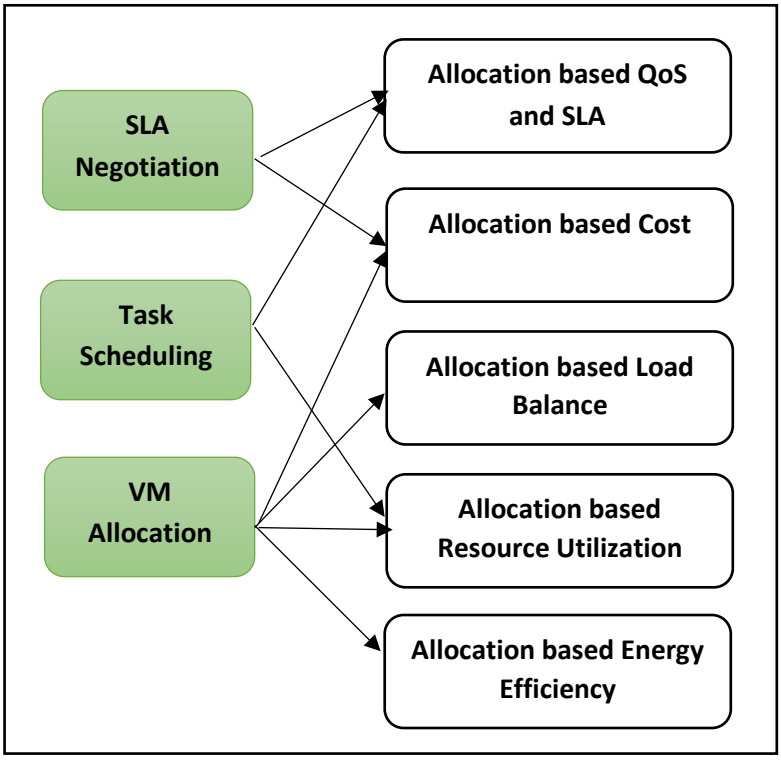

Figure 1.5: Proposed Resource Allocation Objectives.

The main contributions of this research in terms of knowledge can be summarized as follows:

- **In the SLA Negotiation Module:**
  - Designing and developing a multi-objective SLA negotiation based on Parallel PSO algorithms to reduce negotiation time and increase throughput.
  - Developing a SLA Monitor, which monitors the execution of tasks and the utilization of resources to improve performance and reduce the number of SLA violations.
  - The originality of this module lies in applying Parallel PSO in SLA negotiation in cloud computing, an area in which there has been no previous research.

- **In the Task Scheduling Module**:
    - Developing a many-objective PSO (MaOPSO) algorithm to improve real-time task scheduling based on five objectives: Task Execution Time (TET), Task Execution Cost (TEC), the data transfer cost, data transfer time, and VM capacity under deadline constraints.
    - Developing a new modified ranking strategy to evaluate the five objectives as a separate objective in less time than state of the art methods to reduce the waiting time and increase the throughput.
    - The novelty of this approach lies in evaluating a many-objective PSO using a modified ranking strategy and applying it in task scheduling.

- **In the VM allocation Module**:
    - Improving the method of allocating physical resources to satisfy QoS demands in terms of throughput, waiting time and balancing the load by using MaOPSO to utilize the resources efficiently.
    - The originality of this work lies in applying clustering based PSO with K-means in the VM allocation algorithm.
    - Developing a load balancer algorithm using the clustering technique to group the hosts into four classes: over-loaded, high-loaded, under-loaded, and unloaded hosts.
    - Developing a VM migration algorithm to migrate VMs among hosts inside each data center based on the clustering results, thus reducing the power consumption and the imbalance factor between hosts inside each data center.

These contributions have been disseminated in number of academic papers:

- Alkayal, E. S., Jennings, N. R., & Abulkhair, M. F. (2016, November). **Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing**. In *Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on* (pp. 17-24). IEEE.

This paper presented a method of improving the ranking strategy for task scheduling based on multi-objective optimization. Three objectives were evaluated and the results were compared with weighted sum and Pareto set approaches.

- Alkayal, E. S., Abulkhair, M. F., & Jennings, N. R. (2017, September). **Automated Negotiation using Parallel Particle Swarm Optimization for Cloud Computing Applications.** In *Computer and Applications (ICCA), 2017 International Conference on* (pp. 26-35). IEEE.

This paper described the model of applying Parallel PSO to improve SLA negotiation in terms of reducing negotiation time. The results were compared with standard PSO, where the model demonstrated several improvements in terms of performance. This paper is selected from the best papers in the conference to be extended as a chapter of book and will be published in Springer in December of 2017.

- Alkayal, E. S., Jennings, N. R., & Abulkhair, M. F. (2017, November). ***Survey of Task Scheduling in Cloud Computing based on Particle Swarm Optimization***. *The International Conference on Electrical and Computing Technologies and Applications, 2017.*

This paper is accepted to publish in IEEE and it is under the process of publishing. It discusses the previous work in the task scheduling that were applied PSO algorithms. It classified them based on several factors. Finally, it summarized the main points that need to be explored in developing PSO.

1.4 Thesis Structure

The remainder of this thesis is organized as follows:

Chapter 2: details the background to this work in the field of optimization algorithms and PSO. It also introduces and describes the SLA concept, SLA management and the SLA negotiation process. Moreover, it presents an overview of previous work related to resource allocation in cloud computing using PSO to optimize cloud computing environments. Finally, it concludes with a summary of the main points of comparison between this research and previous work in this field.

Chapter 3: describes the general architecture of the proposed resource allocation model and its components. The general design of the model, its objectives and constraints are also discussed in this chapter. The model includes three main algorithms: SLA negotiation based on Parallel PSO algorithm, MaOPSO task scheduling algorithm using ranking strategy, and a VM allocation and migration algorithm using clustering based PSO. Details the design of these algorithms and their implementation are presented in chapters 4, 5 and 6.

Chapter 4: discusses details of the automated SLA negotiation algorithm and its specifications. Additionally, it presents the experimental strategies, evaluation procedures, and evaluation results of applying the Parallel PSO algorithm.

Chapter 5: presents the MaOPSO task scheduling algorithm and how it can be improved using a modified ranking strategy. Additionally, it describes the implementation issues and the experimental procedures. Finally, it discusses the evaluation methodologies, and the results of the algorithm.

Chapter 6: provides details of the VM allocation algorithm and its implementation. In addition, it presents the clustering methodologies used in VM migration based on the PSO and K-means algorithms. The experimental and evaluation processes are then discussed. Finally, it summarizes the results of the algorithm.

Chapter 7: presents the results of the study, its general contributions and then highlights directions for future research. In addition, it offers several practical recommendations based on the findings.

Chapter 2

Related Work

This chapter presents and reviews the background information on essential concepts relevant to the research presented in this thesis. Each section presents a historical overview of the development of each concept up to the present. Section 2.1 discusses meta-heuristic algorithms in general and then focuses on swarm intelligence techniques. Section 2.2 presents an overview of the PSO and reviews applications of PSO in different fields related to this research. Section 2.3 describes the key concepts of SLA negotiation and then reviews previous research in this field. In Section 2.4, a general overview of task scheduling in cloud computing is presented and related work based on PSO is discussed. Section 2.5 describes virtual machine allocation and migration techniques. Finally, Section 2.6 summarizes the main ideas presented in this chapter.

2.1 Meta-Heuristic Optimization Algorithms

This section presents a general overview of meta-heuristic algorithms and then discusses meta-heuristic algorithms related to this study, namely Swarm Intelligence (Section 2.1.2) and heuristic optimization algorithms (Section 2.1.3). Finally, several methods of evaluating multiple objectives will be discussed in Section 2.1.4.

2.1.1 Overview

Optimization algorithms are advanced techniques for solving an optimization problem by determining its optimality. Mathematically, an algorithm is a technique used to produce outputs for a given set of inputs under specific constraints (Blum and Roli

(2003)). In each optimization methodology, there is an objective function, which is used to evaluate objectives under specific constraints. Optimization algorithms work by defining the search space and attempting to maximize or minimize the objective function (Yang (2010)). When an optimization problem is formulated, the algorithm searches the space for optimal solutions using appropriate mathematical techniques. Depending on the objectives, a choice can be made to find the optimal solution, which takes time, or to find a near optimal solution with less time complexity.

For distributed systems, there are many factors involved in selecting an appropriate algorithm and there is no efficient algorithm that can be used for all cases. Instead, there are specific aims and objectives for the optimization algorithm. In general, optimization techniques can be classified into several categories based on different factors. A common form of classification divides the algorithms into deterministic and stochastic algorithms depending on the nature of the algorithm and the method for finding solutions (Yang (2010)). Deterministic algorithms follow repeatable path and variables, while stochastic algorithms are based on the randomness in the path and the variables. For example, in Genetic algorithms, when searching for an optimal solution the population of solutions differs each time because it is based on randomness. Conversely, there is no major difference in the results of stochastic algorithms, although the paths in each population are repeated. Some approaches combine deterministic and stochastic algorithms to benefit from the advantages each provides whilst overcoming the limitations of both.

Stochastic algorithms themselves can be divided into two types: heuristic and meta-heuristic. Heuristic methods find a good optimal solution with low computational cost, but are not guaranteed to find an optimal solution (Madni et al. (2016)). The meta-heuristic algorithms generally perform better than simple heuristics and use randomization with a local search (Yang (2010)). There is no specific definition that determines the difference between heuristic and meta-heuristic algorithms. However, some researchers describe all stochastic algorithms with a randomness property and global search as meta-heuristic (Gendreau and Potvin (2005)). Randomization in meta-heuristic algorithms provides a method for moving from the local space to the global space when applied to global optimization (Blum and Roli (2003)). Meta-heuristic algorithms are used to find near optimal solutions in an acceptable time based on single

or multiple objectives (Gendreau and Potvin (2005)). Compared to deterministic algorithms, meta-heuristic algorithms present better results in terms of quality and computation time (Tsai and Rodrigue (2014)). For these reasons, meta-heuristic algorithms will be used in this research to optimize resource allocation in cloud environments.

Optimization problems, on the other hand, are classified according to the type of variables involved and can be divided into two different categories: combinatorial and continuous optimization problems, based on discrete or continuous variables, respectively. This thesis will focus on discrete optimization problems because resource allocation includes discrete variables that represent the IDs of resources.

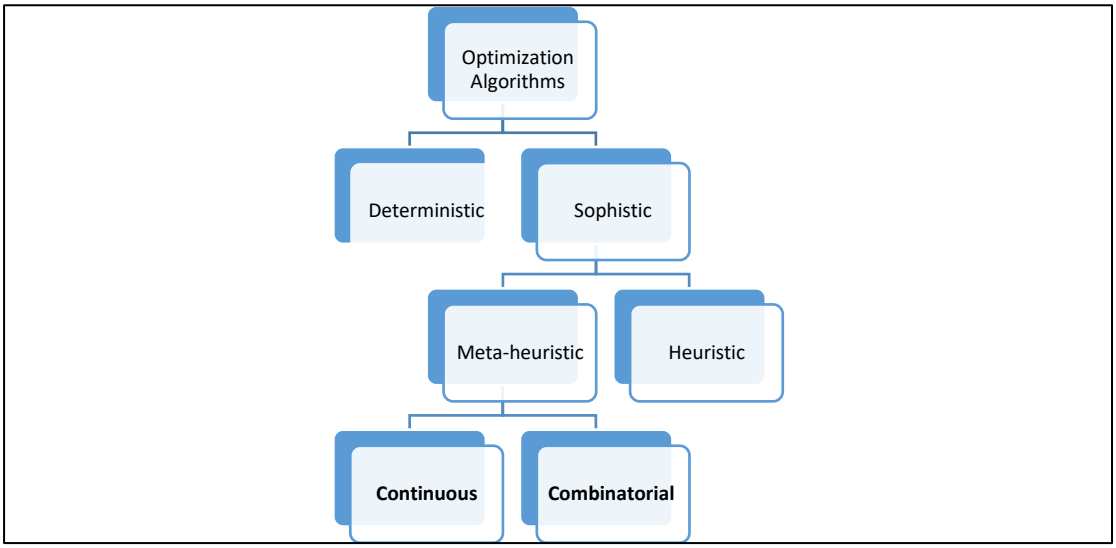

Figure 2.1: Main Classification of Optimization Techniques.

In general, all meta-heuristic algorithms follow the same steps when searching the space to find the optimal solution but with different forms of implementation and detail. The main steps all meta-heuristic algorithms follow are shown in Algorithm 2.1 (Tsai and Rodrigues (2014)). Specifically, each algorithm starts by initializing the solutions and defining the population (Line 1). The next step involves repeating three processes until the termination criteria are met (Lines 2-6). The details of these processes are as follows:

- **Transition**: this process changes the current state or position of the solution(s) to the next state by checking the available solutions. The complexity of this operation depends on the design of the meta-heuristic algorithm.
- **Evaluation**: this process is responsible for evaluating the objective function of the solutions based on the predefined optimization factors.
- **Determination**: this process controls the search by determining the directions and constraints that control the convergence speed and the movement inside the space.

Algorithm 2.1: General Meta-heuristic Algorithm

Start Procedure ( )
1. Initialization (s*)* *// initialize solution set*
2. **while (**the termination criterion is not met) **do** *// do while the condition is valid*
3. v = Transition(s) *// change the state or position of solutions*
4. f = Evaluation (v) *// evaluate the new state based on objectives*
5. s = Determination (v, f) *// Comparing the current with the previous*
6. **repeat** *// repeat until condition is not valid*

End Procedure

Meta-heuristic algorithms include several different groups of algorithms that are classified according to the behavior and strategies of the algorithms. This research will focus on Swarm Intelligence algorithms. A general overview and discussion of these algorithms is presented in the following sections.

2.1.2 Swarm Intelligence Algorithms

Swarm intelligence algorithms describe several related approaches to solving problems that resemble biological swarms and are based on social behavioral models of insects or animals (Blum and Li (2008)). The SI algorithms include multiple swarms that share information in a search space to find a solution. Specifically, the SI is a meta-heuristic that has been developed by simulating the behavior of real swarms or insects to solve problems (Kennedy (2011)). SI algorithms are used to optimize complex problems that have a distributed structure. In addition, they can be applied in systems that have elastic and flexible properties without influencing the overall structure. Consequently, many SI techniques have been applied in cloud computing to improve resource scheduling,

for example ACO and PSO (Guo et al. (2012)). Details on each of these algorithms will now be presented.

2.1.2.1 Particle Swarm Optimization Algorithms

PSO is a global search algorithm that consists of a set of particles characterized by random velocities and positions. Each particle in PSO has a velocity, which represents the movement in the search space and it dynamically adjusts this based on its previous behaviour. Therefore, particles tend to move towards better points within the search space and thus they search the solution space by changing their position and velocity (Trelea (2003)).

There are several SI algorithms in use; however, PSO has been shown to give better results in terms of performance and complexity in large-scale environments. It reduces the computational time compared to other SI algorithms such as Genetic algorithms. It also uses the real values of numbers and does not need to encode these to binary as happens in GA. Thus, in the current research, a PSO algorithm will be used in optimization. The PSO algorithm provides effective performance in a distributed environment such as cloud computing in terms of computational time, where it is faster than meta-heuristic algorithms such as GA and ACO. According to (Pongchairerks (2009)), PSO was found to be faster and simpler than GA in terms of both processing and implementation and it includes few parameters to adjust and improve the convergence speed (for more detail see Section 2.2.1). Thus, PSO will be used in this research to optimize the task scheduling problem.

However, PSO does have some limitations as it suffers from local optima and a low convergence rate in the large space. PSO can overcome these problems in two ways. Firstly, a variant of the PSO algorithm can be used by changing some of its parameters and formulas. Secondly, it can be combined with other meta-heuristic algorithms (see Section 2.2.2 for more detail). In this thesis, the two methods will be used based on the nature of the problem and the objectives. For negotiation and task scheduling, the modified PSO will be used whilst in VM allocation PSO will be combined with another algorithm to improve its quality in data clustering with K-means algorithm. In the next

section, general information on the most common meta-heuristic algorithms will be presented.

2.1.2.2 Ant Colony Optimization Algorithms

ACO is a meta-heuristic algorithm that can be used to solve complex optimization problems by simulating the behavior of ants when searching for foods. It uses a mechanism that simulates the behavior of real ant colonies who cooperate using pheromone paths. When the ants move to find a food source, they follow a path and they leave pheromones on it. Other ants can then follow the trails to the food source by sensing the pheromone (Shishira et al. (2016)). Most ants select the shortest path as a larger amount of pheromones has accumulated on this path (Dorigo et al. (2006); Tawfeek et al. (2015)).

The ACO algorithm has many advantages such as adaptability, robustness and redundancy. ACO methods are applied to solve discrete optimization problems that rely on finding the shortest path to the goal. Moreover, it has been successfully used in other applications such as routing problems in dynamic network, solving traveling salesman problems, multidimensional knapsack problems, job shop scheduling, and task scheduling in cloud environments (Shishira et al. (2016)). Its disadvantages are the overheads and the fact the algorithm converges to the local optimal solution. However, the main problem with the ACO is that convergence is slow; therefore, it is applied in a small area of space (Pongchairerks (2009)).

The steps involved in the ACO algorithm are summarized in Algorithm 2.2. Specifically, the first step involved in searching for a solution is to initialize the pheromone and the optimal solution then distributes the ants randomly as shown in Lines 1-3. The pheromones of all the ants are detected and is computed (Line 6). Then, the fitness function is computed to detect the best path (Line 8). When the short path is detected, the best value is updated (Lines 9-11). The local value and the global value are updated in Lines 12-13. These steps are repeated until the shortest path is selected which based on the large amount of pheromones accumulated on this path.

Algorithm 2.2: Ant Colony Optimization Algorithm

Procedure ACO
1. Initialize (pheromone) *// initialize the pheromone value*
2. Optimal =null *// initialize the best path value*
3. Initialize(ants) *//distribute the ants randomly*
4. **For** each ants
5. **For** all the paths
6. detect the pheromones on each path *// detect the pheromone values*
7. **End For**
8. Best=Computefitness (ants) *// compute the fitness function*
9. **If** (Best < Optimal)
10. Optimal=Best *// detect the shortest path*
11. **End If**
12. Update local pheromone ( )
13. Update global pheromone ( )
14. **Repea**t the steps from 5 to 13 until end condition is met.
15. **Return** Optimal

End Procedure

2.1.3 Evolutionary Algorithms

Evolutionary algorithms are methods that draw on concepts from biological evolution, such as reproduction, mutation and recombination, to solve optimization problems (Simon (2013)). EA algorithms share the same basic idea but differ in their implementation depending on the problems that need to be solved. Evolutionary algorithms are a set of meta-heuristics used to solve many complex problems and aim to find a near-optimal solution because the optimal solution is too complex (Simon (2013)). The most popular EA algorithm is the Genetic algorithm. In the following section the Genetic algorithm will described in more detail.

2.1.3.1 Genetic Algorithms

The GA method is one of the evolutionary algorithm and it is designed to obtain a near-optimal solution in large space problems (Whitley (2014)). The development of a GA method is based on natural selection and Mendel's laws of inheritance in that it simulates the process of natural evolution which involves encoding of chromosomes; selection of Genetic manipulation and evolution; crossover and mutation operation; and, finally, generating and evaluating new generations (Gendreau and Potvin (2005)).

GAs take a large amount of time during optimization because they include many parameters that need to be processed and encoded.

Algorithm 2.3 describes the main steps involved in GA, which defines a set of solutions that collectively represent a population. The algorithm starts creating the population by randomly generating a collection of solutions in Line 1. The solutions are then evaluated by computing the fitness function (Line 3), following which the result is compared with the current solution. A new solution is subsequently formed by a crossover function in Line 5. The next step is a mutation function, which is invoked to replace the worst solution with a new one (Line 6). Finally, these steps are repeated until the stop condition is met (Lines 2-7).

Algorithm 2.3: The pseudo code of the Genetic Algorithm

```
Procedure Genetic ( )
1.    Initialize (P, C)                          // initialize the population
2.    while (the termination condition is satisfied) do
3.          Evaluate (P)
4.          Best =Select (P)                     // select best fitness.
5.          Crossover(P,C)                       // to produce new solution
6.          Mutation  (P,C)                      // replace worst solution with best one
7.    end while
8.    return Best                                // return best solution
End Procedure
```

When comparing these algorithms in terms of their application in cloud computing environments, the GA algorithm finds a near optimal solution, and it does not become trapped in local optimal solutions. However, there is no guarantee of finding a global maximum using GA and it can consume more time than PSO. The drawback of ACO is that it is inefficient in terms of load balancing, because it starts randomly and sometimes fails to find the global optimal solution. Moreover, the time for convergence using ACO is uncertain and depends on the problem space and the number of dimension. In comparison, PSO employs a fast search and its calculation is simple, but it may fall into local optima and suffers from premature convergence. In sum, the common meta-heuristic algorithms used in cloud computing have been explored and compared. Based on this analysis, PSO is chosen for this study. In the Section 2.2, the PSO algorithm will be presented in more detail.

2.1.4 Methods of Evaluating Multiple Objectives

Optimization methods can deal with single or multiple objectives to evaluate solutions. The methods that deal with two or three objectives are called multi-objective optimization, whereas if the number of objectives is greater than three it is called many-objective optimization (Maltese et al. (2016)). The scalability problems are produced using multi-objective optimization methods when the number of objective increases (Li et al. (2015)).

Specifically, MOO is applied to problems that involve multiple objectives and is concerned with mathematical optimization problems where two or three objective needs to be evaluated simultaneously (Chow et al. (2004); Srinivasan and Seow (2003)). MOO is defined as a set of decision elements with specific constraints that optimizes the objective function when many objective functions are involved. MOO presents a possible set of solutions, which are evaluated using trade-offs among several objective functions. A selection strategy is then applied to choose an acceptable solution. The MOO problem, according to (Marler and Arora (2004)) can be mathematically defined as follows:

$$\mathbf{Min}\ \mathbf{F}(\mathbf{x}) = (\mathbf{F1}(\mathbf{x});\ \mathbf{F2}(\mathbf{x}); \ldots;\ \mathbf{Fk}(\mathbf{x})) \qquad (2.1)$$

where:
k is the number of objectives in the fitness function
F(x) is the objective function of the solution x
F1(x) is the objective function for objective 1 for solution x
Fk(x) is the objective function for objective k for solution x, in MOO the k=3

Specifically, several methods have been proposed in the literature for dealing with multi-objective optimization, and these can be divided into three categories: domination, decomposition and ranking approaches (Reyes-Sierra and Coello (2006); Garza-Fabre et al. (2009)). Details on each of these strategies will now be presented and discussed.

First, domination approaches will be considered. In these methods, multi-objective problems are optimized by simultaneously optimizing all the objectives using the concept of Pareto dominance. In most MOO algorithms, the concept of domination is defined as all the solutions that are not dominated by other solutions (Marler and Arora (2004)). In general, the solution can be defined as Pareto optimal if and only if no other

solution exists that dominates it, whereupon the set is called the Pareto optimal set (Tripathi et al. (2007)). However, Pareto set has been shown that, as the number of objectives increases, the convergence ability and the performance of dominance approach decreases (Li et al. (2015)). These methods need to maintain a diverse set of solutions and they cannot apply these when the number of objectives increase.

The second set of approaches are described as decomposition approaches. In these methods, the multiple objectives are aggregated into a single objective, the simple approach and widely used is the weighted sum approach (Garza-Fabre et al. (2009)). The weighted sum approach combines multiple objectives by using their weight, thus forming one single objective as shown in Equation 2.2. However, the complexity in this approach lies in determining the weight values for each objective depending on its importance (Reyes-Sierra and Coello (2006)). Different weight values produce variations in the results and in the processing time.

$$\mathbf{F(x)} = \sum_{\mathbf{m=1}}^{\mathbf{M}} \boldsymbol{F}_m(\mathbf{x}).\ \mathbf{W_m} \tag{2.2}$$

where:
M is the number of objectives
W is the weight value for each objective it ranges from [0-1] and $\sum_{m=1}^{M} W_m = 1$
F(x) is the value for x based on multi-objective
$F_m(x)$ is the value for x based on objective m

The final set of approaches are those based on ranking objectives that are non-Pareto techniques and do not require balancing the weights. Several ranking objectives have been developed such as average ranking, sum ranking, or maximum ranking (Garza-Fabre et al. (2009)). However, instead of comparing two solutions directly, some researchers compare them by checking the rank of each one respect to specific goals. The ranking concept is a collection of items arranged in order according to some factors that they all possess (Gao et al. (2014)). The position of each element in the ranking is called the rank, which is usually represented with a numerical value that refers to its order in the ranking. Then these ranks are combined into a single value that is represent the rank of the solution according to all objectives based on the maximization and minimization of objectives.

The ranking strategy can be applied using several methods such as an average ranking or a maximum ranking. An average ranking selects one objective and builds a ranking

list using the fitness of each solution for the chosen objective then compute the sum of the average rank as shown in Equation 2.3 (Garza-Fabre et al. (2009)).

$$\mathbf{AR(x)} = \sum_{\mathbf{m=1}}^{\mathbf{M}} \mathbf{R_m(x)} \tag{2.3}$$

where:
AR(x) is the average rank of solution x
$\mathbf{R_m(x)}$ is the rank of solution x relative to objective m
M is the number of objectives

The minimum ranking strategy is similar to the average ranking but instead of taken the summation of rans it takes the smallest rank for each solution (Garza-Fabre et al. (2009)). In the next section, more detail about the PSO algorithm will be provided, including its parameters, variants and modifications.

2.2 Particle Swarm Optimization

In this section, the standard PSO algorithm and its improved variants will be discussed in detail. Specifically, Section 2.2.1 provides an overview of PSO algorithm characteristics and describes the steps involved in applying PSO in terms of its operations and parameters. Section 2.2.2 discusses several methods for improving PSO algorithms. General information on Parallel PSO is presented in Section 2.2.3. Finally, in Section 2.2.4 clustering using the PSO algorithm will be described and discussed.

2.2.1 Overview

PSO is one of the SI algorithms that comprise a simulation inspired by the social behavior of animals, and was first introduced by (Eberhart and Kennedy (1995)). It is a search optimization method that finds the best optimal solution through sharing information in the swarm (Reyes-Sierra and Coello (2006)). PSO algorithms deal with swarm of particles, where each particle represents a solution of the problem in the search space. Each particle has two values; one is the best personal experience (pbest) for the particle itself, while the other is (gbest) which represents the best solution among all particles in the swarm. The particle position is computed according to pbest and gbest values, and the velocity determines the speed of movement of the particle

depending on the difference between the particle's previous position and its current position (Kennedy (2011). The velocity of the particle is used to control the movement of the particles and to prevent particles from falling in the local optima by using the inertia w parameter.

Specifically, PSO shifts the particle position in each iteration by updating particle xi at iteration t, as shown in Equation 2.4, and the velocity is then updated based on the two best values and inertia w as shown in Equation 2.5 (Eberhart and Kennedy (1995); Alkayal et al. (2016)).

$$\mathbf{xi}\,(\mathbf{t}) \;=\; \mathbf{xi}\,(\mathbf{t}\;-\;\mathbf{1}) \;+\; \mathbf{vi}\,(\mathbf{t}) \tag{2.4}$$

where:
xi (t) is the current position of particle i at iteration t
xi (t-1) is the position of the particle i at iteration t-1
vi (t) is the velocity of particle i at iteration t

$$\mathbf{vi}\,(\mathbf{t}) = \mathbf{w} \times \mathbf{vi}\,(\mathbf{t}\;-\;\mathbf{1}) + \mathbf{r1} \times \mathbf{c1} \times (\mathbf{pbesti}\;-\;\mathbf{xi}\,(\mathbf{t})) + \mathbf{r2} \times \mathbf{c2} \times (\mathbf{gbesti} - \mathbf{xi}\,(\mathbf{t})) \tag{2.5}$$

where:
xi (t) is the current position of particle i at iteration t
vi (t) is the velocity of particle i at iteration t
vi (t − 1) is the velocity of particle i at iteration t- 1
pbesti is the best position of particle i
gbesti is the position of best particle in a population
w is the inertia weight with range [0, 1]
r1, r2 are the random numbers with range [0, 1]
c1, c2 are the acceleration coefficients with range [0, 1]

In each iteration, there are two best values for each particle. One is the pbest, which is the best for each particle in the swarm, and then the best of all the pbest values is selected as gbest for all particles as shown in Equations 2.6 and 2.7.

$$\mathbf{pbest}(\mathbf{i},\mathbf{t}) = \mathbf{min}\,(\mathbf{f}(\mathbf{pi})) \tag{2.6}$$

$$\mathbf{gbest}(\mathbf{t}) = \mathbf{min}(\mathbf{pbest}\,(\mathbf{i},\mathbf{t})) \tag{2.7}$$

where:
i is the index of particle
t is the iteration number
pbest (i, t) is the best value for particle i in iteration t
f (pi) is value of fitness function of particle i
gbest (t) is the global best for all particles in iteration t

In PSO, several parameters are used to control the execution of the PSO algorithm and improve the results. These parameters include the number of particles, the dimension of particles, the maximum number of iterations, Vmax, learning factors c1 and c2, inertia weight (w), and random numbers r1 and r2. Vmax controls the speed of movement of the particles, which is represented by velocities. The Vmax constant should be selected so that it allows particles to escape from local optima. According to several research studies, situations where the Vmax value is dynamically changing could result in better performance to a range [-Vmax +Vmax] to control the movement (Eberhart and Shi (2000)).

The constant numbers c1 and c2 determine the acceleration values of the particles and their high values correspond to past sub-optimal solutions, whereas low values allow particles to fall in local optima. The c1 constant is related to pbest, while c2 determines how well the particle follows the swarm and is related to gbest. The particle is influenced by its own best position and the best position of its neighbors, so the values of c1 and c2 are set to fixed equal values (Reyes-Sierra and Coello (2006)). The random numbers r1 and r2 are used to provide random movement of particles inside the search space, and their values can be in the range [0-1].

The inertia weight w is computed as shown in Equation 2.8. It is a positive linear function based on the iteration of the algorithm and its value is selected to provide a balance between local and global exploration, and thus ensures the optimal solution can be found with a small number of iterations. (Reyes-Sierra and Coello (2006)) demonstrated that the dynamic value of inertia w performs better than the static value. The value of w therefore changes dynamically with the iteration of the algorithm in the range [0-1], as shown in Equation 2.8.

$$\boldsymbol{w = UW - ((i) / Maxi) * (UW - LW)} \tag{2.8}$$

where:
w is the inertia value that controls the movement of particles and this range from 0-1
UW is the high value of w, which is one
LW is the low value of w, which is zero
i is the iteration number
Maxi is the maximum number of iterations

The main steps in a standard PSO algorithm are summarized in Algorithm 2.4 (Durillo et al. (2009)). The algorithm starts by initializing the particles with the available solutions. The fitness function of each particle is then computed according to predefined objectives that select the best fitness value (Lines 4 and 5). Following this, the velocity and position of each particle are updated using Equations 2.4 and 2.5 (lines 6-7). The best value for each particle is then selected and compared with pbest and gbest values (Lines 8-14). Finally, these steps are repeated until a stopping condition is met.

Algorithm 2.4: Pseudo code of a Standard PSO Algorithm.

```
Procedure PSO ( )
1.      INITIALIZE (S,V, P, pbest, gbest)        //initialize swarm, velocity, position, pbest, gbest
2.      While (stop criteria does not satisfied) do        // iterate while not stop
3.          For each p ϵ S                                 // iterate through all particle
4.              f(p)=Computefitness(p)                     // compute fitness function for particle p
5.              best=Selectbestvalue(f(p))                 // select best value
6.              Updatevelocity(p)                          // update velocity using Equation 2.4
7.              Updateposition(p)                          // update position using Equation 2.5
8.              If (best < pbest)
9.                      pbest=best                         // update pbest value
10.             End if
11.         End For
12.         If  (best <gbest)
13.                 gbest=best                             // update gbest value
14.         End if
15.     End while
16.     Return gbest
End procedure
```

PSO was originally developed for application in continuous optimization problems (Izakian et al. (2010)). Therefore, to apply PSO in a combinational problem, the problem needs to be transformed into discrete values and to encode the representation of the search space. The Small Position Value (SPV) rule is one of the most popular techniques used for this purpose and uses a 1 x n vector to encode n particles of PSO (Shishira et al. (2016)). Another technique known as Integer-PSO is presented in (Netjinda et al. (2012)), which is used when there is a substantial difference in the length of tasks and the processing speed of the resources technique outperforms the SPV. In Garg (2016), crossover and mutation strategies of the Genetic algorithm were applied in PSO to deal with discrete problems. Other researchers, such as Khanesar et al. (2007) have used the binary PSO method for discrete problems, especially in combinatorial optimization where its variables take only discrete values.

2.2.2 Improving Particle Swarm Optimization Algorithms

Several researchers have improved the solutions generated by PSO algorithms using methods that include changing the initial population and modifying the values of the operators of PSO (see Section 2.2.2.1). In Section 2.2.2.2, methods for improving and evaluating objectives in the PSO algorithms will be discussed.

2.2.2.1 Modifying the Initialization and Encoding of Particles

Several methods have been applied to initialize the position and velocity of the particles (Li-Ping et al. (2005)). For example, in Wu et al. (2010) a greedy adaptive search algorithm was developed to initial swarm by mapping the shortest task to the fastest resource. Alternatively, fuzzy matrices were used by (Yang et al. (2011)) to represent the position and velocities of particles such that each element in the matrix denotes a fuzzy relationship between the resources and tasks. In Izakian et al. (2009), a First Come First Serve (FCFS) method was used to initialize the PSO particles, whereas in Liu et al. (2013) randomness with constraints was used.

Heuristic algorithms are sometimes combined with PSO to improve the initialization of the particles such as Max-min in (Miranda and Fonseca (2002)) and first Fit (FF) in (Low et al. (2010)). Alternatively, Alkhashai and Omara (2016) proposed a hybrid Best Fit, PSO, and Tabu Search (BFPSOTS) algorithm based on the PSO algorithm to achieve suitable scheduling of the users' tasks in cloud computing environments. Within the hybrid algorithm, the Best-fit algorithm has been merged into PSO to initiate the swarm rather than generating it randomly, and the Tabu search algorithm has been combined with PSO to improve local research by avoiding any fall in local optimality and improving the quality of the solution.

Conversely, other researchers represent the particle, as a vector of $1 \times n$ where n is the number of the dimensions of the solution (Kennedy (2011). Another representation of encoding particles occurs in a matrix with $m \times n$, where m is the number of particles and n is the number of dimensions for particles (Abdi et al. (2014)). Deciding which is the most suitable encoding representation depending on the nature of the of problem's variables (continuous or discrete) and the number of solutions. In this thesis, the first

representation will be used for the negotiation problem where a vector will be used for each particle because there are a small number of particles representing data centers. In task scheduling, a vector of 1 × n will be used to represent the VMs. In the VM allocation, a matrix with m × n elements will be used to map VMs to hosts with the constraint that only one host can allocate each VM. This matrix representation will be used to simplify the process of controlling the migration of VMs from one host to another.

2.2.2.2 Modifying Based on the Number of Objectives

In this section, the strategies that are used to improve the PSO algorithms based on the number of objectives will be discussed. The complexity of MOO problems such as large search spaces, uncertainty and noise, which means that most single objective optimization techniques are not suitable to solve MOO. Consequently, many techniques have been developed to achieve MOO (Cagnina et al. (2005)). However, there are certain limitations in these programming techniques as they generate a single solution per run (Marler and Arora (2004)). This issue thus initiated the development of other approaches. Compared to traditional mathematical programming techniques, PSO algorithms were found to be suitable as they are population based and can manage a whole set of solutions at a time, rather than just one. In this research, PSO is focused upon because of its proven flexibility in responding to rapid changes in the system and its ability to adapt to the external environment during run-time (Blum and Li (2008)). PSO can use any methods of evaluating objectives that are discussed in Section 2.1.4.

First, PSO can use the weighted sum method that considers each objective function separately by evaluating each particle for one objective function at a time, and the determination of the best position is performed in a similar way to the single-objective optimization case (Garza-Fabre et al. (2009)).

Second, the Pareto set will be considered, which can be used in PSO to evaluate objective functions in each particle based on the concept of Pareto optimality (Durillo et al. (2009)). In the case of multi-objective, selecting the best solution is not straightforward as there can be many non-dominated solutions in the neighborhood of a particle and only one solution from leaders is selected to update the velocity (Li et al.

(2015)). For multi-objective optimization, the PSO needs to consider Pareto dominance every time it updates particles and stores non-dominated solutions (Reyes-Sierra and Coello (2006)). Because MOO problems involve a set of Pareto optimal solutions rather than a single optimum solution, two main points should to be addressed when the PSO is applied to MOO problems (Cabrera et al. (2010)). The first of these concerns the method of selecting the leader archive, which is used to store the set of non-dominant solutions, to control the particles' movement inside search space. The second concerns how to maintain the best solutions.

The pseudo-code of a general multi-objective PSO (MOPSO) extends the steps of standard PSO and is specified in detail in Algorithm 2.5 (Durillo et al. (2009)). Specifically, the algorithm starts by initializing the particles, velocity, pbest, and archive to store the leaders. After initialization, the objective function must be calculated to evaluate objectives for all the particles. Non-dominant solutions for each particle are then selected using an evaluatefitness function (Line 4). Following this, the leader is then selected by using a select_leader function (Line 5). The velocity, position, and archive are then updated (Lines 6-8). The loop will continue until the stop criteria is met (Line 9). Finally, the best solution is returned in the archive value (Line 10).

Algorithm 2.5: Pseudo code of a MOPSO Algorithm based on Pareto set**.**

Procedure MOPSO **( )**
1. Initialize (particle ,position, velocity)
2. Initialize_archive (archive)
3. **do**
4. non-dominate =Evaluatefitness (particle)
5. archive=Select_leader (non-dominate)
6. Updatevelocity (velocity)
7. Updateposition (position)
8. Updatearchive(archive)
9. **while** (stopping criterion is reached)
10. **return** archive

End Procedure

Finally, when we consider the modified ranking methods with PSO algorithms, several ranking strategies had been applied with PSO. For example, Gao et al. (2014) used maximum ranking and took the smallest rank for all objectives. In (Wan and Li (2008)), one objective was used and they ranked the fitness function of solutions based on PSO.

MaOO, on the other hand, differs from MOO because several issues required to be taken into account when dealing with problems that includes more than three objective functions (Figueiredo et al. (2016)). For example, using Pareto-based algorithms when the number of objectives increases, the process of comparing solutions using just the Pareto dominance relation become more difficult (Cabrera et al. (2010)). In addition, the process of selecting the leader from the archive in order to improve the convergence towards solutions using Pareto set is still needs to be addressed (Marler and Arora (2004)). Thus, it is difficult to solve a MaOO with a Pareto set and many techniques have been developed to overcome the limitations of applied Pareto-dominance (Figueiredo et al. (2016). There is, therefore, still a need to improve PSO algorithms to deal with many objectives. In fact, particle swarm based algorithms seem particularly suitable for multi-objective optimization and they can be extended to handle many-objective optimization. In this research, we extend the MOPSO in order to deal with many-objective problems as one objective. This is because of the high convergence speed in single-objective optimization (Figueiredo et al. (2016)). In this research, a modified ranking strategy with many-objective will be developed to improve task scheduling and VM allocation based on four and five objectives. In the SLA negotiation, the weighted sum approach will be used because only three objectives are being dealt with.

2.2.3 Parallel Particle Swarm Optimization

Advances in computing technologies have improved parallel algorithms, which provide several advantages compared to serial algorithms. The methodology of parallel computing focuses on dividing large problems into smaller ones and distributing them among different nodes to find a solution in rapid time (Chang et al. (2005)). Parallel computing has the advantages of reducing time consumption, and increasing the rate at which complex problems are solved (Grama (2003)). The cost of complex optimization problems motivated the subsequent development of parallel optimization algorithms. These algorithms thus solve smaller problems simultaneously (Hao et al. (2016)). The parallel algorithm is concerned with running the same code on multiple processors with the goal of reducing the running time (Grama (2003)). Thus, for optimization, a parallel algorithm will be studied in this thesis.

The PSO algorithm is not time-consuming and can easily be parallelized because it consists of a set of particles that can move individually and then share the results of their movements. A PPSO approach is used to deal with different environments at the same time by applying the same algorithms in a different search space and exchanging the information obtained from these environments. This is applicable to the model of distributed data centers used in this thesis. This will therefore help to improve throughput and search speed. In PSO, one swarm consists of several particles, whereas in PPSO it can include many swarms. The PSO algorithm is used for applying parallel algorithms in a distributed system because, in each step of the iteration, all the particles are independent of each other and thus it is easy to evaluate each particle in parallel. The PPSO is therefore developed to maximize the throughput of the algorithm, reduce computation time, and improve the global search to prevent it falling in to local convergence.

PPSO handles the local and global convergence of the problem to communicate between multiple swarms. PPSO can be implemented in two main ways: the first of which is the most commonly applied model. This is based on dividing the search space into N independent multi-swarms randomly and initializing each individual swarm. The PSO for each swarm is then applied and the fitness function of each independent swarm is evaluated to determine the particle best (pbest) and the swarm best (sbest) in each individual swarm. The velocity and position of each particle in every swarm is updated with new results. The global best (gbest) is determined by comparing the swarm best (sbest) across all the swarms as shown in Figure 2.2. The second method uses several nodes to run many swarms in parallel and the main node used for updating the next iteration then selects the best swarm. This model is used when the same data exists in all nodes and parallelism techniques are used to speed up the process of evaluation. In this work, the first method will be used which involves dividing the search space into smaller spaces and running the swarm in different nodes. This approach is suitable for the proposed model structure as it is based on distributed nodes and has a distributed data center with different numbers of resources and different characteristics. To apply parallel algorithms, the main performance problem concerns facilitating communication and cooperation between different nodes. Two main types of memory architecture can be used to apply parallel algorithms: shared memory and distributed

memory. In the former, all nodes communicate via main memory, while in the latter each node has its own memory. The structure of the proposed model in this research, which includes multiple distributed data centers, is suitable for use with a distributed memory architecture.

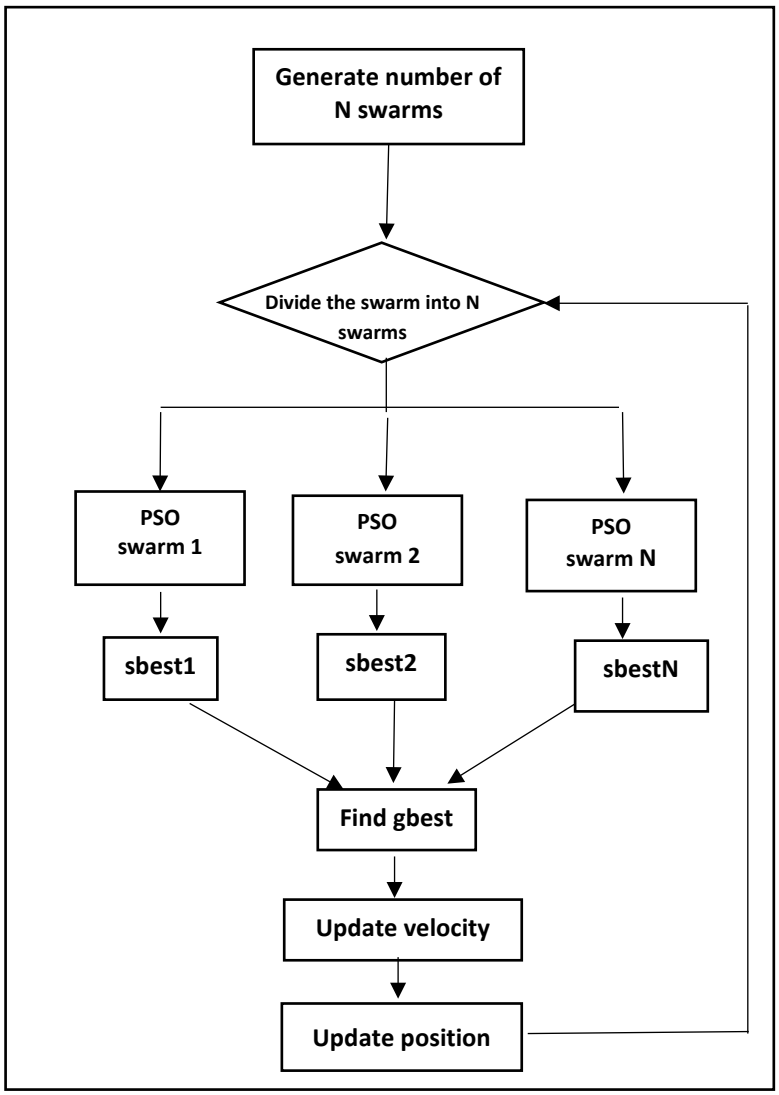

Figure 2.2: Flowchart illustrating Parallel Particle Swarm Optimization.

The PPSO algorithm follows the same steps as the PSO algorithm but with one extra step, which is the communication step. This step is required in PPSO to define the mode of cooperation between swarms. The communication between multi-swarms can be either synchronous or asynchronous. There are several parallel adaptations of PSO, including synchronous PSO (Schutte et al. (2004)) and asynchronous variants (Koh et al. (2006); Venter and Sobieszczanski-Sobieski (2006)). PPSO has been adapted to

solve multi-objective optimization problems (Fan and Chang (2009)). The nature of the communication structure plays a major role in improving the optimization algorithm. Parallel Particle Swarm Optimizers based on the communication between particles are be classified into three categories, namely master/slave PSO, ring PSO and fully connected PSO, as shown in Figure 2.3 (Tu and Liang (2011)).

In the master-slave model, there is generally one master node responsible for managing other slave nodes. Each slave node runs the swarm then sends the results to the master node. In this model, finding the global optimal is achieved in the master node, while evaluating the objective function and modifying particle velocities is executed in the slave nodes. It is simple and easy to implement, which has led to its widespread use in optimizing large-scale problems. In the Ring PSO model, the swarm is divided into multiple swarms, each of which is placed on one node. Each node runs a PSO algorithm on its swarm. Choosing the optimal value and modifying individual velocities occurs locally within each swarm. After a defined number of iterations, the best solution in each node is then migrated to neighboring nodes. In general, this model is suitable for small sized spaces that include small numbers of swarms. However, this model can only be implemented in a shared memory architecture because of its need for communication in each iteration. In the fully connected topology, all nodes are directly connected to each other. All particles in the entire swarm move directly to the best particle found in the whole swarm. This model, like the ring PSO, requires a great deal of communication as it takes more time to manage the swarm and control the movement of each particle.

Based on the advantages of these topologies and the structure of the proposed model, the master/slave topology will be applied to the PPSO algorithm such that the Manager Agent in our model is the master node and distributed data centers are the slaves. This model is chosen because it is suitable for the proposed structure and involves the least amount of communication and control overheads in an efficient manner than other topologies.

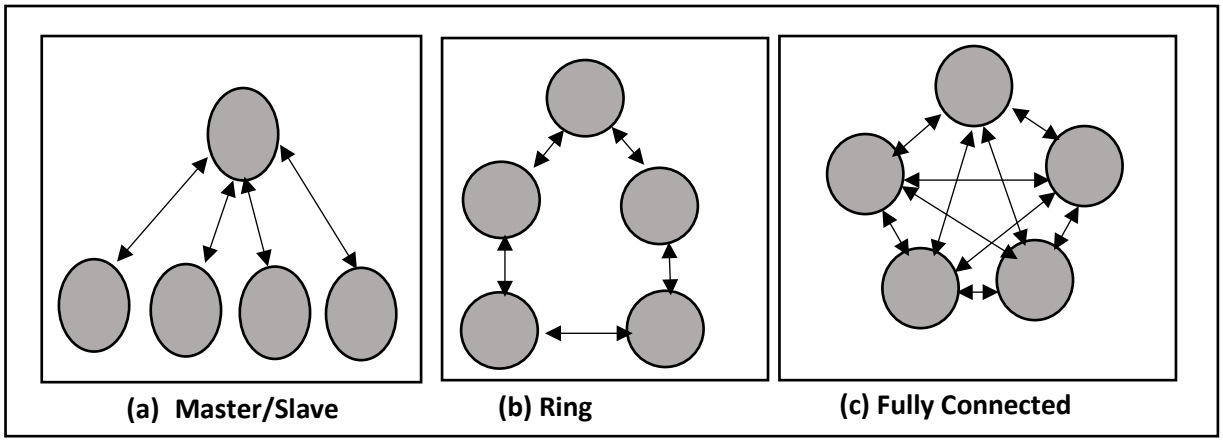

Figure 2.3: PPSO Topologies.

Most research on parallelized versions of the PSO scheme have involved testing different communication strategies asynchronously or synchronously (Koh et al (2006)). Additionally, they have also focused on the implementation of the algorithm in the cluster or GPU processor (Zhou and Tan (2009); Laguna-Sánchez et al. (2009)). However, one of the implementation studies showed how to map the PSO on the parallel architecture. In this thesis, the focus will be on improving the method of communication between nodes to affect the design of the algorithm. In (Gonsalves and Egashira (2013)) the parallel version of the PSO algorithm was described without introducing any complexity in time and quality. In this research, the parallel version of the algorithm is used to improve throughput by reducing the data redundancy of nodes compared to the standard PSO algorithm.

2.2.4 Clustering Based Particle Swarm Optimization

Clustering is a technique for dividing a large dataset into small groups with similar characteristics (Adrian and Heryawan (2015)). It is an attractive and a major task in data mining that is used in many applications such as text mining and spatial data applications. The clustering method is divided into hierarchical clustering and partitioning clustering (Gan et al. (2007)). Hierarchical clustering can be applied by either merging two clusters or splitting a cluster in such that it iteratively merges the two nearest clusters until only one cluster remains in the dataset (Berkhin (2006)).

Another method of clustering is partition clustering which clusters the dataset into several clusters in one level.

One of the most popular approaches to clustering has been to design clustering as an optimization problem with several objectives and constraints (Rana et al. (2011)). In this instance, the partitioning of a given dataset satisfies the objective function based on several objectives. The computed objective functions are based on statistical relationships between the data points in the dataset and the cluster-centroids of each cluster (Kuo et al. (2011)). Several optimization methods have been proposed to solve the clustering problem where the objective function aims to maximize or minimize the inter cluster distance. One of the most efficient algorithms applied to clustering is the PSO algorithm (Abdel-Kader (2010); Govindarajan et al. (2013)). However, PSO often leads to premature convergence and its performance is highly dependent on parameter tuning, therefore several researches have been made to improve its quality and performance in different ways (Rana et al. (2011)). For example, PSO is applied to determine the appropriate number of clusters and initialize their centroids. Upon initialization, PSO has a limitation in that it does not perform well on a large and complex dataset. This is because each solution is represented by a particle that has a boundary of the search space and it cannot then explore any further. Kuo et al. (2011) proposed a PSO algorithm to beat this problem by predefining the number of clusters at the beginning of the execution. To improve the efficiency of the PSO in data clustering, it is sometimes hybridized with other algorithms such as K-means in Mahendiran et al. (2012) and fuzzy techniques in Benameur et al. (2009).

The K-means algorithm is regarded as the most popular technique for clustering data (Hatamlou et al. (2013)). This algorithm starts with random centroids and each object is assigned to the closest centroid as shown in Algorithm 2.6 (Line 1). The centroid values are then re-calculated based on the points of each cluster as shown in Lines 2-7. This procedure continues until a termination criterion is met. Although the K-means algorithm is simple, its performance varies greatly depending on the initialization values of centroids. To solve this limitation, research on the setting of initial centroids has been ongoing using K-means combined with optimization approaches such as GA and PSO to find the global optimization and escape from local optima (Premalatha and Natarajan (2009)).

Algorithm 2.6: K-means Clustering Algorithm.
Input: dataset points, K *// where k is the number of clusters*
Output: k clusters with points assigned to each of them
Procedure k-means (points, k)
1. random initialization of k cluster centroids
2. **do**
3. **for** all points
4. Assign points to closest cluster
5. **end for**
6. **for** all k clusters
7. compute the new centroids
8. **end for**
9. **repeat** until there is no more change in the centroid values

End Procedure

Various research studies have been conducted to improve the efficiency of the K-means algorithm using PSO. PSO provides the optimal initial centroids, thus using these values the K-means algorithm produces better clusters and more accurate results than it does when used alone. In Neshat et al. (2012), a combination of PSO and K-means was developed to take the feature of the PSO in the global search and the speed of convergence of K-means. Furthermore, Saini et al. (2014) developed clustering algorithm based on PSO and K-means to improve the quality of clustering results.

Benameur et al. (2009) proposed a PSO method with a fuzzy clustering algorithm, which developed to produce a better clustering of solutions by dividing the dataset into multiple swarms. Moreover, (Attea (2010)) discovered that as the performance of clustering algorithms degrades there are increasingly more overlaps among clusters in a dataset. These issues have motivated researchers to develop an innovation multi-objective PSO framework for clustering data that delivers more effective results than state-of-the-art clustering algorithms.

GA operators, which involve selection, mutation and crossover, can be applied to produce a new generation of chromosomes to improve the quality compared with the previous generation. Premalatha and Natarajan (2009) used the hybrid approach of PSO with GA to select a better solution without becoming trapped in the local optima, and to provide a quicker convergence speed. This renders PSO-GA more flexible in providing better results within a reasonable processing time. However, in clustering data, GA is more time consuming than K-means.

After studying different clustering techniques, the most common algorithms combined with PSO can be summarized as follows:

- K-means clustering, which produces greater accuracy and requires less computation time, although performance is based on the number of clusters.
- Clustering data using fuzzy measures produces results similar to K-means clustering, but requires more computation time because the fuzzy measure involves more calculations.
- GA yields effective results and, unlike K-means, does not require the number of clusters at the beginning of process. However, it consumes longer time to compute than K-means.

Given these comparisons, the requirements of the current research problem involve applying PSO clustering to cluster hosts in the data centers. This requires high-speed methods where the number of clusters is predefined and limited. Therefore, in this research, the PSO combined with K-means will be used for clustering.

2.3 Service Level Agreement Negotiation

This section presents an overview of the SLA negotiation process and discusses related work on SLA negotiation in cloud computing. Section 2.3.1 outlines the key points related to SLA negotiation in cloud computing. In Section 2.3.2, the automated SLA negotiation process based on multiple agents will be described. Section 2.3.3 will discuss in detail previous work that has applied PSO to improve SLA negotiation. Using parallel algorithms in developing SLA negotiation will be discussed in Section 2.3.4. Section 2.3.5 will describe the techniques and the measurement factors used in SLA monitoring algorithms. Finally, Section 2.3.6 summarizes the main points along with results in the field of SLA negotiation.

2.3.1 Overview

An SLA is a formal agreement between the providers and the consumers that defines the parameters of the services the consumer expects and the provider guarantees (Son and Jun (2013)). SLA is a term widely used to specify QoS objectives, which are achieved through a negotiation process (Abdullah and Talib (2012)). SLA in a cloud

computing context is defined as a contract signed between a cloud provider and a consumer that determines the set of QoS metrics that are used to measure services and penalties in case of violations (Son and Jun (2013)). SLA is an important way of ensuring that the level of service is in line with the expectations of both providers and consumers. QoS is defined as a set of parameters that specify the properties of the service including response time, throughput, availability and failure rate. Some of these QoS parameters are based on consumer requirements and others are related to the provider. Different consumers can adopt different QoS values for the same cloud service depending on the specific requirements in each case (Thio and Karunasekera (2005)).

To apply SLA in any system, the life cycle of management needs to be followed. SLA management is the process that incorporates allocated, negotiated, monitored, accounted and released resources. According to (Lissy and Mukhopadhyay, (2014)), the life cycle of SLA management comprises main four phases as shown in Figure 2.4. The first phase of the SLA life cycle involves creating the SLA Template based on the available and required information. The SLA Template involves a set of parameters that include definition of services, parties, penalty policies and QoS parameters. The next phase is SLA negotiation, which includes the process of completing the contracts between the providers and consumers. The third phase is the SLA implementation phase, which involves SLA generation according to the agreed contract between consumers and providers. After the SLA is completed, it is monitored and maintained in the final phase, SLA Monitoring, which is used to determine whether any changes are needed or if any SLA violation has taken place. The cycle then restarts from the beginning and will continue until the SLA is terminated. The research in this thesis will focus on two specific phases of SLA management, SLA negotiation and SLA monitoring. This is because these two phases play a significant role in the performance of cloud computing systems and improving them will therefore satisfy the objectives of cloud consumers and providers.

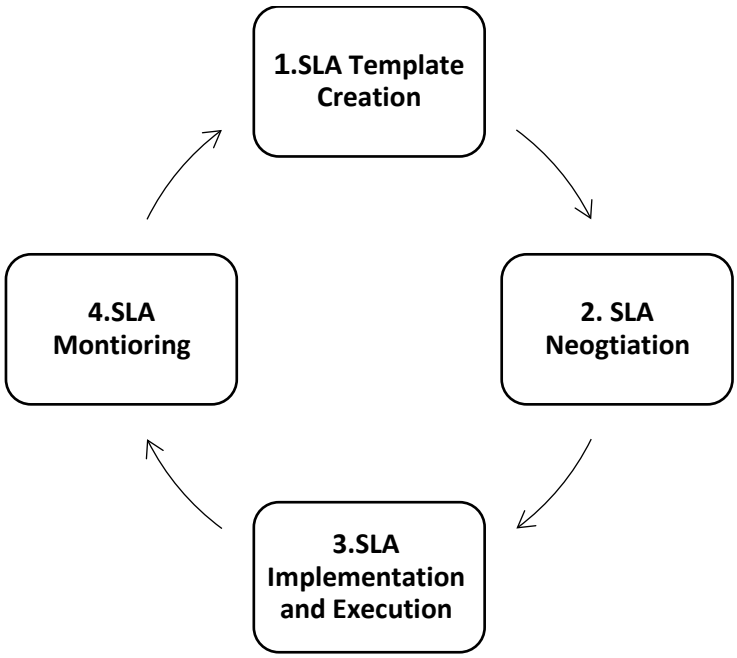

Figure 2.4: SLA Management Life Cycle.

The SLA metrics for cloud computing are defined in the three layers that represent cloud services which are SaaS, PaaS, and IaaS (Section 1.1.1). In this research, the focus will mainly be on the IaaS layer and the significant parameters contained within it, including virtual and physical resources.

Negotiation concept is generally defined as a decision-making process, where parties exchange information and attempt to reach a common acceptable agreement (Dastjerdi and Buyya (2012)). Such negotiation is necessary when there are conflicting objectives between different parties (Rajavel and Thangarathinam (2015)). During the negotiation, process the participants exchange negotiation offers to satisfy predefined negotiation goals under specific constraints.

The automated negotiation model is based on agent technologies and includes four elements: negotiation agent, negotiation object, negotiation protocol, and negotiation strategies (Zheng (2014)). Details of each of these components (Dastjerdi and Buyya (2012)) can be summarized as follows:

- **Negotiation Agent:** includes the parties involved in a negotiation process, which includes in cloud environment the providers, consumers and, in some models Brokers.

- **Negotiation Object**: a set of objectives that the negotiation agents must reach which can be single or multiple negotiation issues.

- **Negotiation Protocol**: specifies the rules that should be followed to control the negotiation. There are different models of negotiation based on the protocol used and the procedure for negotiation between the parties. Models of negotiation protocols can be bilateral, one-to-many, or many-to-many. The model used in this research will employ a one-to-many protocol, as one consumer agent will negotiate with multiple data centers to find the best resources.

- **Negotiation Strategy**: is the technique that negotiating agents use to achieve their objectives under the rules and protocol of negotiation. Negotiation strategies differ in terms of the number of criteria they handle and the number of parties involved in the process. Furthermore, they differ in the rules and constraints that control the negotiation process such as deadline time, the availability of resources or the other preferences of the parties. The most common negotiation strategies are categorized into four main types (Shen et al. (2002)): contract net, auction model, game theory based, and the discrete optimal control model. In the contract net, a manager agent evaluates the task announcements of other agents and bids for the tasks the agent is engaged with only two possible results: accept or reject. In the negotiation based on auction strategy, both parties need to agree on the offer without any penalties being imposed in case of rejection. In the model of game theory, each agent is responsible for providing a plan so that if all parties accept the plan the negotiation is finished. If they do not, agreement cannot be reached and the negotiation process is turned down. Finally, the optimal control model is developed as a market model. This model consists of three steps: collecting all the information, coming up with set of decisions, and making the final decision. This strategy aims to automate the negotiation process and provide the optimal solution. Thus, this approach will be used in the current proposed model to improve SLA negotiation in cloud computing.

SLA negotiation in cloud computing describes the process of allocating resources to fulfil the consumer's requirements and the provider's objectives. If cloud providers and cloud consumers have conflicting objectives, SLA negotiation is initiated to reach an agreement. To apply SLA negotiation, cloud consumers determine their QoS requirements, and then negotiate them with cloud providers. Various models and strategies of negotiation are used in cloud computing. However, the negotiation function in cloud computing is complex because the resources are distributed and diverse, and the process of selecting resources for executing tasks depends on the requirements of both the provider and consumer (Dastjerdi and Buyya (2012)).

An automated negotiation mechanism is thus necessary to achieve agreement and this has become a very important research topic in cloud computing in recent years. However, in an open and dynamic environment, individual negotiations may be initialized or finished during the process of negotiation. Concurrent negotiation based on multi-agents denotes a situation where negotiations are conducted in parallel or concurrently by an agent (Panagidi et al. (2014)).

In most research, SLA negotiation in cloud computing can be achieved using concurrent negotiations. In the concurrent protocol, there is one consumer and many providers residing within the framework of a one-to-many structure. In this model, consumers and providers conduct negotiations in parallel. Consumer agents should decide the providers from which they will use the resource. The consumer utilizes several threads to negotiate with every provider. A negotiation offer is a proposal for an agreement that one negotiation party makes to another based on predefined issues. The offer describes the service specification in SLA that has been negotiated and the associated QoS in terms of satisfaction guarantees. The negotiation constraints are control rules used to express the requirements of a negotiating party during the negotiation processes.

There has been a wide range of research conducted on improving SLA negotiation in cloud computing using meta-heuristic algorithms. Much of this has explored SLA formation, management, negotiation, and monitoring. Most work on SLA negotiation in cloud systems has focused on the IaaS level, and has explored how to improve communication between consumers and providers (Pittl et al. (2015)). Negotiation at the IaaS layer is concerned with virtual/ physical resources and their characteristics,

and is essentially conducted to enhance resource allocation. Resource providers must efficiently manage their infrastructure to run consumer tasks, satisfy consumers' needs, and maximize providers' profits. To achieve this, resource providers need to negotiate with consumers before allocating resources based on the agreed issues. The solutions then developed can be grouped into approaches based on multi-agent systems, PSO algorithms, or on parallel meta-heuristic algorithms. In subsequent sections, an overview of research on all these approaches will be provided and their respective benefits and drawbacks will be discussed.

2.3.2 Automated SLA Negotiation Based on Multiple Agents

Many researchers have proposed different ways of using a multi-agent system for automated negotiation. Using a multi-agent in developing negotiation strategies is suitable for application in a distributed environment such as cloud computing (Chen et al. (2014)). Specifically, (Chen et al. (2014) designed a multi-agent based negotiation framework using a dynamic model that incorporated several factors of the negotiation process such as degree of competition, time of negotiation, number of rounds, and historical information on trade.

Several researchers have also explored automated concurrent negotiation, which is based on a one-to-many form of negotiation (Panagidi et al. (2014)). This strategy involves one buyer and several sellers who express their preferences by exchanging offers and counteroffers (An et al. (2010)). A multi-issue concurrent negotiation mechanism was developed to resolve multi-issues regarding price, time and service quality in cloud computing by (Mansour and Kowalczyk (2012)). Recent research has considered negotiation models for automated SLA negotiation, including strategic behaviors in bilateral negotiations (Silaghi et al. (2012)) or concurrent negotiation ((Nurika et al. (2014)). Concurrent negotiation is a common type of negotiation in cloud computing where the consumer selects the appropriate strategy to be applied by each thread based on the preferences of each interaction (Nurika et al. (2014)). Concurrent negotiations provide many advantages, such as enabling a buyer to negotiate, in parallel, with many sellers (Mansour and Kowalczyk (2012)). This enables buyers to select the best possible agreement. Researchers, aware of the advantages of this

approach, have subsequently proposed a number of models. (Nurika et al. (2014)), for example explored one-to-many negotiation between a buyer and multiple sellers. In this approach, the buyer waits until all the threads send the offers before starting next iteration of negotiation. In (Rahwan et al. (2002)), three methods to coordinate one-to-many negotiation were developed. The information changes between parties after the receipt of an offer and the coordinator decides when to finish the negotiation and how to compute the utility function. Nguyen and Jennings (2003) extended the work presented in Rahwan et al. (2002) and proposed methods to coordinate threads of concurrent negotiation. In Mehdi et al. (2011), the number of steps taken during the negotiation process to reach an agreement was reduced by allowing the broker agent to nominate the offers rather than the consumers. This reduced both the average waiting time and the number of tasks that failed.

To accelerate the negotiation process, (Zhang and Liu (2016)) developed a concurrent automated bilateral multi-issue negotiation mechanism for different combinations of values of a discrete issue. The seller agent and the buyer agent can only respond by varying the price in each thread. This greatly reduced the exchange of information between the two agents and avoided strategic misrepresentations. Furthermore, in (Omezzine et al. (2016)) a negotiation-based scheduling algorithm was developed to deal with both the characteristics of the cloud market and the objectives of SaaS. Their experimental evaluation demonstrates the benefit of including negotiation in the scheduling process.

Finally, in Messina et al. (2014), a cloud negotiation protocol was developed to deal with the problems that arise when having to negotiate SLAs with different providers. They used agent technology to improve and simplify the process of SLA negotiation. Dastjerdi and Buyya (2012) developed a method of automated negotiation based on agent systems. It aimed to offer reliability, balance the load between different VMs, and rank the offers.

In this thesis, the aim will be to automate the process of SLA negotiation by using multiple agents based on multi-issue objectives within the constraints of deadline and cost. The difference in this research is that the focus will be on applying a PSO

algorithm in parallel model to minimize the negotiation time. SLA negotiation based on PSO algorithms will be discussed and analyzed in the next section.

2.3.3 SLA Negotiation based on Particle Swarm Optimization

The general methodology of using meta-heuristic algorithms in negotiation is based on representing the negotiators' proposals and the counter proposals as points within the space of possible agreements (Tsai and Rodrigues (2014)). The algorithms calculate the solution according to the negotiators' objectives, and the utility functions. Recent efforts in automated negotiation have involved the use of meta-heuristic algorithms such as PSO algorithms (Kolomvatsos and Hadjiefthymiades (2014); Panagidi et al. (2014)).

For example, in Copil et al. (2012), negotiation based on the PSO algorithm for creating counteroffers was developed to provide the energy consumed and the performance offered in the cloud. In their research, the swarm represents a potential counteroffer and the energy is used as an objective function. The PSO-based negotiation process evaluates the solution by considering Pareto optimality. The authors argue that providers must manage the trade-offs between the required energy and the charged price during negotiation by using PSO in both the consumer and provider sides.

On the other hand, in Esmaeili and Mozayani (2010), a multi-attribute negotiation model was proposed, based on a multi-objective PSO algorithm that uses agents to conduct a separate negotiation with each opponent where each agent identifies the best offer among incoming offers. The research compared the results of using the weighted sum of multi-objective approaches with the results of an optimal set to demonstrate the model's effectiveness and the required convergence.

Panagidi et al. (2014) developed the PSO approach for concurrent multi-issue negotiation. This was based on the number of threads, which represents the objective issue. However, it is an efficient technique when the number of threads is small because, when the number of issues increases, the profit decreases. In (Maitly and Chaudhuti (2014)), a novel method of SLA negotiation in cloud environments based on a multi-objective GA algorithm was proposed. The model efficiently chooses the most optimized SLAs for inexperienced consumers in cloud environments. The framework

aims to simplify SLA negotiations and matches VM specifications. It offers high performance compared to most current SLA negotiation schemes where the focus is more on profit.

In (Kattan and Fatima (2012)), a combination approach of GA and PSO was developed to improve the bilateral multi-issue sequential negotiation. The PSO is used to balance the computational budget between the two GA algorithms. In our model, when applying PSO algorithm in large scale distributed environments such as cloud computing, the quality of results will be minimized in line with an increase in number of elements to be explored in the search space. In addition, we aim to benefit from the distributed resources to increase the performance of the PSO to speed up the negotiation process. Thus, the emphasis will be on how to apply parallel PSO to deal with large space. In the next section, the methods of applying parallel algorithms in the SLA negotiation process will be discussed.

2.3.4 SLA Negotiation based on Parallel Algorithms

Parallel computing is a computational model involving several computational resources that can run simultaneously (Grama (2003)). Automated negotiations provide many advantages as a buyer can negotiate in parallel with many sellers. This improvement gives the negotiation process the opportunity to select the best possible agreement and reduce the negotiation time. Several researchers have used parallel computing to improve concurrent negotiation strategies by applying meta-heuristic techniques such as GA algorithms. For example, (Sim (2013)) proposed parallel negotiation models that promote negotiation activities between the consumer and provider agents through a broker agent. Multiple broker agent services receive requests from each consumer agent, and many consumer agents send requests to each broker agent.

In (Hashmi et al. (2011)), a GA algorithm was developed for simultaneous web service negotiations. They introduced a parallel GA algorithm to enhance negotiation. Furthermore, (Nurika et al. (2014)) proposed a Genetic optimized performance oriented algorithm for concurrent SLA negotiations in the cloud environment that focuses on aspects such as network speed and execution time to improve overall profit and

performance. In Bousselmi et al (2016), a workflow-scheduling algorithm was developed to improve the QoS by extending the Parallel Cat Swarm Optimization (PCSO) algorithm to provide better results in performance compared to the standard PSO algorithm. In this research, SLA negotiation based on Parallel PSO will be developed to speed up the process of negotiation and improve the results of PSO algorithms (see Chapter 4 for further details).

2.3.5 SLA Monitoring

After the submission of the task to the selected data center, the resource needs to be monitored to avoid any SLA violation and to detect the status of the task. SLA monitoring is therefore an essential requirement for any cloud model as it traces the usage of cloud resources, performance, and ensures the SLA is met. It monitors the execution of the tasks to provide information to cloud providers to allow effective management of data center resources.

Monitoring SLA also plays a significant role in determining the number of violations that have occurred. Most SLA violations happen during load fluctuations or delay time (Emeakaroha et al. (2010); Patel and Sarje (2012)). SLA violation monitoring process starts when an agreement has been initiated. Most importantly, monitoring plays a critical role in determining whether a SLA is achieved or violated.

Consequently, SLA monitoring in cloud computing has been studied by many researchers from different perspectives. Some researchers have proposed the SLA violation approach in one layer of cloud computing, for example, (Emeakaroha et al. (2012); Sakr and Liu (2012)) developed approaches enabling SaaS to detect the SLA violation in applications in the SaaS layer. In fact, most researchers in SLA monitoring have focused on the cloud IaaS layer as a critical and essential asset for SLA violation in terms of security, privacy, quality of hardware, and availability (Emeakaroha et al. (2010)). Generally, cloud providers need optimization algorithms to provide optimal resource allocation to meet the SLA and satisfy QoS (Sahal et al. (2013)). Some researchers have discussed different optimization-based approaches to resource allocation in the cloud environment such as using meta-heuristic algorithms. For

example, (Liu et al. (2011)) used the ACO technique to enable the SLAs to manage the provider's resources. In this research, the focus will be on QoS performance metrics i.e. response time, waiting time, and throughput. The SLA Monitor agent will be used to collect information about QoS metrics from the provider and the consumers. When the SLA agreement is established between the cloud provider and the cloud consumer, the SLA Monitor Agent needs to check the SLA status to detect any SLA violation. Therefore, the focus will be on IaaS layer monitoring to detect whether any SLA violation occurs. SLA violation detection is based on deadline constraints and the VM migration time. The presented SLA monitoring does not include the reliability feature of the cloud computing and not predict the failure before occurrence. Adding this feature for future improvement of the model can improve the reliability of the system and reduce the failure rates.

2.3.6 Discussion of SLA Negotiation Work

Previous research has improved negotiation by using multiple agents to apply automated negotiation, which is particularly suitable for applying in cloud computing. However, there is a need to further improve and simplify the negotiation process in cloud computing to cope with the scalability and increase in the cloud's resources. There are many inherent limitations in current negotiation algorithms that can be discussed in relation to their applications in cloud environments. Such limitations can be summarized as revolving around the need for multi-issues to be simplified and sped up. Moreover, in some cases, the SLA is violated and this needs to be detected before it occurs. Using meta-heuristic techniques produces effective results compared to traditional methods. Parallel negotiation algorithms are superior in terms of negotiation time, number of proposals and average utility when compared with sequential negotiation (Nurika et al. (2014)). However, the parallel negotiation algorithms need to meet several objectives simultaneously. Some parallel meta-heuristic algorithms have been applied to enhance the negotiation process and reduce the time spent on negotiation, for example, GAs in (Hashmi et al (2011); Nurika et al. (2014), and the Cat Swarm algorithm in Bousselmi et al. (2016).

In our research, however, multi-issue concurrent negotiation based on PPSO will be developed to speed up the process of negotiation and reduce the time of migration. To date, no previous research has attempted this. Using PPSO will improve the quality of the results of PSO and speed up the process of negotiation because PSO is an inefficient algorithm when large spaces are involved.

From another perspective, most of the existing research focuses on just the price or QoS. In the proposed model, consumer goals will be focused upon in terms of QoS, time and the provider objective (i.e. price). Thus, in the research reported in the thesis, the concern is SLA negotiation and monitoring, and the focus will be on the multi-issue parallel PPSO SLA negotiation model in cloud computing where a consumer agent negotiates with more than one agent in each data center and each agent is characterized by multiple negotiation issues. In comparison to Copil et al. (2012), PSO will be run in the Manager Module only to reduce communication time and the time spent on the process of negotiation.

2.4 Task Scheduling in Cloud Computing

Task scheduling is a vital process in the field of cloud computing because it affects overall system performance. Traditional scheduling methods are not effective when applied in cloud computing and many researchers have therefore explored cloud computing specifications to develop and optimize task scheduling in the cloud environment. In this section, various scheduling algorithms related to the proposed research will be presented and analyzed from different perspectives.

Specifically, Section 2.4.1 presents a general overview of task scheduling in cloud computing. Section 2.4.2 discusses task scheduling based on PSO algorithms and analyzes the results based on the objectives. In Section 2.4.3, several methods for applying meta-heuristic algorithms will be presented and discussed. Additionally, Section 2.4.4 presents task scheduling algorithms based on heuristic approaches namely Min-min and Max-min algorithms. Real-time scheduling approaches will be discussed in Section 2.4.5. Finally, Section 2.4.6 discusses previous work in task scheduling and summarizes the main points that merit further study.

2.4.1 Overview

Scheduling is the process of allocating or distributing work to processors, humans or machines to be completed within certain time constraints (Zhan et al. (2015)). In the cloud context, task scheduling maps tasks to suitable resources to satisfy specific objectives (Shaw et al. (2014)). Tasks must be allocated efficiently to VMs with minimum delay. The providers must follow the SLAs and meet the QoS requirements as defined by the consumers. One of the important research issues in cloud computing, in terms of performance efficiency, is therefore that of scheduling tasks and resources. Scheduling tasks in cloud environments is classified as an NP-hard problem because it involves large solution space and thus takes a long time to find an optimal solution (Tsai and Rodrigues (2014)). Meta-heuristic algorithms have been applied for task scheduling problems to find sub-optimal solutions within an acceptable time (Tsai and Rodrigues (2014)).

The scheduling function maps tasks to available resources to optimize one or many objectives under certain constraints. Task scheduling can be categorized into two main types: static and dynamic (Singh and Chana (2016)). In the former, the tasks are scheduled in an environment depending on known information about the tasks and resources. In the latter, scheduling depends on the current state of the system (such as load, storage capacity, and network bandwidth), in addition to the submitted tasks. The dynamic allocation changes the decision of the selected resource depending on changes in the system, while static allocation depends on static information and does not consider system changes. In this research, dynamic scheduling is utilized because it is better suited to the elastic feature of most cloud infrastructures (Singh and Chana (2016)). Dynamic task scheduling can be applied in a real-time (online) mode or batch mode ((Mathew et al (2014)). In real-time scheduling, tasks are scheduled immediately when they arrive in the system while in the batch scheduling, tasks are queued when they arrive and then they scheduled (Liu et al. (2010)). From another perspective, tasks can be classified based on dependency factor as either independent or dependent (Pandey et al. (2010)). Task scheduling is called for scheduling independent tasks, and scheduling dependent tasks is called workflow scheduling. Task scheduling is much easier than workflow scheduling because the scheduling process deals with a set of tasks that are independent of each other and they can be executed individually without

any dependency on other tasks (Jayanthi (2014)). On the other hand, task scheduling can be preemptive or non-preemptive depending on the specification of the tasks and resources. In the non-preemptive scheduling, when a task starts running it is not interrupted until it finishes it's execution, while in preemptive scheduling the tasks are prioritized and then are executed depending on their priority.

Several parameters are taken into consideration when optimizing scheduling algorithms in cloud computing, some related to providers whilst others benefit the consumers (Tsai and Rodrigues (2014)). The main parameters that summarize consumers' goals are performance and QoS (Zhan et al. (2015)), the details of which are as follows:

- **Performance**: the efficiency provided by the scheduling algorithm in terms of providing services to consumers based on their requirements. The main attributes of the performance factor are:
    - **Waiting Time:** the time the task takes from submission to end execution. The waiting time should be reduced to increase the performance of the scheduling algorithm. Improving performance and reducing energy consumption are achieved by reducing the waiting time.
    - **Response Time**: the time that elapses between the submission time of the task and the start of execution.
    - **Execution Time**: the total time taken to execute the task from the start until it is completed. The aim of scheduling algorithms is to reduce the execution time.
    - **Completion Time**: the total time taken to complete the execution of a task. It also includes the execution time and the waiting time caused by the cloud system (Mathew et al. (2014)). Many scheduling algorithms prefer to minimize the completion time of tasks.
- **Quality of Service**: this includes various constraints on consumer input such as meeting execution, cost, performance and deadline, which are defined in SLAs contract document.

- **Budget**: This constraint is defined by the cloud consumer and determines the maximum cost that can be paid for services. The scheduling policies are made to minimize the total execution time within the budget (Poola et al. (2014)).

- **Deadline**: is the defined as the allowable time for execution time from submitting a task until the time of completion. A good scheduling algorithm always tries to ensure tasks are executed within the deadline constraints to reduce the number of failed tasks (Mathew et al (2014)).

- **Cost**: indicates the total amount the cloud consumer will pay to the cloud provider for using the resources. The cost of services depends on the computation cost, the cost of transferring data, and the storage cost.

The parameters related to providers' goals can be summarized as follows:

- **Resource Utilization**: measures the resource used to increase the throughput of the system. It is used to keep the resources as busy as possible to maximize the profit.
- **Load Balancing**: is the technique of distributing the load in a cloud data center across different nodes so that no node is under-loaded at any given time. The load should be balanced to increase the utilization of resources. Load balancing over the resources also improves resource utilization.
- **Energy Consumption**: many different scheduling algorithms have been designed to reduce power consumption to improve performance. Reducing energy consumption in cloud data centers is an issue that has recently been considered in several scheduling algorithms. Energy consumption will be affected by utilization such as it becomes high when CPUs are not utilized because idle resources are not used effectively.

In the following subsections, the work undertaken in task scheduling in cloud computing according to these goals will be briefly discussed. The focus will be on methods that apply PSO and previous work based on meta-heuristic, heuristic task scheduling, and real-time scheduling strategies will be discussed.

2.4.2 Task Scheduling Based on Particle Swarm Optimization

Many scheduling algorithms have been proposed for task scheduling in cloud systems. In this thesis, the focus is on research that uses PSO in task scheduling. Many researchers have explored how to use PSO to improve task scheduling in cloud computing using different strategies. PSO is one of the most successful meta-heuristic algorithms for generating optimal solutions by scanning the search space during each iteration and evaluates solutions. This section presents a review of recent work on the use of PSO in task scheduling in a cloud environment. A classification of the scheduling schemes will be presented, based on their objectives for both consumers and providers, and on the numbers of these objectives, i.e. whether they are single, multi-objective or many-objective.

First, task scheduling based on consumers' goals is considered. In this instance, consumers' QoS requirements include several parameters such as makespan and cost. In multi-data centers, there are additional parameters added for network and communication such as distance, bandwidth, and latency.

In Abdi et al. (2014) three meta-heuristic approaches for task scheduling in a cloud environment were compared: PSO, a GA algorithm and a modified PSO. In this scenario, the researchers argued that the number of tasks was greater than the number of resources so tasks could not be migrated to different recourses. When compared, the performance of the modified PSO in which the SJFP (smallest job to fastest processor) algorithm is merged with PSO was better than other techniques in terms of minimizing makespan in task scheduling and thus improves performance.

The PSO algorithm are used to develop task scheduling based on multi-objective optimization. MOPSO has therefore been proposed to evaluate multi-objective optimization problems using Pareto set. Lakra and Yadav (2015), for example, developed a multi-objective task scheduling algorithm that scheduled the tasks to VMs to improve the throughput and reduce the cost without violating the SLA. This model used non-dominated sorting after ranking the tasks based on QoS and VMs. Feng et al. (2012) were concerned with improving resource allocation and developed a MOPSO algorithm by using Pareto-dominance, which searches for optimal scheduling based on total task execution time, resource reservation, and the task's QoS. (Milani and Navin

(2015)) improved task scheduling using MOPSO according to three factors: reduced execution time, waiting time, and missed tasks.

Alternatively, some researchers have focused on improving two objectives such as time and cost (Ramezani et al. (2013); Pandey et al. (2010)). Specifically, Ramezani et al. (2013) developed a MOPSO algorithm for optimizing task scheduling in relation to execution time, task transfer time, and task execution cost. However, in Pandey et al. (2010), the researchers used the cost for both data transmission and computation to minimize the total execution cost in cloud computing environments. Moreover, in (Wang and Zheng (2011)), the MOPSO was applied to optimize the cost of task execution, transfer time and task execution time. All the previous methods used the Pareto set in evaluating multiple objectives. In Zhao et al. (2015), task scheduling based on the completed time and cost was improved by using a weighted sum to develop the multiple-objective function based on QoS parameters. A similar method was used in (Beegom and Rajasree (2014)), whose multi-objective PSO scheduling algorithm was based on makespan and communication cost objectives using a weighted sum approach. (Guo et al. (2012); Verma and Kaushal (2014)) used the same parameters in terms of execution time and cost but differed in their method of evaluation function. For instance, Verma and Kaushal (2014) used a weighted sum approach while Guo et al. (2012) used the optimal Pareto set approach.

The second set of approaches focus on task scheduling based on provider efficiency. Provider efficiency in this respect is related to the profit, utilization of the resources, SLA violation, load balancing among VMs, and the cost of power consumption. In this section, the work that will be presented aimed to improve provider objectives.

Some researchers improved task scheduling by taking energy consumption and profit as objectives function, for example (Jena (2015)). The same objectives were considered in (Liu et al. (2013)). They developed a PSO task scheduling model for distributing tasks over VMs to minimize the cost of task execution and maximize the providers' profit in the cloud environment.

In Awad et al. (2015), task scheduling was improved to increase the utilization of resources using a dynamic PSO scheduler. These researchers proposed a modified PSO

algorithm called load balancing PSO (LBMPSO), which aimed to minimize reliability, execution time, transmission time, and cost.

The third set of approaches consider task scheduling in terms of the goals of both providers and consumers. Thus, many researchers have investigated how to compromise and balance the needs of providers and consumers to satisfy both sets of objectives. For example, Al-maamari and Omara (2015) proposed a dynamic adaptive PSO (DAPSO) algorithm to enhance the performance of basic PSO to schedule tasks for minimizing makespan and maximizing the utilization of resources. Similarly, Zhan and Huo (2012) proposed an improved PSO to reduce average execution time and increase the availability of resources.

Feng et al. (2012) were concerned with improving resource allocation based on a MOPSO algorithm using Pareto-dominance that was based on three factors: total task execution time, QoS and resource utilization. In Al-Olimat et al. (2014), MOPSO task scheduling aimed to improve resource utilization and minimize makespan. Some researchers have developed a MOPSO algorithm by applying a ranking strategy, such as (Alkayal et al. (2016)). Three objectives are involved when computing the fitness function to improve the task scheduling algorithm based on a ranking strategy.

On the other hand, there has been comparatively little research studying many-objective. Amongst the work that has been conducted, (Ye et al. (2017)) considered a four objectives scheduling problem based on an improved knee point driven evolutionary algorithm. In contrast to previous research, our research aims to handle five objectives in task scheduling based on a modified ranking strategy to simplify the evaluation of objective function.

When focusing on the type of optimized parameters, it is clear there are many parameters for scheduling algorithms in cloud environments. The most common parameters that are used to evaluate the performance of the scheduling algorithm are execution time, makespan, cost, energy consumption, QoS, and load balancing. Each algorithm addresses one or more of these parameters depending on its objectives. Some researchers also targeted cost optimization whereas others aimed to shorten the makespan. Additionally, some researchers strived to minimize the overall energy consumption, although these algorithms struggled to meet the QoS. Others aimed to

improve the cost, makespan and execution time. However, research efforts regarding load balancing and QoS need more work regarding the number of objectives because, as noted previously, most research focuses on multi-objective, which include just two or three objectives. Additionally, there is little in the way of research focusing on evaluating many objectives (more than three objectives) when scheduling tasks in cloud computing environments.

2.4.3 Task Scheduling Based on Meta-Heuristic Algorithms

Researchers have demonstrated that meta-heuristic scheduling algorithms provide sub-optimal scheduling results than traditional scheduling algorithms (Tsai and Rodrigues (2014). Given this, different meta-heuristic algorithms have been applied to solve task scheduling in cloud computing such as GAs and ACO algorithms (Kaur and Chhabra (2016)).

First, task scheduling based on ACO Algorithms will be considered. Many scholars have studied task scheduling using ACO algorithms. For example, Wen et al. (2012) proposed a task scheduling algorithm based on an improved PSO, which considers the total task completion time and the total task cost, but does not consider load balancing in the system. In (Tawfeek et al. (2015)), the concept of ACO is used to schedule tasks in a cloud computing environment based on makespan. Alternatively, Madadyar and Bagherzadeh (2011) have introduced the method of initial ants based on the standard deviation of the pheromone of the tasks and the expected time to execute a task on a given VM. Chen and Zhang (2009) developed a task scheduling algorithm based on ACO to minimize the total cost in specific time within deadline constraints. Wen et al. (2012) proposed the ACO algorithm combined with PSO algorithm to improve the performance of task scheduling. This enhances the convergence speed and increases resource utilization ratio. Additionally, it prevents any descent towards a local optimum solution.

A second approach to task scheduling is that based on Genetic algorithms. Several researchers aiming to optimize multi-objective task scheduling in cloud systems have used GAs. For instance, (Arfeen et al. (2011)) demonstrated that effective scheduling

for independent tasks can be completed using GAs in cloud computing environments. The GA in the scheduling tasks algorithm was developed by Zhao et al. (2009) to improve both resource and time utilization, so that the result obtained provides high satisfaction for their objectives. In (Dasgupta et al. (2013)), they developed a scheduling strategy for the load balancing of VMs using a GA. In their research, the scheduling strategy looks for the best solution by using a GA in each schedule. This method provides better load balancing and resource utilization than the static method of resources allocation. In Ying et al. (2009), two modified algorithms of PSO were used in resource allocation and the results were compared with a GA. The research demonstrated that PSO gives better results compared with a GA algorithm. Moreover, according to research conducted by (Pongchairerks (2009)), PSO was found to be better than GAs in most cases. Furthermore, according to Mirzayi and Rafe (2013), PSO was found to be faster and simpler than GAs in terms of the execution and implementation of independent tasks. Therefore, this work will focus on improving the PSO algorithm.

2.4.4 Task Scheduling based on Heuristic Algorithms

Different heuristic algorithms have been proposed for scheduling tasks in cloud environments (Mirzayi and Rafe (2013)). In this section, the heuristic strategies related to the proposed research will be presented. They schedule tasks based on predefine parameters such as completion time and execution time. The Minimum-Minimum Completion Time (Min-min) and the Maximum-Minimum Completion Time (Max-min) are the most heuristic algorithms used in scheduling (Mirzayi and Rafe (2013)).

The minimum completion time (MCT) algorithm maps each task to the VM that has the minimum completion time (Munir et al. (2007)). Conversely, the minimum execution time (MET) algorithm maps tasks to VMs based on the minimum execution time for that task irrespective of availability of the resource (Munir et al. (2007)).The Min-min algorithm begins with determines the MCT for each task on all resources, and then schedules each task to the minimum MCT (Aissi et al. (2005)). The main objective of the improved Max-min algorithm is to assign a task with MET to a resource, which provides MCT (Mao et al. (2014)). In Min-min, the shorter tasks are executed first, so if the number of shorter tasks is fewer than the number of longer tasks then the Max-

min is used (Mao et al. (2014)). However, the Min-min algorithm does not involving loading balance technique while the Max-min algorithm may offer a better load balance among the resources in cloud environments.

In this work, heuristic algorithms will not be used because they focus on minimum completed time and total execution time, and there are other factors that need to be taken into consideration, such as cost and transfer time. The VM load is an important factor because it affects the performance of the system. In addition, the cost factor is required for the consumers in that they want a reduced cost whilst the provider is concerned with profit. The Max-min and Min-min algorithms will therefore be considered in detail to assess their results in relation to the proposed research.

2.4.5 Real-Time Task Scheduling Algorithms

Real-time task scheduling strategies should ensure that tasks could be completed in accordance with deadline constraints. In this section, several real-time scheduling algorithms in cloud computing will be discussed. In real-time cloud applications, the consumers and the providers must have a strong SLA to control the timing of applications and ensure that the deadlines for tasks are met (Zhan et al. (2015)). In the context of cloud computing, deadline means meeting the consumer's requirements, as well as QoS and SLA within the constraints of a specified time (Mathew et al. (2014)). Several researchers have discussed deadlines in task scheduling algorithms while others view the cost budget as a constraint on the scheduling algorithm. The primary objectives of real-time scheduling are to increase throughput and minimize waiting time rather than meeting deadlines.

The Earliest Deadline First (EDF) algorithm assigns priorities to tasks then the task with the shortest deadline is the one that is scheduled (Liu et al. (2010)). EDF is a form of dynamic scheduling in such that if a scheduling event occurs then the queue will be searched for the process that is closest to its deadline. The selected task will then be the next scheduled for execution. EDF is more popular in real-time research because the principle of the EDF algorithm is very simple to understand and implement. In Gupta et al. (2014), a Priority EDF Scheduling method was used involving two task scheduling

algorithms, one of which was EDF and the other was a priority based scheduling algorithm.

He et al. (2014) proposed an algorithm to achieve real-time task scheduling and develop cloud computing resources. In their research, the degree of resource load balancing and task completion time were objective functions. Multi-objective PSO was used to achieve task scheduling. The deadline guaranteed scheduling algorithm proposed by Shin et al. (2015) enhances the guarantees of deadline and resource utilization. The first algorithm receives all tasks that have arrived at the data center, and then sorts those tasks in ascending order depending on their priority, which is assigned according to the deadline. The research described in this thesis will develop a model of task scheduling that meets both the consumers' deadline and the cost budget.

2.4.6 Discussion of Task Scheduling Algorithms

Many task scheduling algorithms have been used in cloud environments. These can be divided into three categories and are presented below:

1. **Real-time algorithms:** These include that approaches schedule tasks with time constraints.

2. **Heuristic algorithms:** These techniques find the optimal or near optimal solution by using a sample space of random solutions. The Min-min and Max-min algorithms, previously discussed, are examples of these.

3. **Meta-heuristic algorithms:** These algorithms use a random solution space for scheduling the tasks, however the main difference between heuristic and meta-heuristic methods is that heuristic methods are problem specific while meta-heuristic methods are problem independent (Masdari et al. (2016)). They generally use population-based concepts inspired by the social behavior of insects and include PSO, ACO and GAs algorithm.

Regarding task scheduling optimization, there are many studies that have developed the optimization of scheduling in cloud computing. However, most of the previous research on multi-objective optimization is based on objectives that do not conflict with each

other. Specifically, these studies apply single objective optimization to solve their problems. The studies then combine previously optimized objectives into a single objective, and treat them as a single objective using a weighted sum equation. This is an inappropriate approach for dealing with many objectives because our work will be based on five objectives while most approaches work well with three objectives.

As discussed in Section 2.4.2, PSO has already been used to improve scheduling in the cloud. For example, PSO that was utilized to satisfy one objective was shown in research by (Suresh et al. (2014); Wang et al. (2014); Pacini et al. (2014)). Other researchers have used multi-objective PSO to achieve different objectives and have combined them into a single objective (Adamuthe et al. (2013); Moorthy et al. (2014)). However, this approach of combined multi-objectives is inefficient especially with conflict objectives because objectives have positive and negative values. Therefore, an effective method is needed to deal with these objectives separately. Thus, the possibility of extending the process of evaluating objective functions in a many-objective PSO algorithm will be explored to find the best solutions for many objectives.

Based on the research that has applied PSO algorithms to schedule tasks in cloud environments, the main points for further study can be summarized as follows:

1- The quality of using PSO algorithms can be improved by redesigning the operator and improving the initialization step of the swarm. This can be achieved by using local search techniques or heuristic algorithms such as Min-min (as discussed in Section 2.4.2).
2- More work is needed to overcome some of the difficulties of PSO. For example, by combining it with another population-based meta-heuristic technique or a local search technique to improve the quality of results (Madni (2016)), as discussed in Section 2.4.3.

3- Several areas still need to be addressed when applying PSO to optimize task scheduling problems in cloud computing (as discussed in Section 2.4.2). These areas include:
    - Applying dynamic scheduling based on the consumer budget so resources can be located and released according to consumer need. PSO needs to be

applied with QoS when establishing an SLA between the consumer and the provider (as discussed in Section 2.4.1).

- PSO needs to be improved and applied to the load balancing problem, the energy optimization problem, and VM placement and migration.
- Scheduler algorithms in cloud applications need to be scaled with an increase in the number of requests and resources in the cloud infrastructure. Furthermore, the distribution of cloud resources is another issue that needs to be considered. Therefore, PSO applied to these properties of the cloud needs to be improved by using the distributed capabilities of PSO to apply Parallel PSO (as discussed in Section 2.4.2).
- Most of the current research focuses on the cost of using resources by considering the processing resources. However, there are other applications involved including storing and transferring a large dataset. Thus, there is a need to develop scheduling methods that consider the storage issues for applications. In addition, execution time and storage cost trade-off need to be improved and evaluated (as discussed in Section 2.4.2).
- Applying new strategies for providing effective communication and sharing information between multiple data centers should be studied (as discussed in Section 2.4.1).
- Applying PSO with real-time algorithms is a concern that requires greater scrutiny through research (as discussed in Section 2.4.5).
- Most work on scheduling tasks is based on two or three factors. However, there are many factors that can be used to evaluate resources in cloud computing. There is a pressing need to enhance the method of evaluating many-objective optimization to deal with the increased numbers of objectives and compromise the conflicts inherent in the best method (as discussed in Section 2.4.2).

2.5 Virtual Machine Allocation

Virtual machine allocation is the process of mapping a virtual machine to the most suitable host (Pietri and Sakellariou (2016)). There are many hosts and each can run

several VMs in cloud computing infrastructure. Mapping VMs among hosts in an efficient manner is a complex function especially when the number of VMs and hosts increased. VM allocation is a key role in cloud management because it directly affects system performance. Specifically, VM allocation involves mapping between hosts and VMs (Adrian and Heryawan (2015)). VM allocation and migration are an integral part of any resource allocation algorithm in cloud data centers and therefore, our research aims to improve VM allocation and migration.

In cloud computing, VM allocation is responsible for selecting resources and scheduling tasks so that the consumer's requirements and provider's goals are met ((Pietri and Sakellariou (2016))). In general, the main requirement of consumers is to minimize response time while the provider's goals are to maximize resource utilization and profits. The VM allocation's main goal can be either to maximize the usage of available resources or conserve power by being able to shut down idle resources (Shankar and Bellur (2010)).

The VM allocation problem is a type of optimization problem and several optimization techniques are used to address it such as deterministic, heuristic and meta-heuristic algorithms (Lopez-Pires and Bar´an (2015)). Deterministic algorithms include the optimization techniques that follow the same steps at each iteration and provide the same results such as linear programming, binary integer programming and constraint programming. Heuristic algorithms are those used in VM allocation such as First Fit (FF), Best Fit (BF), and First Fit Decreasing (FFD) (Lopez-Pires and Bar´an, (2015); Mastelic et al. (2014)). Meta-heuristic algorithms are those that solves problem with certain constraints by using randomness such as GA, ACO and PSO. In this thesis, the focus will be on PSO algorithms and their variants duo to the advantages they provide in solving problems, as discussed in Section 2.1.2.1.

Specifically, VM allocation includes two main processes: VM scheduling and VM migration. VM scheduling involves mapping the VM to the host whilst migration transfers VMs from one host to another to satisfy specific objectives. VM allocation has been viewed as an optimization problem by researchers such as (Shah et al (2013); Panchal and Kapoor (2013)). On the other hand, research has also been conducted where VM allocation has been broken down in to the separate problems of VM

scheduling and VM migration (Xu and Li (2011)). In the following sections, the two methods will be reviewed in detail and research in this area discussed.

In Section 2.5.1, VM scheduling will be discussed in detail and the main objectives in developing it will be presented. The VM migration process will be described and presented in Section 2.5.2. Section 2.5.3 discusses the strategies of applying load balancing during the allocation of resources. Section 2.5.4 discusses related work on VM allocation. Finally, Section 2.5.6 summarizes the key points and outlines the issues that need to be discussed and studied in VM allocation.

2.5.1 Virtual Machine Scheduling

Many beneficial advantages of cloud computing such as scalability, load balancing and flexibilities because of applying the virtualization technology in term of VMs (Mastelic et al. (2014)). These VMs are scheduled to a set of hosts, which is known as VM allocation. Automating the process of virtual machine scheduling has become a necessity due to a growth in the number of data centers. Generally, the VM allocation method aims to maximize the utilization and minimize the number of idle resources.

VM Scheduling can be applied to the allocation of new VMs to an appropriate host or for reallocating VMs that have migrated from one host to another. The VM scheduling process involves categorizing the virtual machines' characteristics and resource requirements, the utilization of resources and the allocation goals. VM scheduling goals can be single objective such as maximize the utilization of resources, saving power consumption and cost reduction. Some approaches to VM scheduling have been developed to achieve two or more objectives. Nowadays, VM allocation includes more than three objectives because several objectives need to be taken into consideration (Lopez-Pires et al. (2016)). The main policies of VM scheduling according to their concerns can be summarized as follows:

- **Efficient Resource Utilization:** Improving resource utilization and decreasing communication overheads are the most promising topics in managing resource at the IaaS layer. Unutilized resources on each data center may vary largely

with different VM scheduling solutions. The resources should be utilized in an efficient method to increase the total profits.

- **Efficient Power Consumption**: This concerns the scheduling of VMs for a small number of hosts to reduce overall power consumption. It seeks to minimize the total power consumption of the data center. VM scheduling is one of the most important and efficient forms of technology for reducing power consumption in the cloud. The proposed method of rescheduling the power of VMs efficiently is to place them on only part of the hosts and transform the others into a low power state (sleep or off).
- **Cost Reduction:** This aims to optimally schedule the VMs over the hosts to reduce the overall cost. In addition, reducing the cost of the power consumed will reduce the overall costs. Meeting the QoS and SLA prevents violations and reduces the penalties, which improves profit.
- **Efficient Load balance:** Load balancing is very challenging in cloud computing due to the dynamic changes in the resource requirements (Shah et al. (2013)). An efficient load balancing approach can reduce the number of migrations and energy consumption by minimizing the number of active resources. It is responsible for distributing the dynamic workload evenly across all the resources in the entire system to avoid over-loaded or idle resources (Shaw et al. (2014)).

The extent to which VM allocation can improve these objectives will now be discussed. First, VM will be considered in terms of effective resource utilization. A multi-objective ACO algorithm to place VMs was proposed by (Gao et al. (2013)). The objectives to be met were minimization of the total resource utilization of hosts and the number of VM migrations. In (Zhong et al. (2010)), an improved GA algorithm was designed that could assign VMs efficiently, enabling the maximum utilization of available resources. Consequently, the physical resources reached the maximum usage rate and thus the number of physical resources decreased.

Second, VM scheduling for power conservation is considered. Quang-Hung et al. (2012) proposed a power-aware VM allocation algorithm that represents several combinations of FF and the shortest duration time heuristics. However, their VM allocation algorithms did not lead to an optimal solution because they run as an FCFS

algorithm. Given this problem, several researchers have explored how to modify the FF algorithm to find the optimal solution. Specifically, (Lu and Zhang (2015)) developed a Modified Best Fit Decreasing (MBFD) algorithm. The MBFD algorithm takes a sorted list of VMs as input and migrates them in descending order based on their host utilization. It allocates them to the selected host that provides the smallest remaining processing capacity. Thus, this algorithm ensures high utilization of resources as none of them will be idle. Wang et al. (2013) implemented PSO to solve an energy-aware VM scheduling optimization problem in the cloud data center.

Buyya et al. (2010), on the other hand, proposed a model of power consumption based on a correlation between the energy of CPU utilization and the time of the work. Furthermore, (Bohra et al. (2010) developed a correlation model between consumption of power and resource utilization. Several researchers have also attempted to minimize power consumption in cloud environments by virtualization that involves applying VM migration to optimize the utilization of resources (Ye et al. (2010); Luo et al. (2012)).

Third, VM scheduling for cost effectiveness is considered. In Mark et al. (2011), a hybridized approach combined the GA, ACO and PSO algorithms for efficient scheduling of VMs on physical resources. The authors reported that the Evolutionary Optimal VM Placement (EOVMP) algorithm could provide a near optimal solution for stochastic problems and the prediction of the demand forecaster exhibited acceptable efficiency. In (Lee et al. (2010)), a GA approach was developed that used topological information to schedule VM resources. Additionally, a prediction engine was employed to take advantage of topological intelligence and for performance evaluation. The target was to decrease the total finishing time of an application, which automatically results in price reduction.

Finally, load balancing can be applied at many levels such as the data center level, hosts level, VM level and the task level. Load balancing at the level of VM scheduling deals with the assignment of VMs on relevant hosts to balance the load on each host. VM scheduling plays an important role in balancing the load of the system so that the resource utilization is increased. This policy therefore tries to balance the load in order to utilize the resources in an efficient way and minimizes the difference between the

loads. In this research, the focus will be on balancing the load between hosts inside the data center.

Some SI approaches, such as ACO and PSO, have also been used to schedule VMs over available hosts. In (Lu and Gu (2011)), for example, an ACO approach was introduced to find the closest idle or under-loaded cloud resource quickly, and for sharing the load of an over-loaded virtual machine flexibly. For optimal identification of hosts and load sharing of over-loaded virtual machines, the behavior of ants was adopted. In (Cho et al. (2015)), a hybrid algorithm based on a meta-heuristic approach was proposed for load balance oriented VM scheduling in the cloud environment using a combination of PSO and ACO.

Research by (Patel and Sarje (2012)) explored VM scheduling policies to increase the utilization of cloud resources. Zhao et al. (2016) implemented a clustering based load balancing heuristic using Bayes Theorem, whilst (Panchal and Kapoor (2013)) used a K-means clustering approach for the scheduling of VMs in a cloud computing environment. In this thesis, a new dynamic VM allocation policy is introduced that takes VMs as per consumer requirements and allocates them in cluster form to the available data centers. These clusters of VMs are formed using a K-means clustering algorithm.

In the next section, methods of applying VM migration in cloud computing and their benefits they provide in achieving their goals will be discussed.

2.5.2 Virtual Machine Migration

A VM migration strategy is used in cloud systems to maximize the utilization of resources by moving the VMs from under-loaded and over-loaded hosts to unloaded hosts. Moreover, VM migration reduces the power consumed in the cloud data center by switching off the idle hosts. Several methods are used to migrate VMs from one host to another. These methods differ in terms of the factors that are used for applying migration and the main objectives of migration. Thus, an intelligent and efficient migration algorithm is required to balance the load and improve the utilization and

performance of the system. VM migration algorithm aims to minimize energy consumption, minimize violation of SLAs and reduce the number of hosts active at a given time. In some cases, VM migration can increase the number of SLA violations when it applied without taken QoS performance into consideration as a factor in the migration process. For example, a VM is migrated from one host to another it must transfer its primary memory to the destination host which leads to increase the migration time and waiting time. Additionally, in the transfer process, the requested CPU cannot be delivered, as the VM will be in a transitional state. For this reason, along with the demands of power consumption, the amount of VM migration must be minimal, as this will reduce the number of SLA violations. An efficient VM migration strategy will minimize power consumption as well as minimize the number of SLA violations.

VM migration can be broken down into three sub-problems, the details of which are as follows:

- **Host Detection**: To detect the status of the hosts, which may be generally either over-loaded or under-loaded.
- **VM Selection**: After detecting the over-loaded host, the VMs are selected for migration and several strategies for selecting VMs can be used.
- **VM Placement**: The process of selecting the host to which the VM is migrated. The destination host must not become over-loaded after the placement of the migrated VM.

In the following subsections, VM migration processes are discussed in detail along with the main research conducted on each.

2.5.2.1 Host Detection Strategies

The objective of the host detection algorithm is to recognize when a host is over-loaded or under-loaded. Detection is based on the usage and load of host resources in terms of CPU, RAM, storage and bandwidth. To detect the status of the hosts several methods

are used. In this research, the focus will be on methods using threshold detection and the clustering based on meta-heuristic algorithms.

Some researchers have used threshold concepts in VM allocation to determine the status of machines and tasks. For example, (Lin et al. (2011)) used a threshold-based scheme for allocating resources and distributing the available VMs over cloud requests to improve resource utilization and reduce the usage cost based on changes in the system load.

A single threshold method is based on defining the high limit of the utilization of the host's CPU (Beloglazov and Buyya (2010)). In deploying the VMs, it maintains the usage rate of CPUs below this threshold to reserve idle resources and prevent SLA conflict. The double threshold method involves setting the upper and lower limit threshold of the utilization rate of the host's CPU. Thresholds can be used for detecting the status of the hosts and this can be static or dynamic. For example, Beloglazov and Buyya (2010) introduced the concept of adaptive threshold for VM allocation. Specifically, they used predictions based on VM resource usage to dynamically determine upper and lower utilization thresholds to classify the hosts. However, static threshold policies do not work well in a highly dynamic environment such as cloud computing (Verma et al. (2014)). Thus, a dynamic strategy for detecting status of hosts will be developed in this research.

Selecting the thresholds values based on the resource utilization is particularly important, because the research described in this thesis will study dynamic thresholds, which can define the thresholds dynamically based on the current utilization of the resources to improve the efficiency of the resource scheduling strategy. Compared to previous work in this field, most algorithms only rely on CPU utilization for host detection. Multiple factors will be used to enhance VM migration such as CPU utilization, memory utilization and bandwidth utilization.

In this research, we focus on clustering algorithm to detect the status of hosts. The clustering approach combines hosts into one group based on their state, which may be free, fully loaded, partially loaded, or underutilized. Shindler et al. (2011) proposed a K-means clustering which is accurate and fast approach to deal with allocation of VM issues. (Panchal and Kapoor (2013)), on the other hand, proposed a dynamic VM

allocation algorithm that also uses the K-means clustering method and it mapped the VMs to the nearest clusters.

(Hemalatha et al. (2013)) proposed a honey bee clustering algorithm, which searches for the host that can best serve new VM requests in the manner of a bee. It also provides support for the reallocation of VMs and reduces network latency. Alternatively, (Malathy and Somasundaram (2012)) proposed a novel approach based on a reservation cluster. In this approach, unscheduled VM request tasks are placed into the reservation cluster schedule. The reservation cluster schedules all tasks concurrently, which means less computation time and reduced usage of resources.

(Panchal and Kapoor (2013)) allocated the VMs dynamically in cloud computing applications, using K-means clustering algorithms where the parameter was costs in the data center and clustering was carried out according to the number of data centers. Alternatively, (Veeramallu (2014)) conducted VM allocation using K-means clustering algorithms based on energy saving and the number of data centers; each cluster was then allocated to available hosts on the data center similar to the process outlined by (Panchal and Kapoor (2013)). In this research, clustering based on PSO and K-means will be applied to improve detection of hosts' statuses and the results of migrating VMs.

2.5.2.2 Virtual Machine Selection

VM selection is the process of selecting one or more VMs from a set of VMs in one host to be migrated to another host to balance load or reduce power (Beloglazov and Buyya (2012)). Several VM placement strategies have been developed based on many factors such as utilization, migration time and load. In more detail, (Verma et al. (2014)) proposed an algorithm to solve the selection problem of VMs by migrating from the over-loaded hosts to rebalance the load for all hosts in the data center. Beloglazov and Buyya (2012) have proposed algorithms to solve the problems of detection of over-loaded hosts and VM selection. They proposed that host over-loaded detection algorithms and Local Regression provide the best results compared to other algorithms. For example, a Power Aware Best Fit Decreasing (PABFD) algorithm is used for VM placement, but this is only based on a power consumption metric that specifies the best

host to which the VMs should be migrated. In this research, several parameters are used to allocate and cluster hosts including power, cost, capacity, utilization and execution time.

In Shidik et al. (2016), the use of K-means clustering as a VM selection technique for dynamic VM scheduling has been evaluated. Several attributes, such as VM processing in MIPS and VM memory size, are applied in clustering VMs. Moreover, Median Absolute Deviation (MAD) has been used as a form of over-loaded detection that works before the VM selection mechanism. The results of the experiment show that the number of clusters using K-means can influence energy consumption and QoS in the cloud data center.

2.5.2.3 Virtual Machine Placement

VM Placement selection algorithms are used to determine the allocation of new VMs or to reallocate the migration VMs. Many VM placement approaches based on meta-heuristic algorithms were developed in the cloud environment. Specifically, (Xiong and Xu (2014)) have addressed this issue and presented a model using the PSO technique. Their fitness function is based on the total distance between actual utilization and their best value of utilization, taking into consideration energy efficiency. Wang et al. (2013) solved the same problem using a modified PSO. Their modification consists of redefining the parameters and operators of PSO, implementing an energy efficient local fitness first technique and developing a new two-dimensional particle encoding scheme to achieve better quality solutions. The algorithm is compared with FF, BF and MBFD algorithms. Wang et al. (2013) overcome the energy optimization problem by combining PSO with a FF mechanism with the additional aim of maximizing revenue acquisition. On the other hand, (Beloglazov and Buyya (2012)) introduced an algorithm for VM migration based on three different criteria: migration time, CPU utilization and power consumption. A VM is selected if it requires the minimum time to complete a migration. Although this work highlights multiple objectives, the researchers did not use multi-objective techniques. Instead, they dealt with it as a single objective. This method combines the set of objectives into a single objective by multiplying each objective according to a pre-defined weight. In the work described in this thesis, the

migration will be based on CPU utilization for each host and the general load of the system. In this way, the performance is improved by decreasing the waiting time and the utilization of all hosts is increased.

Some researchers have focused on balancing the load to utilize the resources. These include (Madhusudhan and Sekaran (2013)), who developed a GA VM placement and load balancing to utilize the resources effectively. Several algorithms use the GA approach, which is based on the current available resources on the host and the current demand of the VMs parameters, which are used to make decisions regarding placing VMs on hosts. However, sometimes VM migration algorithms increase the number of migrations. Thus, the current research aims to improve the migration process by minimizing the number of VM migration as possible.

2.5.3 Discussion of VM Allocation Related Work

Many different researchers aiming to improve the methods of selecting hosts to execute the VMs have studied the issue of VM allocation. In particular, several meta-heuristic algorithms have therefore been used to solve VM scheduling problems and optimize energy consumption, profit, resource utilization and load balancing. Researchers such as (Khanna et al. (2006)) have developed VM selection based on CPU utilization load without considering other resource factors such as memory load, storage size and network bandwidth.

Regarding VM migration, previous work presented by (Beloglazov and Buyya (2012); (Luo et al. (2012)) (analyzed in Section 2.5), focused on migrating all VMs to under-loaded hosts. However, this is not always a good strategy, because to reduce migration time not all the VMs should be migrated to under-loaded hosts. In contrast, it is better to check the load after migration and compare it with the load before migration to evaluate the effect of migration because in some cases the migration process has negative effective on the load. Moreover, detecting the status of hosts requires intelligent mechanisms that are more adaptive than thresholds such as meta-heuristic clustering.

Finally, most of the research deals with multi-objective optimization while the VM allocation problem often includes many objectives to be evaluated, leading to greater improvement in many-objective optimization techniques.

2.6 Summary

This chapter has presented background information on cloud computing, SLAs, optimization techniques and resource allocation, which are the main processes that will be drawn upon in this thesis. Firstly, the key concepts of heuristic and meta-heuristic algorithms were presented. Optimization based on particle swarm algorithm was then discussed and the algorithms based on PSO were briefly introduced. These represent the main methods underlying the proposed algorithms for allocating resources in cloud computing. The chapter then presented an overview of SLA negotiation and analyzed key research in this area. The general concepts and models of task scheduling algorithms in cloud computing were then presented. Finally, the VM allocation process was discussed along with work to optimize VM scheduling and migration.

The research on resource allocation in cloud systems still requires further study and improvement. Several existing issues have not been fully addressed while new challenges continue to emerge. Subsequently, an effective resource allocation system is required to achieve consumer satisfaction and maximize the profit for cloud service providers. A great deal of research has been conducted and many solutions have been presented in cloud computing environment in respect to task scheduling and the VM allocation problem. However, several issues and challenges require further research, before an optimal solution that is practical for most cloud environments is found.

Some of the challenging issues raised in relation to previous research are as follows:

1) There is a need to reduce consumer SLA violations when utilizing resources because most of the models reduce performance in order to reduce the cost and improve the utilization (as discussed in Section 2.3.1).

2) There is a need to improve resource allocation strategies to reduce scheduling time and thus to improve the real-time scheduling framework (as discussed in Section 2.4.1).
3) There is an urgent need to handle conflicting objectives such as minimizing the cost for cloud consumers and maximizing the profit for cloud providers. At the same time, it is important for cloud providers to utilize and manage the resources in an efficient manner to reduce power consumption (as discussed in Section 2.4.1).
4) Load balancing is a key concern in managing and scheduling the workload in the cloud infrastructure. This will satisfy several objectives such as meeting the QoS requirements of consumers, maximizing profit and enhancing the usage of resources. To balance the load, VM migration is considered as a means of utilizing the resources efficiently (as discussed in Section 2.5.2).

In sum, the differences in our research compared to previous research can be summarized as follows:

In terms of SLA Negotiation:

- An automated SLA negotiation model based on PPSO will be developed; no previous study has attempted this. PPSO will be developed to enhance the results of PSO by dividing the large search space into small spaces and reducing the time complexity of the algorithm.
- Most of the previous research has focused on the price or QoS. In contrast, in our model, the focus will be on the consumer goal of price and QoS in terms of throughput, waiting time and completion time. Additionally, the provider goals of resource utilization and profits will be considered. This is because balancing these conflicting objectives can improve the performance of cloud services.
- For multi-objective optimization, the optimizer needs to consider Pareto dominance every time it updates particles and stores non-dominated solutions to approximate the Pareto front. In our model, the strategy of using multi-objective based on a weighted sum strategy to reduce the time spent updating the particles is improved by using parallel computing.

- An SLA monitoring function will be developed to detect the number of violations that related to the number of tasks that missed deadlines or migration time.

In terms of Task Scheduling:

- Our work differs from previous work in that it focuses on optimizing task scheduling based on many objectives to satisfy several goals for consumers and providers simultaneously.
- Most existing research on task scheduling dealing with multi-objective optimization evaluates two or three objectives. Specifically, these studies often apply methods that may not provide better quality when there are more than three objectives, as discussed in section 2.1.4. In this work, the modified ranking strategy will be developed, combining two methods to evaluate many-objective optimization, involving more than three objectives, in shorter time than other methods.

In terms of VM allocation:

- A VM scheduling algorithm based on MaOPSO will be developed to deal with many-objective optimization and will use the same strategy used in task scheduling to rank the objectives according to different factors related to VM allocation.
- The load balancing technique will be optimized by applying clustering with PSO and K-means in the hosts of data centers to detect the over-loaded and under-loaded hosts and balance the load.
- A VM migration algorithm based on the results of clustering will be developed to utilize the resources effectively and reduce power consumption.

The specification of the presented model in this thesis and its main structure will be discussed in detail in Chapter 3. Additionally, Chapter 3 will describe all the modules in the model and explains their responsibilities in resource allocation. Finally, the general information on the implementation and evaluation of the model will be provided.

Chapter 3

A Resource Allocation Model

This chapter details the architecture and design of the proposed optimized resource allocation model presented in Section 1.3, which involves three modules. The first section presents an overview of the model and explains its goals and objectives. The second section describes the design issues and constraints on the model and its objectives. Section 3.3 presents the main architecture modules of the model and the responsibilities of each component. Section 3.4 presents information on the implementation and configuration of the proposed model. In addition, it discusses the main parameters for the evaluation used in the experiments. The final section summarizes the main elements of the proposed model.

3.1 Overview

The provider infrastructure model can include one or multiple data centers. The provider architecture in the proposed model consists of multiple distributed data centers that are controlled and managed by one manager. This model was chosen because it offers several benefits such as scalability using many data centers. However, allocating resources in multiple data centers is more complicated than using one data center. There are many factors and parameters that need to be considered such as data center location, network bandwidth, latency, data transfer cost and data transfer time. The proposed model of resource allocation consists of three phases: SLA Negotiation, Task Scheduling and VM Allocation as described in Section 3.1. Each phase of the model aims to improve the specific problem of resource allocation to satisfy certain goals. Specifically, when scheduling tasks at the level of VMs and hosts, the model aims to

apply PSO to optimize SLA negotiation between cloud consumers and the provider. The main goal is to dynamically allocate resources (virtual or/and physical) to execute the tasks requested by consumers in ways that simultaneously benefit both consumers and providers. Consequently, the optimized model will improve resource utilization, maximize throughput and profit and reduce waiting time to enhance performance. Furthermore, it also reduces power consumption by switching off idle resources, which indirectly increases profit by minimizing the cost of power consumption.

In the model, the provider has multiple data centers modeled as a set of DC= $\{DC_1, DC_2, \ldots, DC_d\}$; where d denotes the number of data centers in the provider structure. Each data center contains several hosts and the cost of using data center resources comprises the costs of processing, memory usage, bandwidth, storage usage and power. Each data center $DC_{i=1,\ldots,d}$ consists of a number of physical machines (hosts) modeled as a set of Host = $\{H_{i1}, H_{i2}, \ldots, H_{ih}\}$; where h is the numbers of hosts inside data center i. Thus, in this model, the number of hosts in each data center differs according to its capabilities. Each host is described by a CPU processing speed defined in Millions of Instructions Per Second (MIPS), numbers of CPU cores, the amount of available RAM, size of storage capabilities and the required network bandwidth. In this instance, storage denotes the network storage device that is used to store the data files for the task.

Each host $H_{j=1,\ldots,h}$ can deploy many VMs represented as a set of VM= $\{VM_{j1}, VM_{j2}, \ldots VM_{jv}\}$; where v denotes the numbers of VMs that can be run in each Host j. Different types of VMs associated with data centers will be used in the model. Each VM is characterized by specific properties representing the capabilities of the processing storage and the cost, which includes {*VM type, CPU speed, Memory size, Storage size and Network bandwidth}*.

In our model, the tasks denote the software file that needs to be run in the cloud resources and to return results in the output file. The tasks in this model are independent which means there are no dependencies between them. The task is characterized by specific properties that represent the requirements of the consumers regarding the execution of tasks, which includes:

$\{task\ Id, Task\ name, Task\ length\ (in\ MI), CPU\ speed\ , budget\ cost, memory\ size, storage\ , reqired\ network\ bandwidth,\ input\ file\ size\ \}$. The consumers specify these requirements when they submit the tasks.

Specifically, our model of optimized resource allocation includes many phases, as shown in Figure 3.1. The first phase is SLA negotiation, which is responsible for selecting the data center to execute the tasks and uses Parallel PSO to reduce negotiation time and increase throughput. The second phase is the MaOPSO tasks scheduling algorithm, which maps tasks inside each data center to satisfy many objectives: maximizing resource utilization, increasing profits and reducing waiting time. The third phase is VM allocation, which maps VMs to hosts using clustering based on PSO and K-means to improve the utilization of resources and reduce power consumption. Details of each algorithm and related information are presented in Chapters 4, 5 and 6 respectively.

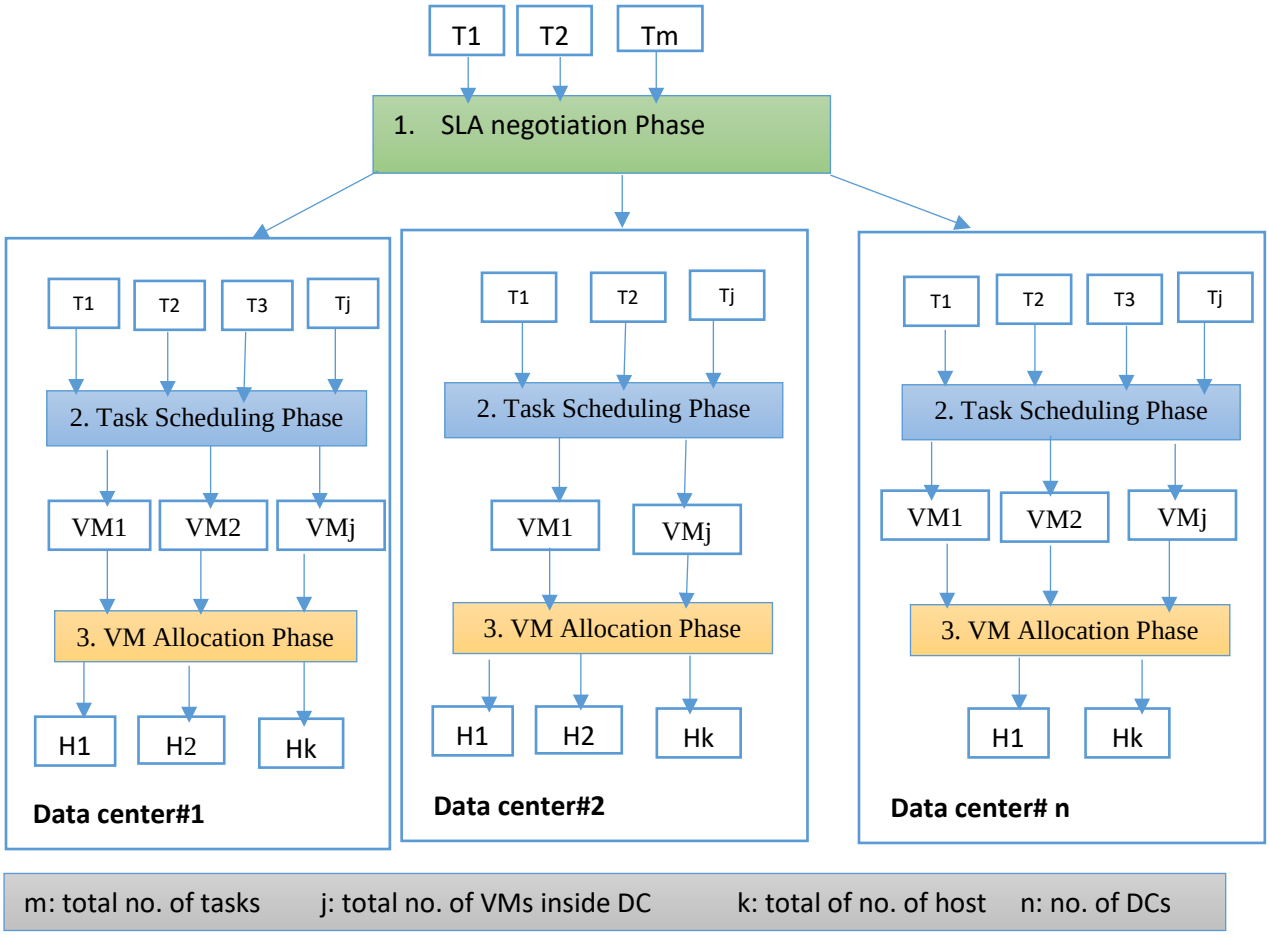

Figure 3.1: Optimized Resource Allocation Phases.

3.2 Design of an Optimized Resource Allocation Model

It is important to identify constraints on the design of the proposed model before discussing its structural detail. The resource allocation model is designed and implemented according to the following constraints:

- The model includes n data centers where each data center has its own resources.
- Each task should be allocated to only one data center at the negotiation phase.
- The task needs to be assigned to one VM, which is compatible with its requirements in terms of CPU power, RAM size, storage size and cost.
- When a task is assigned to a VM, the execution of that task should be completed in the same VM, i.e., the tasks are non-preemptive (see Section 2.4.1).
- The VM can process one task at a time because it is assumed that each VM has one core to prevent the problem of contention when running many tasks.
- The total processing requirements of all tasks hosted on a resource should not exceed the maximum processing capacity of that resource.
- The total memory requirements of all tasks hosted on a resource should not exceed the maximum memory available for that resource.
- The number of host and VMs are not fixed because some of the hosts will be switched off if there is no more work to save power.
- At any time, the total number of virtual machines used in a host should not exceed its capacity, which is dependent on the number of CPU cores and assumes that one core can run only one VM.
- All tasks are independent and the scheduling algorithm assumes there is no communication between them.

The main design goals of the proposed allocation model can be summarized as follows:

- All decisions regarding the allocation of the (VMs/physical) machines should be made automatically based on the proposed algorithms.
- Utilization of the system can be improved by balancing the load to avoid over-loaded and under-loaded hosts. This ensures the efficient use of resources.
- Dynamic resource allocation in the proposed system is used to adhere to QoS requirements and reduce SLA violations.

- Reducing the time required for mapping and allocation to reduce the waiting time and improve performance
- The task should be finished before the deadline to improve throughput and increase profit.
- The profit is increased by maximizing the utilization of the resource and reducing the number of SLA violations.
- Reduce power consumption by improving the migration method to turn off idle resources.

3.3 General Resource Allocation Architecture

The architecture of the proposed model comprises four main entities: Consumer Agent, Broker Agent, Manager Module and Provider Modules. These main entities represent all layers of cloud computing and each is responsible for a specific function regarding resource scheduling and allocation. Specifically, Consumer Agents in the SaaS layer interface and deal with consumers. The Manager Module in the PaaS layer provides the negotiation and management capabilities of the model for consumers and the provider infrastructure. The Provider Module is in the IaaS layer and manages the infrastructure of the cloud, which includes virtual and physical resources. The general architecture of the proposed model is illustrated in Figure 3.2. The details of each module involved in this architecture are discussed below, along with their functions and responsibilities.

- **The Consumer Agent:** This agent is located on the consumer's side and represents the interface the consumers deal with to submit the tasks. It is responsible for receiving tasks from consumers who submit their specified requirements in terms of cost budget, CPU speed, storage size and deadline. The Consumer Agent then sends the tasks with their requirements to the Broker Agent.

- **The Broker Agent:** This Agent acts as a mediator between the Consumer Agent and the Manager Module. The Broker receives tasks from the Consumer Agent, represents these tasks in the form of proposals, and then sends them to the Manager Module. When consumers submit tasks, they are accepted through the Broker

Agent, which defines the QoS requirements based on the consumer's request. In addition, this Agent collects the results of feedback and acknowledgements and sends them to the Consumer Agent, which is concerned with the state of each task.

- **The Manager Module**: This module is a central module in the proposed architecture. It is responsible for allocating resources according to the information received from the DC Manager Agent in the Cloud Provider module. Interaction between the three submodules is needed to satisfy the objectives of the proposed model. The submodules of the Manager are the SLA Monitor Agent, The Negotiation Agent and the Task Dispatcher Agent, as shown in Figure 3.3. The Manager accepts proposals from the Broker Agent through the SLA Negotiator Agent and receives offers from the DC Manager Agent in each data center. Then the negotiation process begins in this module based on the predefined objectives of the providers and consumers. In addition, it manages the execution of the consumer's task in the data centers of cloud providers. It receives updates on the state of each task from the DC Manager Agents. The Manager submodules are described in more detail below:

    - **The SLA Negotiator Agent:** this agent receives proposals from the Broker Agent and offers from each data center through DC Manager Agents. It starts the negotiation by determining the most appropriate offer for each proposal within the constraints of the cost budget and deadline. It selects the data center that can execute a task in line with the specified requirements. It applies Parallel PSO negotiation to conduct the mapping between consumers and data centers. It then creates the agreement form, and signs it on behalf of the consumer and providers. The design and implementation of the Parallel PSO negotiation algorithm are discussed in Chapter 4.

    - **The Task Dispatcher Agent:** The function of this Agent is to assign tasks to the selected data center based on the results from the SLA Negotiator Agent, which decides the data center ID for each task.

    - **The SLA Monitor Agent:** this agent is responsible for collecting information about the agreed SLA and monitoring it to detect any SLA

violation. It is responsible for monitoring the progress of the submitted tasks and detects any violation of the SLA. A violation in this model relates to the deadline time constraint.

- **Cloud Provider Module:** This module represents the IaaS layer in the cloud environment, which includes the VMs and hosts. It consists of many distributed data centers, which in this instance is assumed ten in order to evaluate our model with a large number of data centers. Each data center is a centralized repository for both physical and virtual resources and has five components: DC Manager Agent, Host Monitor Agent, Task Scheduler, Load Balancer Agent and VM Manager Agent (Figure 3.4). The details and responsibilities of each of these components is as follows:
    - **The DC Manager Agent:** It acts as an interface between the data center resources and the Manager Module. It is the local manager for each data center. The DC Manager Agent in each data center sends offers that can satisfy the QoS of the consumer to the Manager Agent. This agent has many responsibilities, periodically sends the status of the cloud to the Manager Agent from the Host Monitor Agent, and receives the list of tasks to be executed from the Manager Module.
    - **The Host Monitor Agent:** It is responsible for monitoring the load in each host and notifying VM migration if there is an unbalanced load among the hosts. It collects information about all hosts and VMs in the data center, which includes the status of the host, utilization of resources, and power consumption. This module also provides information to the VM Scheduler to map the VMs to hosts.
    - **The Task Scheduler Agent:** The Task Scheduler Agent utilizes the latest status information from the Host Monitor Agent regarding VM availability and load. The MaOPSO task scheduling algorithm is used to search the VMs inside the data center to find the best VM for each task based on five objectives: Task Execution Time (TET), Task Execution Cost (TEC), Data Transfer Time (DTT), Data Transfer Cost (DTC) and VM capacity. Details of the MaOPSO task algorithm will be presented in Chapter 5.

- **The Load Balancer Agent:** This module detects over-loaded and under-loaded hosts. It clusters the available hosts to apply the VM migration algorithm inside each data center. It applies PSO based clustering to cluster hosts into four classes: over-loaded, high-loaded, under-loaded, and unloaded. Details of the VM migration algorithm are provided in Chapter 6.

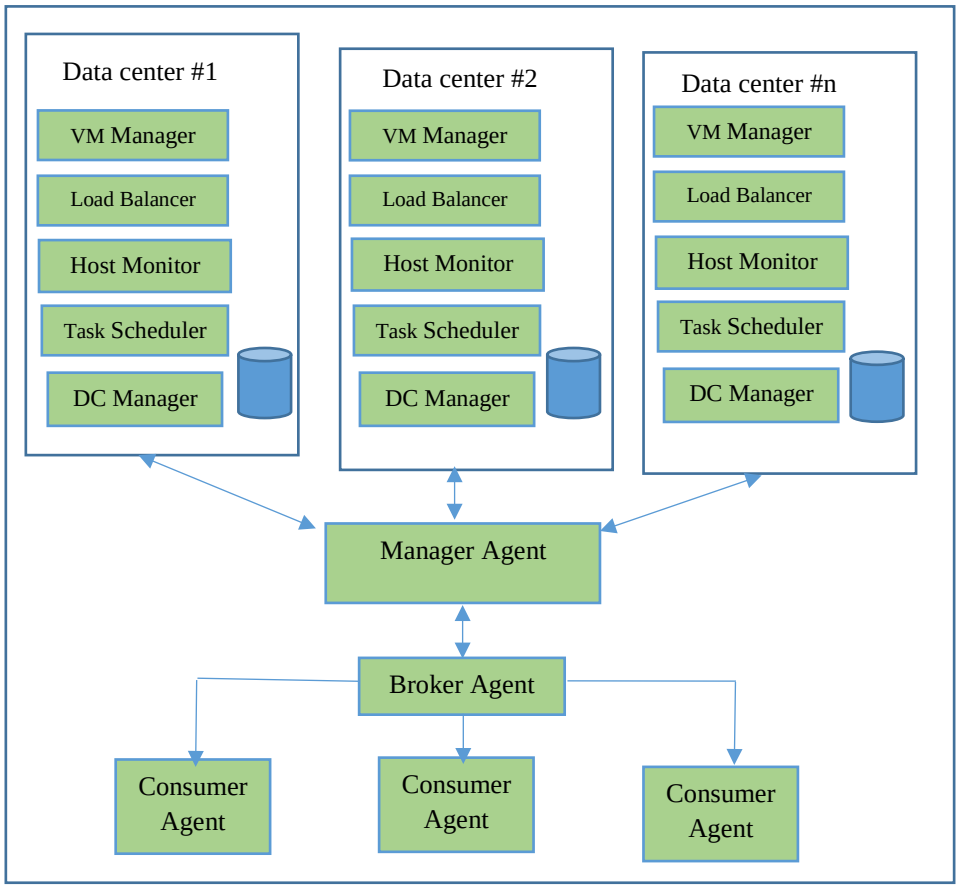

Figure 3.2: General Architecture of Resource Allocation in Cloud

- **The VM Manager Module:** This module is responsible for allocating the VMs inside the data center. It begins the VM migration process based on the information from the Host Monitor Agent. The VM Manager Module includes two submodules: VM Scheduler Agent and VM Migrator Agent. The details of these are as follows:
    - **The VM Scheduler Agent:** This Agent is responsible for allocating VMs over hosts inside the data center. It uses MaOPSO to allocate VMs to reduce the time and improve throughput based on four

objectives. This algorithm maps the VMs to the available hosts**.** It periodically collects information from the Host Monitor Agent to make decisions on the placement of VMs.

- **The VM Migrator Agent:** This is responsible for starting the migration of VMs from unloaded and over-loaded hosts to unloaded ones. It triggers the migration of VMs to hosts depending on the information provided by the Host Monitor Agent.

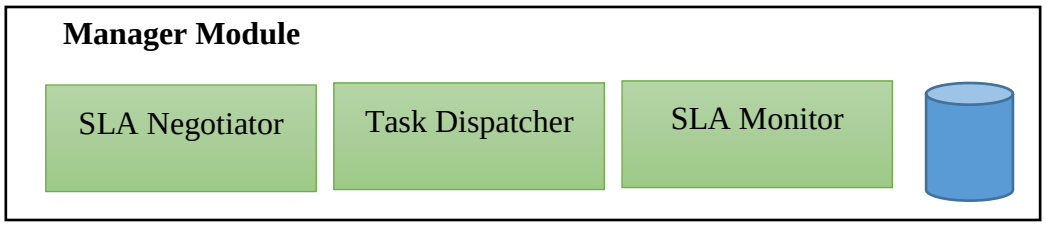

Figure 3.3: Manager Module Architecture.

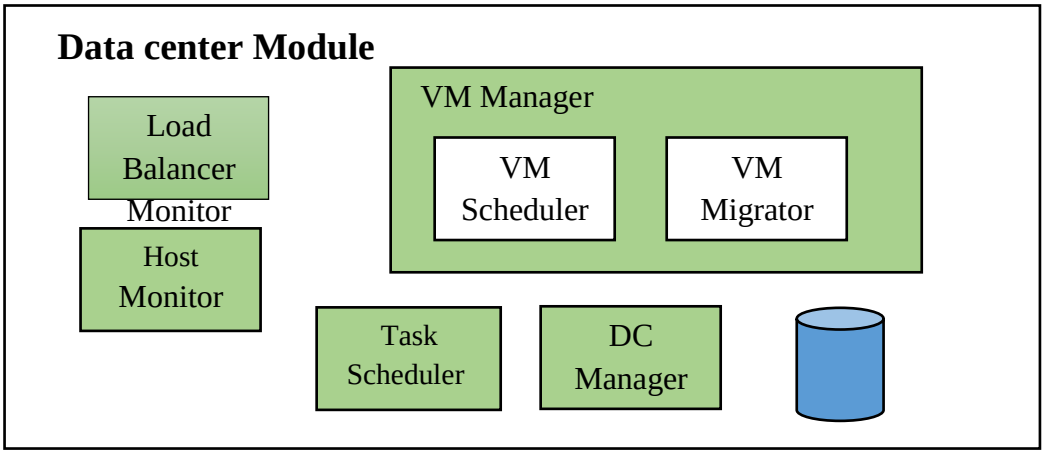

Figure 3.4: Architecture of Provider Module.

- **Cloud Database:** it includes information on the data centers, hosts, VMs, and tasks. It consists of static information such as specification of the hosts and VMs and dynamic information on change regarding the status of the system such as host load and VM utilization. In the proposed model, a distributed database is used, which includes a primary database in the Manager Module and several secondary databases in each data center. Periodic synchronization is needed to update any changes to the primary database. Figure 3.5 illustrates the entity relationship between tables in the proposed database model.

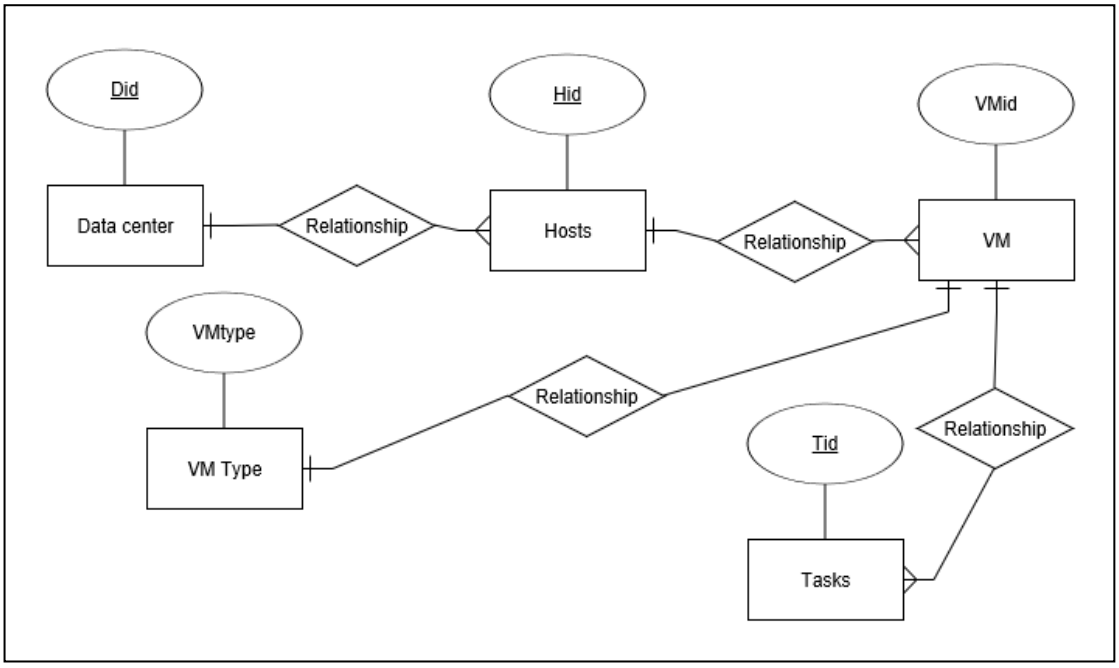

Figure 3.5: The Entity Relationship Diagram for the Proposed Database.

3.4 General Resource Allocation Implementation

This section will discuss the main points related to the implementation of the proposed model, and the main setting of the simulation components that will be used to implement the required modules. In addition, it describes the factors that will be used to evaluate the resource allocation phases. However, the special configuration for each algorithm and its evaluation will be discussed in subsequent chapters.

First, we consider the reasons for using simulation. In cloud computing environments, various resources exist including hardware, software, and the network. In addition, consumers have different dynamic and sometimes competing QoS requirements. Using a real cloud environment, such as Amazon EC2 or Microsoft Azure, to design and evaluate the model's performance metrics with different configurations and settings is difficult because of the limitations of these infrastructures. This is because, in the proposed model, several configurations of the data center with different specifications need to be tested. Moreover, using real environments restricts the evaluation process because there are limits on the infrastructure, and the reevaluation of experiments then becomes very difficult in terms of measurement. Thus, retesting is extremely difficult

and requires additional changes to the environment and infrastructure. Furthermore, it is expensive, costly in terms of budget, and time-consuming to re-configure benchmarking parameters to change the applications and workloads so they can run more tests. Because of these limitations, most developers and researchers prefer to use simulators for benchmarking experiments and evaluations (Sakellari and Loukas (2013)). Using simulation tools and environments give developers many advantages such as testing and retesting the different configurations of the infrastructure in less time and using easier methods. Moreover, using simulators can improve the flexibility of the models by allowing the developer to define a structure that is easy to use, modify and customize depending on the different requirements. For all these reasons, the model will be evaluated using simulator software.

However, there are various simulation tools available for cloud systems, such as CloudSim, GreenCloud, and MDCSim (Malhotra and Jain (2013)). In this research, the three phases of our model will be implemented and evaluated using a CloudSim environment. There are several reasons for this. Firstly, CloudSim is an open source simulator developed using the Java language so it is available for the public to use and to improve. In addition, it includes several submodules that simulate the main components and layers of cloud environments. This enables flexible customization of the simulation by adding or modifying modules according to the desired design. Specifically, the model in this research was implemented using CloudSim 3.0.3 which was the most recent version up until the middle of 2017. CloudSim is a general and extendable simulation model that facilitates the modeling and evaluation of cloud computing infrastructures and services (Calheiros et al. (2011)). It supports several functionalities, such as the queuing and processing of events, the creation of cloud system entities (services, hosts, data centers, brokers, and virtual machines), communication between components, and management of the simulation clock (Calheiros et al. (2011)).

CloudSim includes the following main classes:

- **Data center**: models the main hardware infrastructure of the cloud and is managed by the providers.

- **Broker**: represents a broker module, which manages the communication and negotiation between consumers and providers.

- **Host**: models a host inside the data center.
- **VM**: models a virtual machine that is running on the host and executes the tasks.
- **Cloudlet**: models the cloud applications and services in the SaaS layers (in this model it is denoted by the term task).
- **VmAllocation**: a policy that determines the method of allocating VMs to hosts and is specified in the data center characteristics.
- **VmScheduler**: a policy of allocating processing cores inside the host for VMs. It runs on each physical host in the data center to distribute CPU cores among VMs.
- **CloudletScheduler**: a policy of scheduling cloudlets to CPU inside the VM and is specific to each VM.

To implement the model, several modifications were made to the CloudSim simulator classes to customize them to the specified problem. Existing classes were therefore modified and new classes added for negotiation, monitoring, computing the load, and clustering hosts. Details of these modifications will be discussed in subsequent chapters.

The architecture of CloudSim consists of five layers, as shown in Figure 3.6. These layers represent the main structure of any cloud computing environment. In our model, the five layers are Cloud Data center, Manager, VM, Network, and Broker, as shown in Figure 3.7. In this research, the Network layer is required because the tasks will be scheduled among multiple data centers. The Cloud Data center layer provides APIs to start and terminate the instances of cloud components, which represent the Provider Module in our model. The Manager layer is responsible for management, resource allocation in the SLA negotiation phase and the task scheduling phase. This represents the Manager module in the model. In addition, it collects information from each data center and allocates resources. The VM layer manages the mapping of VMs to hosts within a data center, and represents the VM Manager in the model. In addition, it collects resource utilization information (e.g. CPU, memory, disk, etc.) on each host and on the virtual machines. The Broker layer provides the interface between the consumer and the Manager layer. In the proposed model, the Consumer Agent represents this layer.

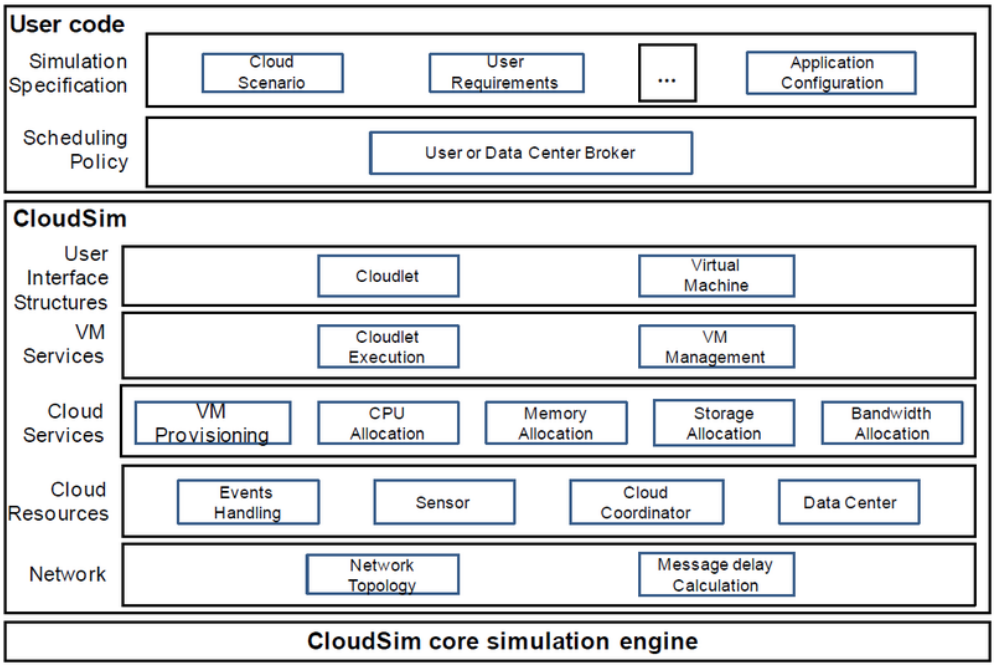

Figure 3.6: CloudSim Architecture (Calheiros et al. (2011)).

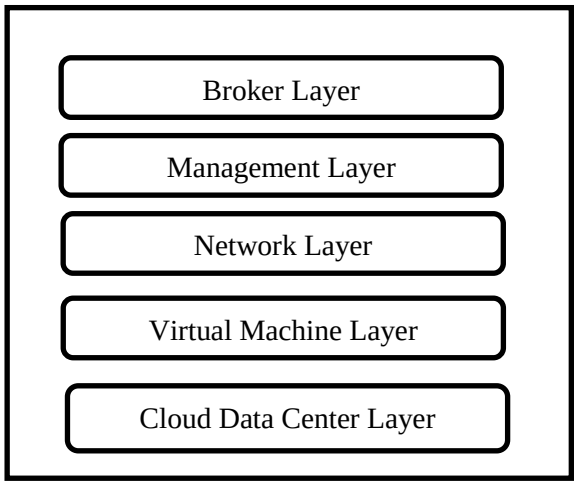

Figure 3.7: Proposed Model Layers

3.4.1 Configurations and Specifications of Resources

To evaluate the proposed resource allocation model, a series of experiments were performed to evaluate the objectives discussed previously in Chapter 1. Each phase of the proposed model has several such objectives; these will be evaluated after discussing the implementation information in detail in Chapters 4, 5, and 6. In this section, general information related to the simulation and specification details that will be used in all

phases will be described, including the characteristics of the data centers, hosts, VMs, and the tasks that will be used in the evaluation experiments.

Table 3.1 specifies the data centers that are used in our model which include the supposed values for cost of using the resources in the data center. To simulate the host machines, Amazon's Elastic Compute Cloud (EC2[1]) instances that differ according to CPU type were used. Four types of EC2 were selected: high-CPU, extra, small, and micro, with different characteristics for each host, as shown in Table 3.2. The data centers in the proposed model differ in terms of the costs of processing, memory, storage, transferring data, and power. The selected values are based on EC2 instances for general-purpose applications.

It should be noted that the network bandwidth values in the host specification used in the evaluation tests of our model in the three phases is multiplied by 100 (see Table 3.1) and for VM bandwidth (see Table 3.2) is multiplied by 10.

DC ID	Cost per Processing $/ seconds	Cost per Memory $/ MB	Cost per Storage $/ MB	Data Transfer cost $/ Mb	No. of Host	Cost Power $/ seconds
1	3.0	0.05	0.1	0.1	4	0.5
2	3.0	0.05	0.1	0.1	4	0.5
3	2.0	0.04	0.1	0.05	4	0.5
4	2.0	0.04	0.1	0.05	4	0.5
5	3.0	0.05	0.1	0.05	4	0.5
6	3.0	0.05	0.05	0.1	2	0.3
7	3.0	0.04	0.05	0.1	2	0.3
8	3.0	0.04	0.05	0.1	2	0.3
9	2.0	0.04	0.05	0.1	2	0.3
10	2.0	0.04	0.05	0.1	2	0.3

Table 3.1. Specification of Data centers.

[1] https://aws.amazon.com/ec2/instance-types/

Host ID	Host Type	RAM (GB)	Bandwidth (Mb/s)	CPU (GHz)	Storage Size (GB)	No. of cores
1	Micro	1	1000	Up to 3.3	100000	2
2	Small	2	10000	Up to 3.3	200000	2
3	Extra	4	20000	Up to 3.3	400000	4
4	high-CPU	8	40000	Up to 3.3	400000	4

Table 3.2. Specification of Host Types.

VMs were simulated with the specification that fits in the simulated hosts. Four types of VM that vary in CPU speed and memory size with the specification of EC2 instances were used as shown in Table 3.3, which shows that each VM has one core. In cloud computing, the creation of a VM is implemented in two ways. The first VM is creating based on task requirements then it is mapped to a host that can fit, while the other depends on creating a VM in the host after which the task is mapped to a suitable VM. In our model, we combine the two methods in such that VMs were created depending on the host specification and tasks then mapped to them. In each data center, there are specific type of VMs based on the specification of the hosts. This is because it manages the VMs in a more efficient way than the first method and reduces the number of created VMs, therefore utilizing the resources more efficiently. In contrast, the first method creates a number of VMs that is equal to the number of tasks, and thus leads to complexity when allocating VMs to hosts. The specific characteristics for each VM are shown in Table 3.3 below.

VM ID	VM Type	RAM (MB)	Bandwidth (Mb/s)	CPU (MIPS)	Storage Size (MB)	PE
1	Small	4048	1000	10000	10000	1
2	Medium	4048	2000	20000	20000	1
3	Large	8096	4000	40000	40000	1
4	xLarge	8096	8000	80000	80000	1

Table 3.3: Specification of VM Types.

To simulate the tasks, a dataset called the Large Hadron Collider Computing Grid (LCG) were used (Schwickerath et al. (2005)). This describes the work of 11 days of activity from multiple nodes. It includes 188.041 lines, and each line in the file contains information about the completed task along with information on the submission time, consumer ID, compute element name, and task runtime. This dataset was used because it includes task information relative to that needed in the proposed model. Tasks that are not included in our model are tightly coupled tasks and complex workflow applications. The tasks are simulated to be submitted to our system, which includes different number of VMs and hosts with random types based on the predefined specifications. The tasks are taken from the file without sorting and thus the results are varied because the tasks are different in length and specifications. In the model we consider the process of dealing with tasks are dynamic and online and the tasks are scheduled directly when they arrived and are not queued.

3.4.2 Evaluation Parameters

After running the simulation, specific parameters were measured based on the objectives of the proposed model. The main parameters used, as indicators of the model's goals (see Section 1.3) were average waiting time, average execution time, throughput, resource utilization, cost, profit, and power consumption. Details on each factor are presented as follows:

- **Average Waiting Time (AWT):** The average waiting time of a task is defined as the ratio of the sum of waiting times of all tasks to the total number of tasks. The average waiting time is measured by computing the difference between the time the task is submitted to the system and the time of starting the execution of all the tasks, as shown in Equation 3.1. The waiting time includes the time taken for negotiation, mapping tasks to VMs, data transfer time, and the time at which VMs are migrated.

$$\mathbf{AWT} = \sum_{t=0}^{m} (\boldsymbol{ST}(\boldsymbol{i}) - \boldsymbol{SubT}(\boldsymbol{i}))/\mathbf{m} \quad (3.1)$$

where:
AWT is the average waiting time of all tasks in seconds
m denotes the number of tasks running in the system per unit of time

ST (i) denotes the start time of execution of the task i in seconds
SubT (i) denotes the time for submission of the task i in seconds

- **Average Completed Time (ACT):** This is the total time taken by each task to finish execution. It is measured by computing the difference between the time of submission of each task and the time of ending execution of each task.

$$\mathbf{ACT} = \sum_{t=0}^{m}(\boldsymbol{Ext}(\boldsymbol{i}) - \boldsymbol{Subt}(\boldsymbol{i}))/\mathbf{m} \quad (3.2)$$

where:
ACT is the average completed time of all tasks in seconds
m denotes the number of tasks running in the system per unit of time
Ext (i) denotes the finish execution time of the task i in seconds
Subt (i) denotes the time of submission of the task i in seconds

- **Throughput (TH):** The throughput measures the overall performance of the system. Throughput indicates the number of tasks that our model can execute in a specific time.

$$\mathbf{TH} = (\mathbf{C}/\mathbf{T}) \times \mathbf{100} \quad (3.3)$$

where:
TH denotes the throughput of the system
C is the total number of completed tasks
T denotes the simulation time in seconds

- **Average VM Utilization (AVU)**: This represents the utilization of the VM in terms of the CPU, memory, storage and bandwidth used by all tasks to finish execution. The CPU utilization for each VM is defined as a percentage ratio of the CPU, memory, storage and bandwidth utilization divided by 4 (which is the number of factors including in computing VM utilization). The CPU utilization is the amount of CPU used for all tasks in a VM over the total CPU of the VM. Memory utilization, storage utilization, and bandwidth utilization are computed in the same way as CPU utilization, as shown in Equations 3.4, 3.5, 3.6 and 3.7.

$$\boldsymbol{CVM\ (i)} = \sum_{i=0}^{n}(\boldsymbol{UC}(\boldsymbol{i}) \,/\, \boldsymbol{AC}(\boldsymbol{i})) * \mathbf{100} \quad (3.4)$$

$$\boldsymbol{MVM\ (i)} = \sum_{i=0}^{n}(\boldsymbol{UM}(\boldsymbol{i}) \,/\, \boldsymbol{AM}(\boldsymbol{i})) * \mathbf{100} \quad (3.5)$$

$$SVM\ (i) = \sum_{i=0}^{n}(US(i) \,/\, AS(i)) * 100 \quad (3.6)$$

$$BVM\ (i) = \sum_{i=0}^{n}(UB(i) \,/\, AB(i)) * 100 \quad (3.7)$$

$$UVM\ (i) = (CVM\ (i) + MVM\ (i) + SVM\ (i) + BVM\ (i))/4 \quad (3.8)$$

where:
UVM (i) is the average utilization of VM i
CVM (i) is the CPU utilization of VM i
UC (i) is the used CPU for all tasks executed in VM i
AC (i) is the total CPU of VM i
MVM is the memory utilization of VM i
UM (i) is the used memory for all tasks executed in VM i
AM (i) is the total memory of VM i
SVM (i) is the storage utilization of VM i
US (i) is the used storage for all tasks executed in VM i
AS (i) is the total storage of VM i
BVM (i) is the bandwidth utilization of VM i
UB (i) is the used bandwidth for all tasks executed in VM i
AB (i) is the total bandwidth of VM i
n denotes the number of VMs

- **Average Resource Utilization (ARU):** Host utilization in our model represents the total utilization of all VMs running on the host. The average resource utilization is computed by summing the host utilization of all available hosts in the data center (Equations 3.9 and 3.10).

$$HU\ (j) = (\sum_{i=0}^{i=n} UVM(i))/n \quad (3.9)$$

where:
HU is the average host utilization for host j
UVM (i) denotes the utilization of all VMs in the host j
n denotes the number of VMs in host j

$$ARU = \sum_{j=0}^{j=m} HU(j)/m \quad (3.10)$$

where:
ARU denotes the resource utilization
HU is the average host utilization for host j as shown in Equation 3.9
m denotes the number of hosts in the data center

- **Execution Cost (EC)**: This factor denotes the cost consumers should pay to execute tasks in the providers' resources. It includes the cost of CPU processing, memory, using data storage, and transferring data (Equation 3.11). These costs are all defined in the data center specifications and differ based on the resources.

This is an important factor for consumers and one that most resource allocated algorithms aim to minimize.

$$EC = \sum_{i=0}^{i=n} \begin{matrix} ((T(i) \times costCPU) + (R(i) \times costRAM(i)) \\ + (S(i) \times costStorage(i)) + (D(i) \times costB(i))) \end{matrix} \quad (3.11)$$

where:
EC is the cost of execution of all tasks
n denotes the number of tasks executed in the data center
T (i) is the execution time in seconds of task i
costCPU is the cost of processing CPU in $ / seconds
R (i) is the size of RAM of task i in MB
costRAM is the cost of memory used in $ / MB
S (i) is the size of storage of task I in MB
costStorage is the cost of storing data in $ / MB
D (i) is the size of the task I file in Kb
costB (i) is the cost of transferring data in $ / Mb

- **Total Profit (TP):** The profit model is based on a pay-as-you-go policy that is applied in many cloud systems to address the highly variable demand for cloud resources and to calculate the cost of executing tasks in the cloud data center (Lee et al. (2012)). Thus, profit represents the total income the provider can gain from executing tasks in their resources. It is calculated depending on the total execution cost for all tasks, penalty costs, and power consumption costs, as shown in Equation 3.12. The penalty cost is computed based on the provider policy and is defined in the SLA; it represents the costs the provider can afford to pay if the SLA is not satisfied. In the proposed model, the penalty is paid for execution delay and is computed according to the delay constraints, as shown in Equation 3.13.

$$TP = (EC - Pcost - \sum_{i=0}^{i=t} Pen(i)) \quad (3.12)$$

$$Pen(i) = \begin{cases} (reqt(i) - exet(i)) \times P & if\ reqt(i) > exet(i) \\ 0 & otherwise \end{cases} \quad (3.13)$$

where:
TP is the total profit the provider gains from execution of all tasks
EC is the cost of execution of tasks
Pen (i) is the penalty the provider can afford to pay for delay in task i
Pcost is the cost of using power for executing tasks, as shown in Equation 3.14
reqt (i) is the time task i is required for execution
exet (i) is the time task i is taken for execution
P is the penalty cost defined for the delay

T is the number of tasks

Power Consumption (PCost): This is based on the power cost in kWh, which differs according to the specifications of the data center. It is computed by multiplying the power cost by the total execution time for all tasks, as shown in Equation 3.14.

$$\boldsymbol{PCost = PC \times Powercost} \tag{3.14}$$

where:
PCost is the total power cost of execution of all tasks
PC is the total execution time for all tasks
Powercost is the cost of power defined in $/seconds

- **Task Completed Rate (TCR)**: This is the number of completed tasks over the total number of submitted tasks.

$$\mathbf{TCR = T_c \,/\, T_S} \tag{3.15}$$

where:
TCR is the rate of completed tasks, and ranges from [0-1]
T_c is the number of completed tasks
T_s is the number of submitted tasks

- **The SLA Violation (SLAV):** In this research, an SLA violation can occur in relation to deadline and migration time. A SLA violation of deadline constraints occurs when the task deadline is missed and is computed according to the rate of completed tasks computed as shown in Equation 3.16.

$$\mathbf{SLAD = (1 - TCR) \times 100} \tag{3.16}$$

where:
SLAD denotes the SLA violation based on deadline
TCR is the task completed rate computed by Equation 3.15

A SLA violation for migration time occurs when the consumer does not receive their requested resources. In technical terms, SLA violations occur when VMs cannot acquire the amount of MIPS that are requested. In this case, the SLA violation occurs when the requested CPU is greater than the available capacity of CPU. It is computed as the sum of unallocated MIPS to the sum of the requested MIPS as shown in Equation 3.17.

$$\mathbf{SLAM} = \sum_{i=0}^{i=m} (\boldsymbol{RMIPS(i) - AMIPS(i))/AMIPS(i)} \tag{3.17}$$

where:
SLAM denotes the SLA violation based on migration time
RMIPS (i) denotes the MIPS requested by the VM i for running the task
AMIPS (i) denotes the actual MIPS that were allocated to the VM i.

The overall SLA violations are computed as shown in Equation 3.18.

$$\mathbf{SLAV = SLAD + SLAM} \tag{3.18}$$

where:
SLAD denotes the SLA violation based on deadline
SLAM denotes the SLA violation based on migration time
SLAV denotes the overall SLA violation

- **The Imbalance Factor (IF):** This refers to the balance of the load among VMs. It is measured based on the time for executing tasks in VMs. A small value of IF indicates good load balancing. Equation 3.19 shows how to compute IF.

$$\mathbf{IF = (T_{max} - T_{min}) / T_{av}} \tag{3.19}$$

where:
IF is an imbalance factor
T_{max} is the maximum time for execution of tasks
T_{min} is the minimum time for execution of tasks
T_{av} is the average time for execution of tasks

- **Average Fitness Values (AFV)**: The average fitness value for each solution is calculated by applying the PSO algorithm and evaluating selected objectives, as shown in Equation 4.20.

$$\mathbf{AFV = \sum_{i=1}^{m} F(i)} \tag{3.20}$$

where:
AFV is the average fitness value for all tasks
i is the index of the task
m is the total number of tasks in the simulation
F (i) is the value of fitness function for all solutions

- **Processing Time (PT)**: the processing time is the time taken to run a specific algorithm. In the SLA negotiation, it is called negotiation time and in VM migration, it is called migration time.

3.5 Summary

This chapter described the general architecture of the resource allocation model presented in Section 1.3. The proposed model comprises three phases: SLA negotiation, task scheduling, and VM allocation. In addition, the chapter discussed the specifications of each phase of the model. The chapter also presents general issues related to implementation of the model using the CloudSim simulator and the main classes that are used or modified. Finally, it outlines the main parameters used in the evaluation of each phase of the model.

The model phases will be discussed and presented in subsequent chapters where Chapter 4 covers the SLA negotiation, Chapter 5 describes the task scheduling algorithms and Chapter 6 presents the VM allocation algorithm.

Chapter 4

SLA Negotiation Based on Parallel Particle Swarm Optimization

This chapter discusses details of the SLA Negotiation phase based on Parallel PSO, which is the first phase of the proposed model (see description in Chapter 3). It starts by providing an overview of the SLA Negotiation algorithm. In Section 4.2, the sequential PSO Negotiation algorithm is then presented. Section 4.3 discusses the design of the synchronous and asynchronous Parallel PSO negotiation algorithms. Implementation details for the proposed algorithms are covered in Section 4.4. In Section 4.5, evaluation methods and discussion of the results of applying the algorithms are presented. The last section summarizes the main contributions of these algorithms according to the objectives discussed in Section 1.3.

4.1 Overview

The proposed model includes many distributed data centers consisting of several virtual and physical resources. Resource allocation in cloud computing needs to be able to deal with the features of cloud computing which are described as large-scale, scalable, and dynamic (see Section 1.1.1). Thus, dynamic real-time scheduling algorithms are required to allocate cloud resources. In a dynamic real-time scheduling algorithm, the waiting time should be reduced; therefore reducing waiting time is one of the key goals of our model. This will be achieved by reducing the time for mapping between task and cloud resources. Searching inside each data center for suitable resources for each task is a complex process, and this complexity is increased as the number of resources and tasks grows. The proposed model utilizes techniques from meta-heuristic optimization

and the structure of the model (the distributed data center) to reduce mapping time. It improves the negotiation strategy between consumers and providers by applying the Parallel PSO algorithm. Thus, instead of searching all the providers' resources, the search space is divided into smaller spaces, which are then searched concurrently. Consequently, the time spent on mapping is reduced which leads to improved throughput and increased profit. Specifically, Parallel PSO in the proposed model divides the swarm into multi-swarms, each of which is run on one data center, and the best solutions are then returned to the Manager Module. Using a Parallel PSO algorithm rather than a sequential PSO is more effective, because better solutions are provided by searching small spaces than by searching large spaces. In addition, the structure of the distributed data center is suitable for applying parallel PSO because each data center can evaluate one set of solutions rather than evaluate all in one node.

Cloud consumers aim to reduce the time of execution and providers aim to maximize their profit by reducing the number of tasks that exceed deadline. The negotiation is conducted to satisfy the goals of both parties. The negotiation steps can be reduced to save time if both parties agree. In addition, negotiation can be improved if the consumers and providers allow a third party to select resources based on their objectives and perform mapping processes on their behalf to reduce traffic and communication messages. In this way, the overall waiting time can be reduced, including mapping time. Thus, the throughput of the system will be increased.

Specifically, in this work, the cloud consumers and the providers must define the requirements and objectives, and the Manager Module then starts the negotiation process based on these requirements. The SLA Negotiator Agent in the Manager Module selects the best offer for each task from the available data centers by applying the Parallel PSO algorithm. The objective function of this algorithm is based on three factors: execution cost, network delay, and current load in the data center. Specifically, the Broker Agent send tasks to the SLA Negotiator Agent along with QoS requirements and constraints such as deadline, cost budget, CPU speed, and memory size. The SLA Negotiator Agent then begins the negotiation process by communicating with all data centers through the DC Manager Agent, collecting the results and then selecting the most appropriate solution. If the request is accepted, a formal SLA agreement is created and signed which guarantees QoS between both parties, including the consumer's

requirements and the penalty cost if the agreement is violated. The Negotiator Agent uses the Parallel PSO algorithm to map tasks to the data center to satisfy predefined goals such as improving performance by reducing waiting time, avoiding SLA penalties, and improving throughput. Specifically, the SLA Negotiator selects the data center from those in the cloud infrastructure according to a set of parameters that include CPU speed, RAM size, storage size, cost budget, deadline and penalty.

The process for the proposed SLA negotiation algorithm is shown in Figure 4.1. It involves communication between four components: Broker Agent, Negotiator Agent, Task Dispatcher Agent and DC Manager Agent. The Broker Agent accepts the tasks, including the requirements from consumers, and then prepares the proposal for the SLA Negotiator Agent. The SLA Negotiator Agent acts as a mediator between the consumers and the DC Manager Agent, and is located in each data center to manage the resources. These components then communicate to manage the SLA Negotiation and create SLA agreements. The SLA Negotiation Processes can be summarized as follows:

1. The Broker Agent receives proposals from the consumers that describe all the tasks along with their requirements and constraints including the deadlines and cost budget. The Broker Agent then sends these proposals to the SLA Negotiator Agent.

2. In the proposed model, the DC Manager Agent collects information about data center resources from the Host Monitor Agent. Resource information includes data center status, which describes the current load of the data center, the types of VMs, the specifications of each VM in terms of software, hardware, and the cost, and dynamic information such as availability and current load.

3. The SLA Negotiator Agent accepts proposals from the Broker Agent and then starts the Parallel PSO algorithm by sending the proposals for each data center in parallel.

4. In each data center, based on the type of VMs, a set of offers for the proposals is prepared. The PSO algorithm then runs to find a suitable mapping between each proposal and the offers prepared; the result, which includes the data center

ID, is sent back to the SLA Negotiator Agent. The steps undertaken by the PSO algorithm are presented in Algorithm 4.4.

5. The SLA Negotiator Agent accepts the results and selects the best. These tasks are then passed to the selected data centers to execute and a confirmation message is then sent with the agreed SLA to the Broker Agent. However, when the negotiation fails because of deadline or cost constraints or there is no data center, the failed message is sent to the Consumer Agent.

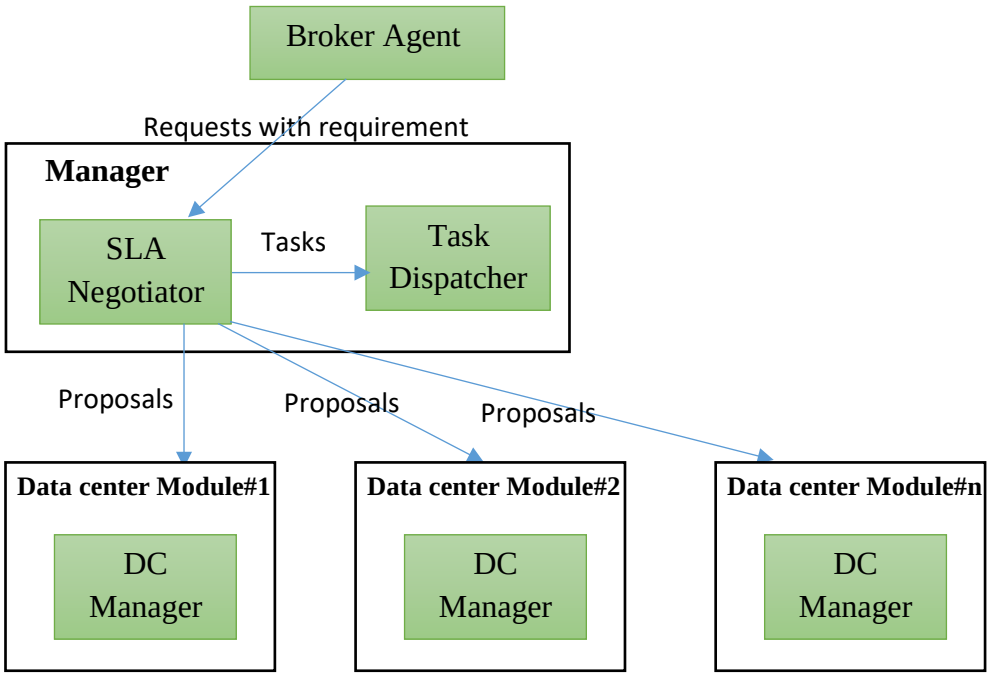

Figure 4.1: SLA Negotiation Processes.

4.2 SLA Negotiation Algorithm Formulation

The negotiation inside the data centers is carried out using a Parallel PSO algorithm that is based on three factors: cost, network delay, and data center load. The consumers send tasks, including the requirements and the constraints of deadlines and cost budgets, to the Consumer Agent who then uses them to form the proposal. The proposal includes these attributes: *{Task length, data file size, CPU Processing, Memory Size, Storage size, Max Cost, Deadline}.*

The data centers provide a set of offers based on the type of VMs that can initiate and run on the hosts. The offer includes both static information and dynamic information based on the existing status of the system. The main attributes of the offer are: *{Data center ID, VM Cost, VM Type, CPU Processing Speed, RAM Size, Storage Size, network Bandwidth, Data center load, Penalty}.*

The input and output data are considered first. In this instance, the SLA negotiation algorithm receives a set of tasks in the form of proposals and the ID of the data centers as input data. The output results of this algorithm are the mapping between tasks and data centers, which then enables tasks to dispatch into the selected data centers.

Next, the problem constraints are addressed. The negotiation process is conducted under specific constraints to reach an agreement. These constraints include:

1) Tasks should be assigned to one data center at a time so that, in the mapping matrix, each row has only one element with a value of 1, as shown in Equation 4.1.

$$\sum_{i=1}^{n} \boldsymbol{map}\,(\boldsymbol{i},\boldsymbol{j}) \;\leq \mathbf{1}\;\forall\; \boldsymbol{j} = \mathbf{1},\mathbf{2},\ldots,\boldsymbol{m} \tag{4.1}$$

where:
i is the task index
j is the data center index
m is the total number of tasks
n is the total number of data centers
m (i, j) is a binary value {0, 1}

2) Each task should be scheduled within a deadline, which indicates the maximum response time of cloud providers to tasks from consumers. Cloud providers must respond to consumers' requests within a reasonable time, otherwise an SLA violation occurs and cloud providers need to pay the penalty for violation to the consumer. The deadline can be computed in several ways; in this research, Equation 4.2 is used because execution time is based on processing time, which relates to the CPU speed of the VM in such that the minimum and maximum time for executing task are considered. The deadline is computed as an absolute value because it is should be a positive value.

$$\boldsymbol{Del}\,(\boldsymbol{i},\boldsymbol{j}) = |(\boldsymbol{L}(\boldsymbol{i},\boldsymbol{j}) \;+\; \mathbf{0.5}\;(\boldsymbol{L}(\boldsymbol{i},\boldsymbol{j}) - \boldsymbol{High}(\boldsymbol{i},\boldsymbol{j}))) * \mathbf{100}\,| \tag{4.2}$$

where:
i is the index of the task
j is the index of the VM type in the data center
Del (i, j) is the delay for task i using VM type j
L (i, j) is the minimum time for executing task i using VM type j
High (i, j) is the maximum time for executing task i in VM type j.

The execution time of a task with a network delay should not exceed the deadline, as shown in Equation 4.3.

$$\boldsymbol{Exe}\,(\boldsymbol{i},\boldsymbol{j}) + \boldsymbol{NDelay}\,(\boldsymbol{i},\boldsymbol{j}) < \boldsymbol{Del}\,(\boldsymbol{i},\boldsymbol{j}) \quad (4.3)$$

where:
Exe (i,j) is the time for executing task I in VM j.
Del (I,j) is the maximum delay time allowed to execute task i in VM j, which is computed as shown in Equation (4.2).
NDelay (i, j) is the network delay of task i in VM j as shown in Equation 4.6

3) Cloud providers require prices for using the VM. The consumer must pay the price requested by the cloud providers based on the resources used. The amount a consumer must pay per hour for using a VM from a resource provider should be determined and the consumer specifies the maximum cost allowed as follows:

$$\mathbf{TEC}\,(\mathbf{i},\mathbf{j}) < \mathbf{Costbudget}\,(\mathbf{i}) \quad (4.4)$$

where:
TEC (i, j) is the execution cost of task i in VM j that is computed by using Equation 4.5
Costbudget (i) is the maximum permitted cost specified by the consumer.

4) The minimum requirement in terms of the processing speed of the VM in MIPS, minimum memory size, and storage disk size should be satisfied in the selected data center such that $CPU_{req} \leq CPU_{av}$,$Memory_{req} \leq Memory_{av}$, $Storage_{req} \leq Storage\ _{av}$,$Bw_{req} \leq Bw_{av}$.

where:

$\mathrm{CPU_{req}}$ is the requested CPU in term of MIPS
$\mathrm{CPU_{av}}$ is the available CPU in term of MIPS
$\mathrm{Memory_{req}}$ is the requested Memory in term of MB
$\mathrm{Memory_{av}}$ is the available Memory in term of MB
$\mathrm{Stroage_{req}}$ is the requested Storage in term of MB
$\mathrm{Storage_{av}}$ is the available Storage in term of MB
$\mathrm{Bw_{req}}$ is the requested Bandwidth in term of Mb/s
$\mathrm{Bw_{av}}$ is the available Bandwidth in term of Mb/s

The objective function is the next issue to be considered. The aim in SLA negotiation is to find the best mapping to a data center within existing constraints. The best mapping is determined by using the fitness function, which is based on minimizing three factors: cost of execution, network delay, and data center load. Details on these factors are as follows:

- **The Task Execution Cost** (TEC): This is the total cost needed to execute the task in the VM, which includes the cost of processing per second, the cost of storage, the cost of memory and the cost of bandwidth, as shown in Equation 4.5.

$$\boldsymbol{TEC(i,j)} = ((\boldsymbol{Exe(i,j)} \times \boldsymbol{costCPU(j)} + (\boldsymbol{R(i)} \times \boldsymbol{costRAM(j)}) + (\boldsymbol{S(i)} \times \boldsymbol{CostStorage(j)}) + (\boldsymbol{D(i)} \times \boldsymbol{costB(j)})) \quad (4.5)$$

where:
TEC (i, j) is the cost of execution task i in VM j in $
Exe (i, j) is the execution time in seconds of task i in VM j in seconds
costCPU is the cost of processing CPU in $ / seconds
R (i) is the size of RAM in task i in MB
costRAM is the cost of memory used in $ / MB
S (i) is the size of storage of task i in MB
costStorage is the cost of storing data in $ / MB
D (i) is the size of the task i file in Kb
costB (i) is the cost of transferring data in $ / Mb

- **The Network Delay**: This is the time required to transfer data between two nodes and depends on the topology of the network. This is computed based on the data length of the task and the VM bandwidth, as shown in Equation 4.6.

$$\boldsymbol{NDelay}\ (\boldsymbol{i,j}) = \boldsymbol{TL(i)/B(j)} \quad (4.6)$$

where:
NDelay (i, j) is the delay when moving task i to VM j
TL (i) is the length of task i in MI
B (i) is the bandwidth of VM j in Mbs

- **The Data Center Load**: This encompasses by the utilization of all hosts in the data center, as shown in Equation 4.7. The host utilization is computed as the summation of used MIPS in all VMs in the host divided by the total MIPS in

the host, as shown in Equation 4.8. In this model, only the utilization of CPU processing is considered because the proposed model focuses on the tasks that need high processing speed rather than large storage capabilities. The processing speed is measured in MIPS. The data center load is a percentage value in the range (0-1)

$$\boldsymbol{DCload} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{m}} \boldsymbol{Hload}\,(\boldsymbol{j}) \tag{4.7}$$

$$\boldsymbol{Hload}\,(\boldsymbol{j}) = (\sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} \boldsymbol{usedMIPS}(\boldsymbol{i}) / \boldsymbol{TotalMIPS}(\boldsymbol{j}))/\mathbf{100} \tag{4.8}$$

where:
Dload is the load of the data center
Hload (j) is the utilization of the host j
usedMIPS (i) is the MIPS used by all tasks executed in the VM i
TotalMIPS (i) is the total MIPS in the VM i
m is the number of hosts in the data center
n is the number of VMs in host j

The objective function aims to determine the lowest cost for execution with the least amount of network delay and the smallest data center load. The fitness function is formulated using a weighted sum approach, as shown in Equation 4.9. The cost of execution is given a more substantial weight because it has a high preference amongst consumers.

$$\boldsymbol{Min\ F(xi)} = \mathbf{0.4} * \boldsymbol{TEC(i)} + \mathbf{0.3} * \boldsymbol{NDelay} + \mathbf{0.3} * \boldsymbol{DCload} \tag{4.9}$$

where:
TEC (i) is the cost of execution task i in the resource in seconds (Equation 4.5).
NDelay is the delay of the network in seconds. (Equation 4.6).
DCload is the load of the data center (Equation 4.7).

In the next section, the SLA negotiation based on sequential PSO will be described in detail and the results compared with the Parallel PSO algorithm.

4.3 Sequential PSO Negotiation Algorithm

Initially, the SLA negotiation is developed using the PSO technique to compare it with the Parallel PSO proposed in the current model. PSO starts with a specific number of particles; each particle denotes specific solutions for the tasks during the negotiation process (Section 2.2.1). In this case, the search space is large because it includes all

VMs in all data centers. Specifically, Algorithm 4.1 summarizes the main steps of SLA negotiation based on the PSO algorithm. The algorithm maps arrival task to the data center in Line 2 by calling upon the Procedure Negotiation_PSO (Algorithm 4.2) in sequential PSO or Negotiation _PPSO to run the Parallel PSO algorithm specified in Algorithm 4.3. In this model, we deal with real time scheduling which includes dealing with tasks directly and do not queued them to reduce the waiting time.

Algorithm 4.1: SLA Negotiation Algorithm.

```
Inputs: T, D, type                                   // list of tasks and data centers
Outputs: map (t, d)                                  // map matrix of tasks and data center ID
Procedure SLANegotiation (T, D, type)                // negotiation type parallel or sequential
1.  for arrival tasks in T list
2.      If type ==sequential then
3.              ID=Negotiation_PSO (t, D)            // map tasks to data center by algorithm 4.2
4.      Else if type == parallel
5.              ID=Negotiation_PPSO (t, D)           // map tasks to data center by algorithm 4.3
6.      End if
7.      If (ID!= null)                               // there is suitable resource for task
8.              Update map (t, ID)
9.              Dispatch task to selected data center
10.     Else
11.             Update t status to failed            // there is no suitable resource for tasks, task failed
12.     End
13. End for
14. Update T list
15. Repeat for step 2 until Task list T is empty
End procedure
```

The details of the Negotiation_PSO algorithm in Line 3 are listed in Algorithm 4.2. The algorithm begins by initializing the main values of particles, number of iterations, position, and velocities. The initialization step includes encoding the particle values to deal with discrete PSO. The SPV method (as described in Section 2.2.1) is used in this model because it is the strategy most commonly applied in existing research. In this method, the particles are represented by a 1 × n vector; where n is the number of tasks and the value assigned to each position is the index for all data centers. The position is represented by an m × n matrix and an encoding strategy must be used to represent the solutions, where m is the index of the data center and n is the number of tasks. The values for each element in the position matrix can be either 0 or 1 with the constraint that only one element can take the value of 1 in each column. This is because each column represents a task allocation and each row represents an allocated task to the

specific resource. This ensures that during allocation, each task is assigned to only one data center. The velocity is represented by an m × n like position but it also includes integer values to control the movement of the particles, as shown in Equation 2.4. It revalues the changes in each iteration according to the new best values.

After initialization, the fitness function for all elements in the particle is computed based on Equation 4.9 and the best value is updated. The PSO algorithm searches for solutions with the minimum fitness value. The velocity and position are then updated based on the new best value, as shown in Equations 2.4 and 2.5 (see Chapter 2). These steps are repeated until the maximum number of iterations is reached. Finally, the best solution is the last best value of the particle, which includes the best mapping of tasks and the data center IDs. Figure 4.2 depicts a flowchart illustrating SLA negotiation based on a sequential PSO algorithm.

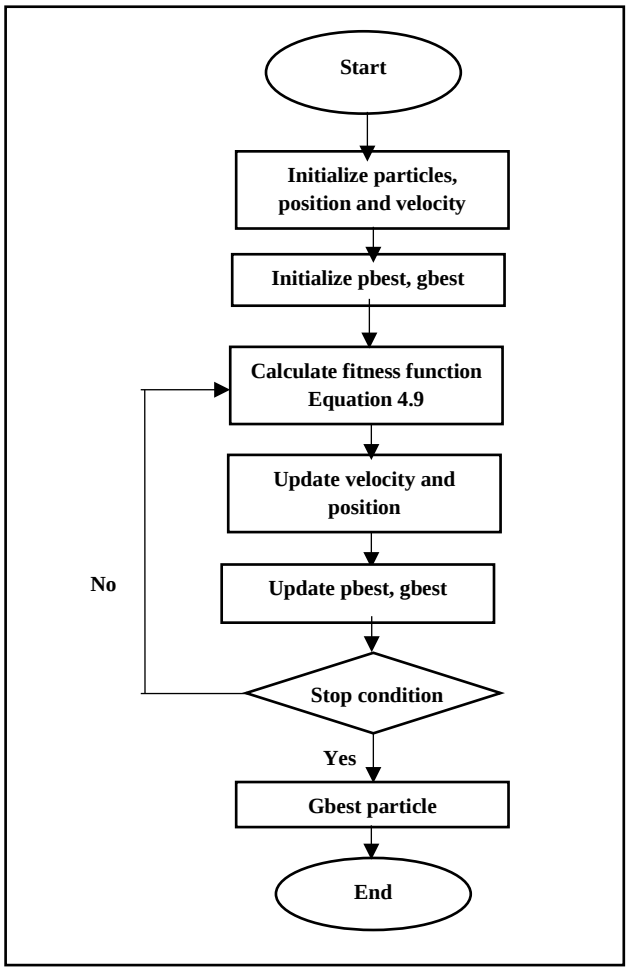

Figure 4.2: Flowchart for the SPSO Algorithm.

Algorithm 4.2: Sequential PSO Negotiation Algorithm.

```
Procedure Negotiation_PSO (t, D)
Inputs: t, D                                          // task t and D data centers
Outputs: d data center                                // the output is the selected data center ID
1.  Initialize (V, P, pbest, gbest)                   // initialize velocity, position, pbest, gbest
2.  while (stop criteria do not satisfied) do         // iterate while not stop
3.    For each p in t                                 // iterate through all values in particle t
4.      b=Evaluation (p)                              // objective function using Equation 4.9
5.      If (b < pbest)                                // update pbest value
6.          pbest=b
7.      End if
8.    End For
9.    Update (V, P)                                   // using Equations 2.4 and 2.5
10.   If (b < gbest)
11.         gbest=b
12.   End if
13. End while
14. Return gbest                                      // include data center ID
End procedure
```

4.4 Parallel PSO Negotiation Algorithms

The Parallel PSO algorithm needs to be improved to solve the SLA negotiation problem. As described in Section 2.2.3, the Parallel PSO is a set of independent swarms that communicate through a specific topology. The communication topology between multi-swarms has a significant effect on the performance of the Parallel PSO algorithm. It defines the method for transforming the parameters among swarms. The most common Parallel PSO topology is a master-slave topology model (Belal and El-Ghazawi, 2004). In this model, a single module (master) controls the optimization algorithm, and utilizes an external (slave) module to compute solutions. This model treats each slave as a swarm then selects the best solution from those available. Most research results indicate that the best architecture for applying Parallel PSO is a global best, which is based on the master-slave model (Belal and El-Ghazawi, 2004). Therefore, our model is also based on a master-slave model (as described in Section 2.2.3).

Several PPSOs have been proposed and they can be implemented in two main ways. The first method divides the search space into multiple swarms and each swarm executes the same algorithm on different nodes (Schutte et al. 2004). The second method is based on designing an algorithm using many nodes (equal to the number of particles) to compute and evaluate the fitness function separately and simultaneously

(Koh et al. (2006); Venter and Sobieszczanski-Sobieski (2006)). The second method of applying PPSO is suitable for complex problems whose fitness evaluations take a large time to compute in order as they help reduce computation time (Gonsalves and Egashira (2013)). In this research, the first method will be used because there are many distributed data centers with different characteristics and the aim is to speed up the process of negotiation to decide the best data center to run the task.

Specifically, the implementation of the Parallel PSO algorithm starts by considering each data center as a separate swarm and runs PSO. The master receives the best solutions, compares them and selects the minimum value, which is then deemed the best. The PPSO divides the search space into multiple swarms to focus on optimizing the smaller space of the problem. In our model, the search space is divided because there are resources in each data center that it needs to discover. The general steps of Parallel PSO are summarized in Algorithm 4.3. It starts by initializing the best value (Line 1). Then it sends tasks t to all data centers in parallel (Lines 3-7). The best results are selected as shown in Line 8. These steps are repeated while there is a task in the task list.

Algorithm 4.3: Parallel PSO Negotiation Algorithm.

Procedure Negotiation_PPSO (t, D)
Inputs: t, D *// task t and data centers*
Outputs: d data center *// the output is the selected data center ID*
1. Initialize best *// initialize best*
2. **While** (there are tasks in t) **do** *// iterate while there are tasks*
3. **Parallel**
4. **For** i=0: N *// where N is the numbers of data centers*
5. Result[i] = NPSO (i, t); *// tasks t send to all DCs to run NPSO* (Algorithm 4.4)
6. **End for**
7. **end Parallel**
8. Select best result (result) *// sort the result to select the minimum*
9. **end while**
10. **return** best *// return the best solutions for tasks*
End procedure

In more detail, Algorithm 4.4 runs in each data center. It starts with initializing values as shown in Line 1. After the initialization step, the objective function is calculated using the fitness function, as shown in Equation 4.9 (Lines 3-10). The objective function value is calculated for each particle, and the particles' pbest and gbest values are then evaluated (Lines 12-14). Finally, when the algorithm is finished, the global best value is sent to the Negotiator Agent in the Manager Module (Line 14).

Algorithm 4.4: PSO Negotiation Algorithm in data center**.**

```
Procedure NPSO (task t, data center i)
1.    Initialize (V, P, pbest, gbest)          // initialize swarm, velocity, position, pbest, gbest
2.    while (stop criteria do not satisfied) do          // iterate while not stop
3.    for each p in t                          //  // iterates all tasks in particle t
4.       for each v in data center i           // iterates all VM in data center i
5.          b=Evaluation (p (v))                        // objective function using Equation 4.9
6.          If (b < pbest)                              // update pbest value
7.               pbest=b
8.          End if
9.       End For
10.   End for
11.   Update (V, P)                                     // using Equations 2.4 and 2.5
12.      If (b < gbest)
13.              gbest=b
14.      End if
15.   End while
16.   Return gbest                                      // best value resource in data center i
End procedure
```

Based on the communication methods between swarms and the master node, there are two parallel adaptations of PSO: synchronous PSO (Schutte et al. (2004)) and asynchronous PSO (Koh et al. (2006); Venter and Sobieszczanski-Sobieski (2006)). In synchronous Parallel PSO (SPPSO), the master node waits for all swarms to finish searching before it decides the best solution whereas in synchronous Parallel Particle Swarm Optimization (ASPPSO) algorithm, the master node can select the first best solution and does not wait for the others to finish. Both approaches will now be discussed.

In SPPSO, the optimization algorithm waits at the end of every iteration until solutions for all the particles have been returned before updating particle velocities and positions (Zhou and Tan (2009)). The synchronization is required to ensure that all swarms have completed the evaluation of the fitness function and that results are returned before the velocity and position calculations can be executed (Belal and El-Ghazawi, 2004). In the SPPSO implementation, all swarms are running in separate data centers in parallel and the master node therefore waits for all swarms to be completed before selecting the best solution, as shown in Algorithm 4.3.

The problem for the SPPSO algorithm is that all the swarms need to wait for others to finish before deciding which is the best. To improve the PSO algorithm, an asynchronous parallel variant has been developed to speed up the convergence of the PSO to an approximate optimal solution (Gonsalves and Egashira (2013)). In the

ASPPSO, the algorithm does not wait for all solutions to be returned before selecting the best solution (Line 5) (Venter and Sobieszczanski-Sobieski (2006)). The difference between synchronous and asynchronous optimization lies in Line 5 of the Algorithm 4.3. In the synchronous approach, the master node waits for all swarms to compute the results then the best solution is chosen. However, in the asynchronous model, the first result from any swarm is used to continue the algorithm. In our model, Algorithm 4.3 is run in the SLA Negotiator Agent while Algorithm 4.4 is executed in each DC Manager Agent. The results of both are then sent to the SLA Negotiator Agent, which selects the solution.

4.5 SLA Monitoring Algorithm

The SLA Monitor Agent monitors the SLA for each task to find any violation. In each data center, the SLA Monitor collects information from agreed SLAs and checks whether there is any SLA violation before sending an alert. The SLA Monitor detects whether the SLA violation comes from missing the task deadline. Another type of violation is detected using the Host Monitor Agent. Algorithm 4.5 presents the main steps of the SLA Monitor Agent. It starts by initializing the SLAV by zero (Line 1). It checks the deadline and the execution time. If the execution time exceeds the deadline, then the SLA violation is increased by one (Lines 2-5). Finally, the number of SLA violations is returned (Line 6).

Algorithm 4.5: SLA Monitoring Algorithm.

Input: SLA parameters (deadline, execution time)

Output: number of SLAV
Procedure VMMonitoring (deadline, execution time)
1. Initialize the SLAV with 0
2. **For** each task in Task list
3. If execution time > deadline then
4. SLAV = SLAV +1
5. **End for**
6. **Return** SLAV

End Procedure

4.6 Parallel PSO Implementation

The model was implemented using CloudSim 3.0.3 as described in Section 3.4. This simulation model involves many data centers and many consumers. Virtual machines are created according to the specifications of the data center, and many tasks are simulated by using data from the dataset file. To implement the model, several modifications are made to the CloudSim simulator classes to customize them to evaluate the algorithm.

To implement the algorithm, the same specification of tasks, VMs, Hosts, and data centers were used that were described previously in Chapter 3. To run the SLA negotiation algorithm as a parallel algorithm, a conceptual model was used because it was not affected by the underlying details of the implementation. To simulate parallel computing, the focus was on parallel implementation using multi-threading. The rationale of a container (e.g., using agent frameworks like Jade) (Taranti et al. (2011)) was not adopted because this was a simulator rather than a real system. In addition, Jade is restricted to iterative computation and can be adaptive to recursive computation but not with the same performance (Gautier et al. (2007)). The approach is based on multi-threading programming in Java. Most of the previously developed algorithms in Parallel PSO negotiations have focused on the implementation of the Parallel PSO algorithm and the type of hardware that is used. In this research, the focus was on evaluation of the Parallel PSO algorithm itself and its role in improving resource allocation. In the proposed model, ten data centers are assumed so the swarm is split into ten multi-swarms containing the same number of data centers. These are then evaluated in parallel.

The CloudSim classes are then extended by adding some classes and modifying the available classes to adapt to the goals. For example, the Data center Broker class that represents the manager of the data center was parallelized to manage many data centers and a dispatcher function was added to submit tasks to the selected data center. In addition, a negotiation class was added to represent the negotiation process and determine the data center ID that can execute the task. Moreover, several classes were added to run and execute the PSO algorithm.

4.7 Experimental Evaluation

The methodology of the experiments and the specifications of the simulation environment used to test and evaluate the proposed algorithms will be described in Section 4.7.1. The results of the experiments will be presented and discussed in Section 4.7.2.

4.7.1 Experimental Methodology

The important issue in this chapter is evaluation of the effectiveness of the PPSO only. Therefore, it is assumed that in this case that after the selection of SLA negotiation, the task scheduling inside the data center is based on FCFS. The VM is then scheduled to available hosts based on FCFS. This is because it is important to evaluate each phase alone without any influence from other algorithms.

The performance of the Parallel PSO Negotiation algorithms was evaluated and compared with the results of the sequential PSO. The performance and the efficiency were then measured in terms of the negotiation time, average fitness values, and speedup to compare both the synchronous and asynchronous algorithms. The proposed model was then evaluated in terms of the resources allocated to determine whether the model satisfies the objectives discussed previously in Chapter 1. The details of these experiments are as follows:

- **Methodology 1:** This includes several experiments to test the effectiveness of the Parallel PSO by changing the number of iterations and comparing the results with the sequential PSO. In addition, it determines the best parameters within which to run Parallel PSO and satisfy the objectives by changing the number of iterations.
- **Methodology 2:** This was implemented for each of the Parallel PSO algorithms to measure the performance factors of the model in terms of negotiation time, waiting time, completed time and throughput. The results were then compared with the sequential PSO algorithm to determine the effectiveness of the proposed algorithms.

The specification of the data center, host, and VMs was as the same as described in Chapter 3. The specific numbers used in the Parallel PSO Negotiation algorithm are listed in Table 4.1. To simulate the task, the dataset LCG was used, as described in Chapter 3. Each experiment was run ten times and the average score across all experiments was then calculated with 95% confidence intervals.

# Data centers	10
# Hosts	50 per Dara center
# VMs	50 per Data center
# Tasks	50-500

Table 4.1: Setting of the Parameters for Experiments.

The values of the parameters used in the PSO are listed in Table 4.2, and include the number of particles and the number of iterations. Other parameters also control the PSO such as c1, c2, r1, r2, and inertia w values. These values are chosen to improve the quality of the solutions and the specific reason for selecting each parameter is described in Table 4.2.

PSO Parameters	**Values**	**Reason**
# swarms	10	Depends on the number of data centers
# particles	10	Depends on the number of data centers
# iterations	5-50	Based on the results of the tests of fitness function, there is no further change after 50 iterations in PSO
W	[0-1]	Specified based on Equation 2.8
c1 , c2	2.0,2.0	As reported in (Li-Ping et al. (2005))
r1, r2	[0-1]	

Table 4.2: Setting of the Parameters for PSO.

After running the simulation, the following parameters are measured and used as indicators to test the effectiveness of the proposed algorithm. To evaluate the performance of the Parallel PSO, the following factors needed to be computed:

- **Negotiation Time (NT):** This is the total time (in seconds) that the PSO algorithm takes to reach solutions, as shown in Equation 4.10.

$$\boldsymbol{NT = \sum_{i=1}^{m} T(i)} \tag{4.10}$$

where:
NT is the total negotiation time for all tasks
i is the index of the task
m is the total number of tasks in the simulation
T (i) is the time, in seconds, taken for negotiation until an agreement is reached

- **Speedup (Sp):** This measures the improvement in execution time using parallelism. The speedup is computed by dividing the total execution time in the sequential algorithm by the total execution time using the parallel algorithms, as shown in Equation 4.11.

$$\mathbf{Sp = T1/Tc} \tag{4.11}$$

where:
Sp is the speedup of the algorithm
T1 is the total execution time in the sequential algorithm in seconds.
Tc is the total execution time of the parallel algorithm on c processors in seconds.

- **Average Waiting Time:** is defined in Equation 3.1**.**

- **Average Completed Time:** is defined in Equation 3.2.

- **Throughput:** is defined in Equation 3.3.

- **Execution Cost:** is defined in Equation 3.11.

- **Profit:** is defined in Equation 3.12.

- **SLA Violation:** In the evaluation of SLA negotiation, the focus is on the SLA violation, which comes from the missing deadline as defined in Equation 3.16. This is because no method for migration is presented in this phase of the model.

In Chapter 6, the model for SLA violation for both missed deadline and migration time will be evaluated as shown in Equation 3.18.

- **Imbalance Factor:** is defined in Equation 3.19.
- **Average Fitness Values**: This is the mean of the fitness values across the entire swarm. It is defined in Equation 3.20.

4.7.2 Experimental Results

The results of methodology #1 will be considered first. In this experiment, the Parallel PSO algorithms were evaluated in terms of finding the desired solution compared to sequential PSO. This was achieved by changing the number of iterations from 5 up to 50, then computing the comparison factors. Table 4.3 presents the results of the negotiation time that show that negotiation time is reduced until iteration 15, after which it then increases (see Figure 4.3). Thus, regarding the results of waiting time, increasing the iteration numbers reaches the objective of minimizing waiting time (see Figure 4.4). Regarding throughput, the results in Figure 4.5 show throughput is increased until iteration 20, after which it slowly decreases. Thus, selecting the number of iterations as 20 is reasonable given these factors.

# Iteration	Negotiation Time (seconds)			Average Waiting Time (seconds)		
	PSO	SPPSO	ASPPSO	PSO	SPPSO	ASPPSO
5	6931	3439	2888	128	114	66
10	8600	5488	3494	122	111	63
15	3686	2835	2106	118	97	56
20	4127	3517	2679	98	85	62
25	4913	3254	2974	92	76	58
30	5411	4021	3189	88	74	57
35	5815	4267	3256	84	67	52
40	6457	3036	3571	78	61	51
45	6999	3966	3660	69	60	48
50	7137	3523	2896	66	57	41

Table 4.3: Negotiation Time and Average Waiting Time Results (in seconds).

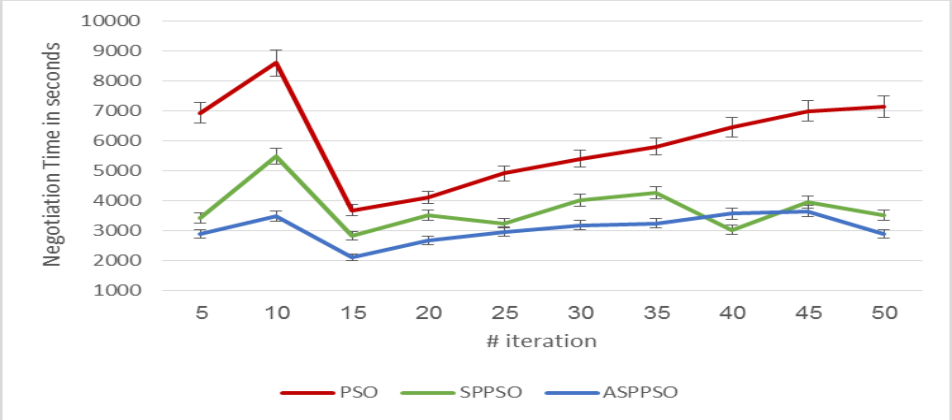

Figure 4.3: Negotiation Time Results (data shown with 95% confidence intervals).

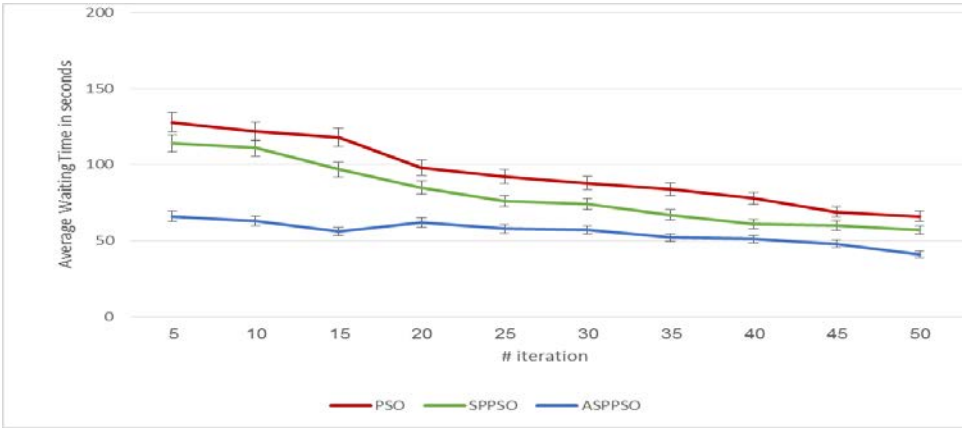

Figure 4.4: Average Waiting Time Results (data shown with 95% confidence intervals).

# Iteration	PSO	SPPSO	ASPPSO
5	62	64	78
10	65	67	74
15	66	68	77
20	68	72	82
25	66	75	79
30	65	72	76
35	67	73	81
40	69	75	82
45	69	77	79
50	71	75	78

Table 4.4: Throughput Results**.**

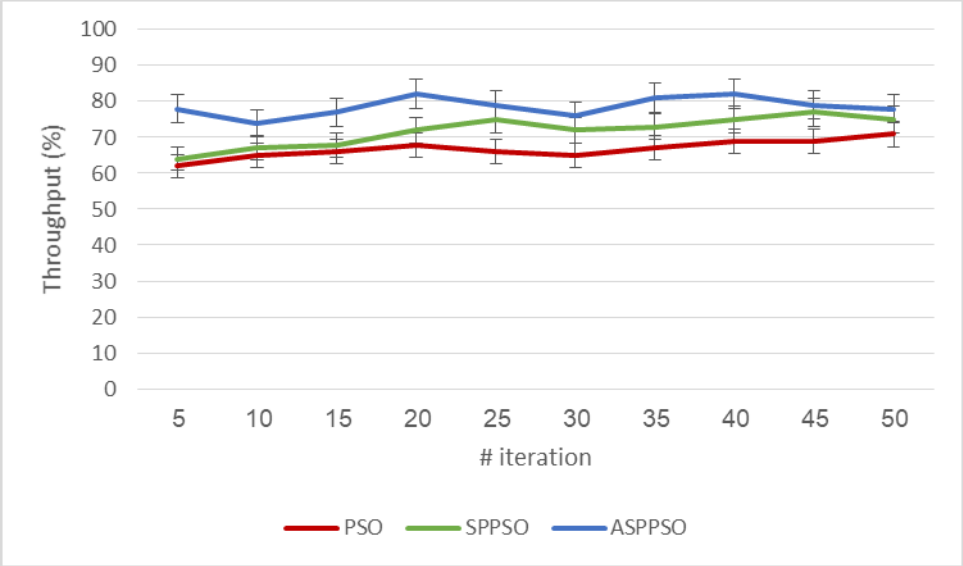

Figure 4.5: Throughput Results (data shown with 95% confidence intervals).

To illustrate the effectiveness of the ASPPSO compared to sequential PSO, the average fitness values for both are compared. The results of SPPSO are not considered because they are approximately similar to those of the ASPPSO. The results in Figure 4.6 show that the ASPPSO fitness values change as the number of iterations increases, while in the PSO the convergence speed is low while the fitness values remain steady and do not change. This means that the PSO may fall in local optima and there is no improvement in the results as the number of iterations increases.

# Iteration	PSO	ASPPSO
5	49	81
10	52	70
15	51	62
20	50	65
25	55	59
30	52	60
35	54	70
40	56	61
45	53	73
50	51	78

Table 4.5: Average Fitness Value Results.

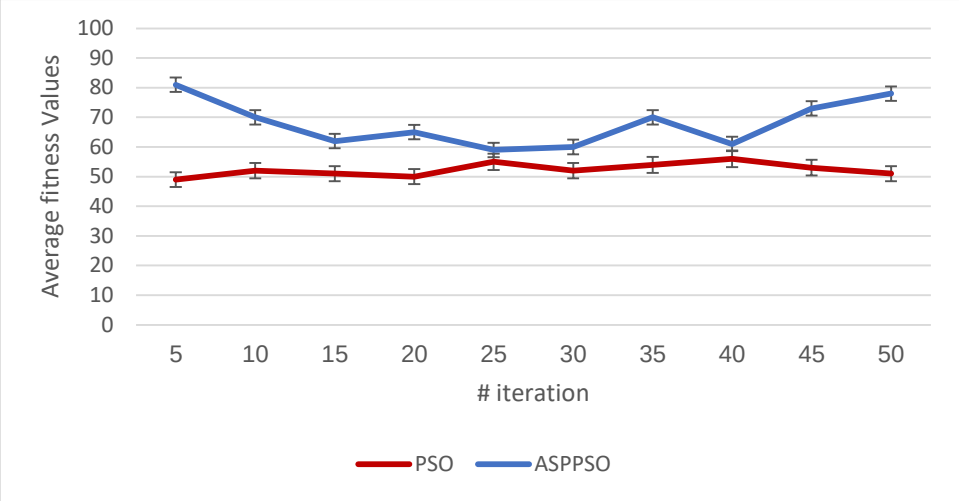

Figure 4.6: Average Fitness Value Results (data shown with 95% confidence intervals).

The results of methodology #2 will now be considered. These involve comparing the performance of the two versions of the Parallel PSO algorithms and the sequential PSO algorithm in terms of negotiation time, speedup, waiting time, completed time, and throughput. When the negotiation time in the SPPSO and ASPPSO algorithms was evaluated and compared with the results of the sequential PSO algorithm, it was found that the ASPPSO algorithm takes the least amount of time. This is because ASPPSO reduces the time spent processing by using multiple swarms in different nodes and updates the particles based on the swarm that finishes first. Specifically, the negotiation time is the highest in sequential PSO (as shown in Figure 4.7).

# Task	Negotiation Time			Speedup Results	
	PSO	SPPSO	ASPPSO	SPPSO	ASPPSO
50	1395	590	469	1.364407	1.974414
100	1668	950	738	1.455789	2.260163
150	2909	1504	1205	1.934176	2.414108
200	3396	2268	1688	1.997354	2.911848
250	4944	3083	2176	2.361425	3.272059
300	6255	3568	2500	2.660032	3.502
350	9596	3869	3063	3.301751	3.932876
400	10022	4365	3400	3.695991	4.176471
450	12379	4714	3954	3.826008	4.507537
500	14428	5813	4450	4.248202	4.742247

Table 4.6: Negotiation Time (in Seconds) and Speedup Results.

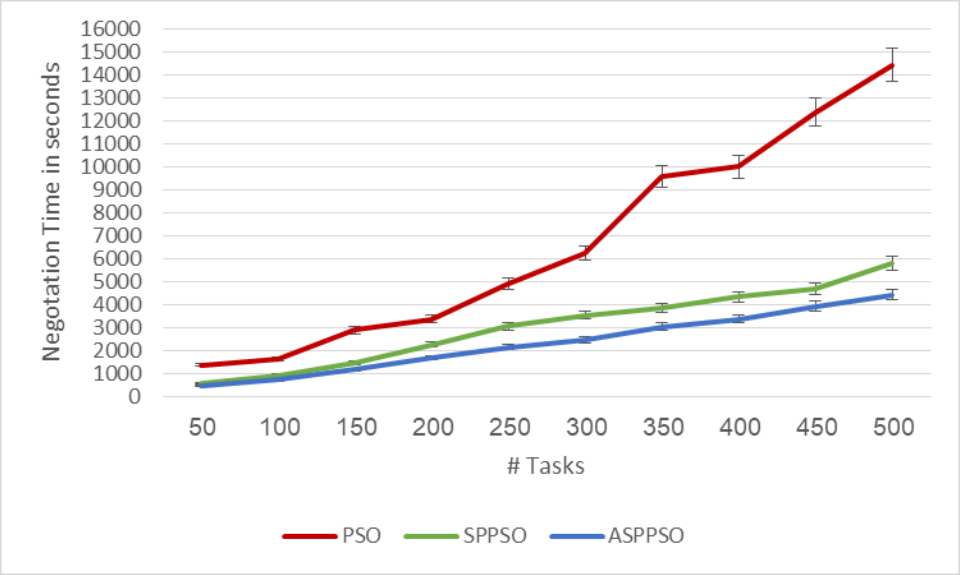

Figure 4.7: Negotiation Time Results (data shown with 95% confidence intervals).

Table 4.6 presents the speedup results of the SPPSO and ASPPSO algorithms. From these, it is apparent that the speedup when applying ASPPSO is higher than when using SPPSO. For example, running 300 tasks in ASPPSO gives 2.5, while executing the tasks using SPPSO gives 1.9. This means that the speedup in APPOS is increased by a ratio of approximately 20% compared to SPPSO.

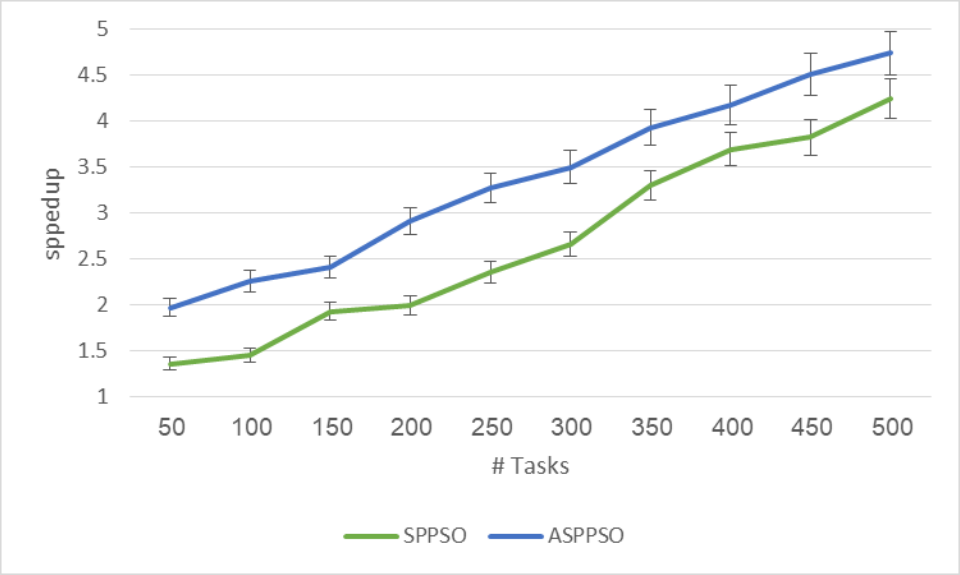

Figure 4.8: Speedup Results (data shown with 95% confidence intervals).

The results of the waiting time experiment are presented in Table 4.9 and in Figure 4.9. These show that increasing the number of tasks increases the waiting time of the tasks. The waiting time includes mapping time and the time taken to schedule the tasks until they begin execution. ASPPSO gives the shortest waiting time compared to the other algorithms because it searches for the optimum VM for each task and does not wait for all swarms to finish searching. The ASPPSO and SPPSO algorithms give shorter waiting times compared to PSO especially as the number of tasks increases. For example, with 450 tasks they reduce the waiting time by 20% using SPPSO and 30% using ASPPSO.

# Task	Average Waiting Time			Average Completed Time		
	PSO	SPPSO	ASPPSO	PSO	SPPSO	ASPPSO
50	16.17	36.14	17.3	3150	1437	1365
100	62.01	106	97.27	9487	3535	2980
150	147	149.7	148.53	12280	6894	5231
200	201	215.98	195.7	15306	7295	6033
250	239.8	224.25	205.25	19005	7679	7121
300	295.04	290.6	261.5	22717	10184	8972
350	370.2	369.25	348.5	24236	11394	10594
400	426	417	384.6	25605	15455	13281
450	515.8	468.95	456.6	27751	17339	13535
500	527.3	486.4	476.42	32103	19529	15747

Table 4.7: Average Waiting Time and Average Completed Time Results (in seconds).

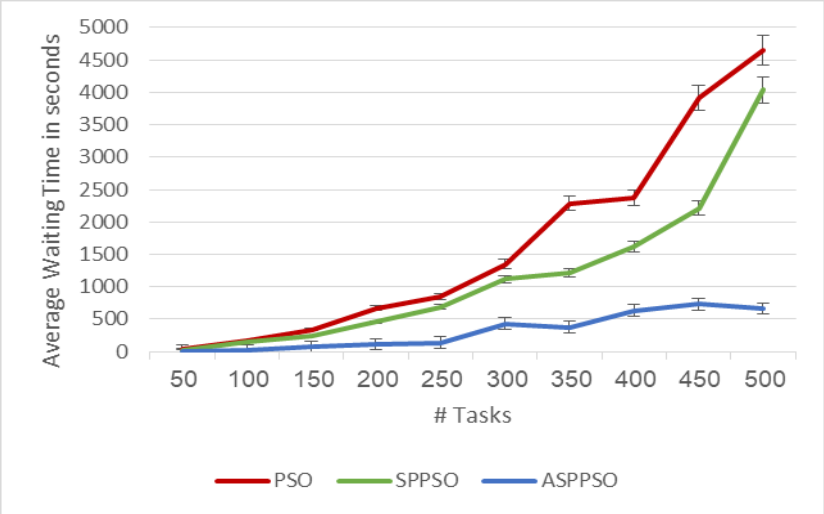

Figure 4.9: Average Waiting Time (data shown with 95% confidence intervals).

The results of completed time show that the ASPPSO algorithm takes the least time to complete the tasks compared to SPPSO and PSO, as shown in Figure 4.10. From Table 4.7, the results show that both SPPSO and ASPPSO offer good performance in terms of reducing the time taken to complete the tasks compared to PSO. This is because our model tries to select the best mapping for each task, which reduces the mapping time. The results in Figure 4.10 show that the time for completing tasks using the ASPPSO algorithm improves by 15% compared to the SPPSO and by 35% compared to the sequential PSO.

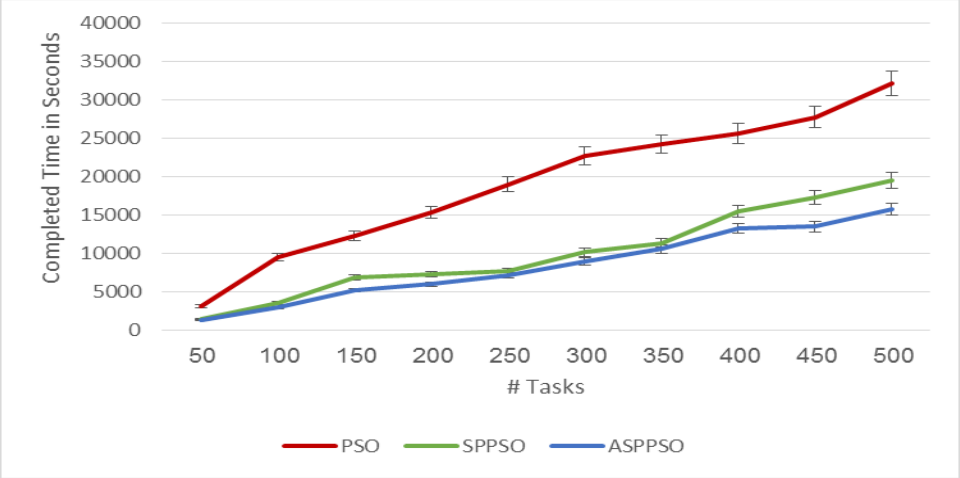

Figure 4.10: Average Completed Time Results (data shown with 95% confidence intervals).

The throughput measurement indicates the performance of the proposed model when executing many tasks in a small amount of time. The results in Table 4.8 show that the model offers good performance in terms of throughput when many tasks are involved. This is because our model tries to select the best mapping for each task, which increases the number of tasks executed in a short time. By comparing the results in Figure 4.11, it is clear that the throughput of the ASPPSO algorithm is the highest. It improves the throughput by 10 % compared to the SPPSO and by 20% compared to the sequential PSO.

# Task	PSO	SPPSO	ASPPSO
50	48.9	54.8	55.19
100	60.7	62.15	66.21
150	61.05	65.85	68
200	62.2	68.75	70.3
250	65.45	70.75	72.14
300	66.5	71.88	73.2
350	68.35	72.88	74.5
400	70.3	73.59	75
450	72.2	74.4	76.3
500	73.39	75.35	77

Table 4.8: Throughput Results.

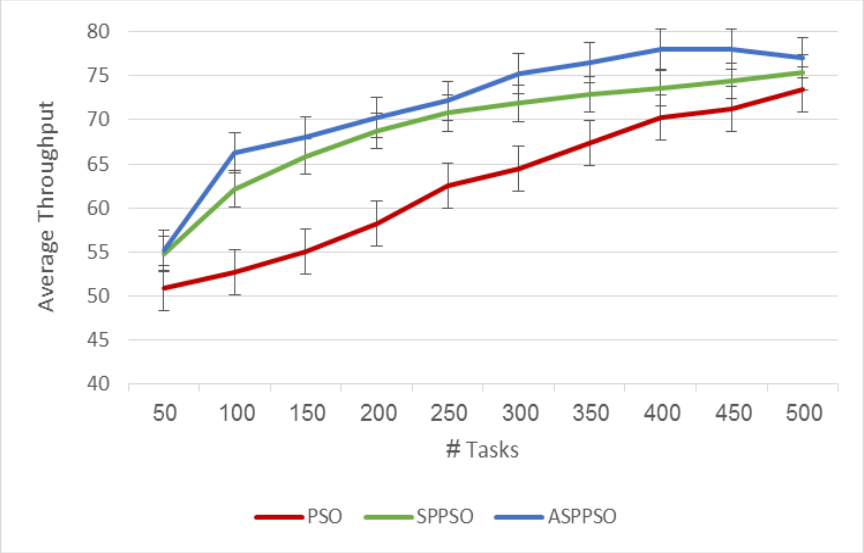

Figure 4.11: Throughput Results (data shown with 95% confidence intervals).

When the SLA violation rates are evaluated to check the number of tasks that missed deadlines when applying the SLA Negotiation algorithm, it was found that the SLA violation rates using ASPPSO give the lowest value. This means that ASPPSO reduces the number of violations and is more efficient in satisfying the QoS than PSO and SPPSO. This is because the deadline factor is taken as a constraint of the evaluation function in Equation 4.9. With a small number of tasks, the difference between algorithms is not great but with an increased number of tasks, the difference becomes substantial.

# Task	PSO	SPPSO	ASPPSO
50	17.4	16	10.6
100	20.8	18.5	15.5
150	24.7	21.3	19.8
200	28.6	24.5	22.16
250	36.2	29.8	25.9
300	44.6	33.7	27.6
350	53.4	36.6	29.8
400	58.3	42.8	34.2
450	64.5	50.6	39.3
500	70.7	65.7	44.9

Table 4.9: SLA Violation Rate Results.

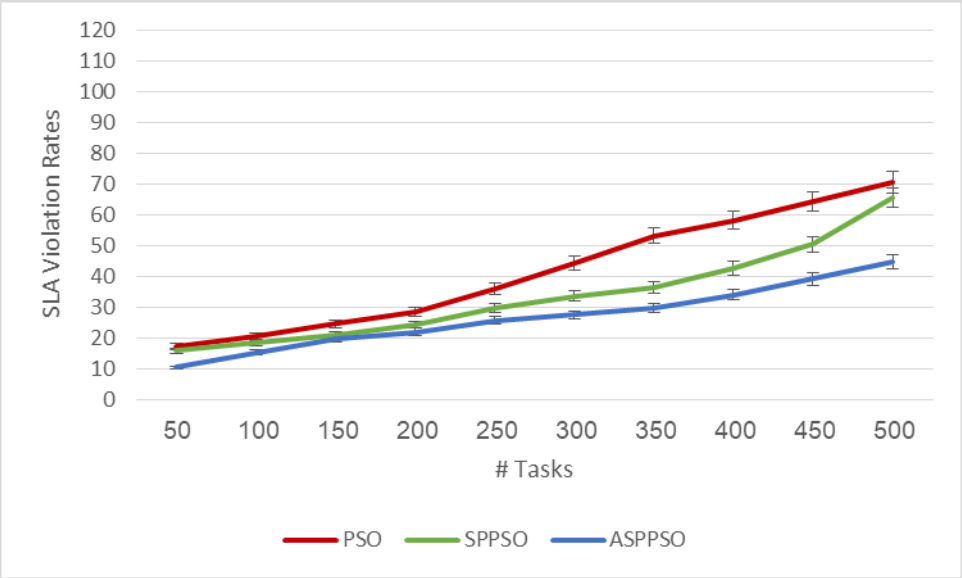

Figure 4.12: SLA Violation Rate Results (data shown with 95% confidence intervals).

When the total profit is computed to check the effect of providing QoS in the SLA after applying the SLA Negotiation algorithm, it was found that the total profits using ASPPSO gives the highest value as shown in Figure 4.13. This means that ASPPSO increases the profit by about 15% more than SPPSO and 25% more than PSO. This is because it reduces the SLA violation rates.

# Task	PSO	SPPSO	ASPPSO
50	570	627	754
100	896	905	1164
150	1034	1134	1305
200	1296	1673	2003
250	1776	2189	2512
300	2014	2632	2990
350	2465	3090	3532
400	2834	3471	3937
450	3461	3910	4420
500	3971	4265	4708

Table 4.10: Total Profit Results.

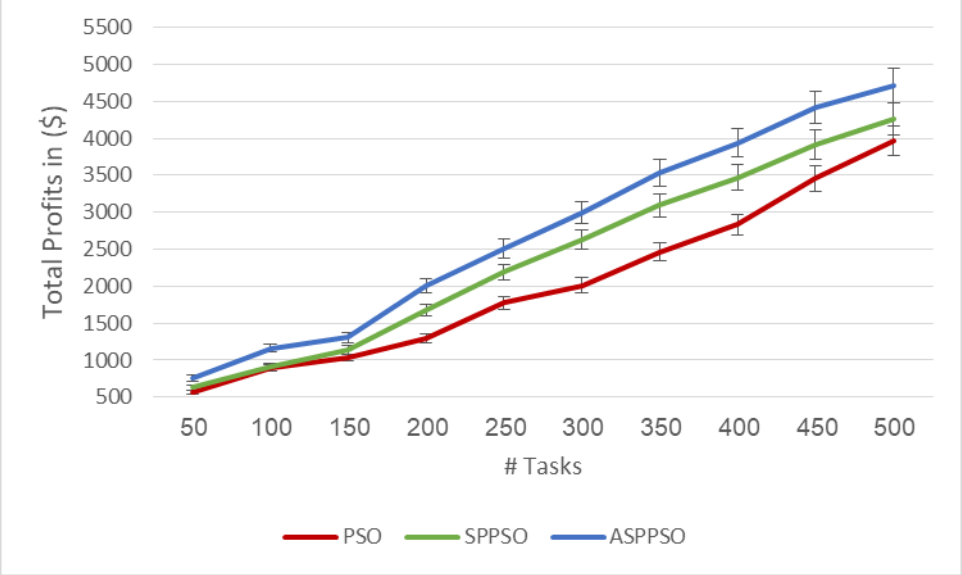

Figure 4.13: Total Profit Results (data shown with 95% confidence intervals).

4.8 Summary

The main goals of this phase were to improve the QoS in terms of throughput and waiting time, and increase the profits. Thus, a model for SLA negotiation in cloud computing was proposed to reduce the negotiation time and increase the throughput of the system. The Parallel PSO algorithm was used to optimize negotiation between the cloud consumers and the cloud providers. Two versions of the Parallel PSO algorithms were developed: SPPSO and ASPPSO. The results of the negotiation strategy were used in task scheduling to find the best VM for each task. These improvements increased the

speed of the negotiation process. Specifically, they increased the performance in terms of both waiting time and throughput. Furthermore, there is an improvement in ASPPSO compared to SPPSO in terms of waiting time and throughput.

The main findings from the experimental analysis are as follows:

- Comparing the ASPPSO results with the SPPSO results shows an improvement in waiting time of up to 20% and in completed time of up to 15%.
- The ASPPSO algorithm shows an improvement in performance compared to the PSO algorithms of approximately 35% for waiting time and 30% for completed time.
- The throughput in the ASPPSO algorithm increased by about 10% compared to the SPPSO and by approximately 20% compared to the PSO.
- The speed of ASPPSO increased by about 20% compared to the SPPSO.
- The average fitness values of the ASPPSO converged more quickly than those of the PSO algorithm.
- Profits when applying ASPPSO increased by about 15% more than with SPPSO and by 25% more than with PSO.
- SLA violation rates when using ASPPSO decreased by 15% compared to SPPSO and 25% compared to PSO.

Chapter 5

Task Scheduling Based on Many-Objective Particle Swarm Optimization

In this chapter, the task scheduling technique based on the MaOPSO algorithm will be discussed. This is the second phase of the proposed model as described in Section 1.3. Section 5.1 provides a general overview of task scheduling in cloud computing. In Section 5.2, the design of the proposed MaOPSO algorithm will be presented in detail. Section 5.3 describes the main issues involved in implementing the MaOPSO algorithm and the main configuration used to run the simulation. In Section 5.4, the methodologies and the results of the evaluations will be discussed. Finally, Section 5.5 summarizes the primary contributions of the task scheduling phase.

5.1 Overview

Task scheduling is one of the most important research fields, because it needs to be optimized to produce efficient performance in a cloud environment. Task scheduling in cloud environments aims to find a sub-optimal solution in quick time, which involves mapping the task to resources in order to meet the required objectives. PSO algorithms have been shown to find sub-optimal solutions within a reasonable amount of time (as discussed in Section 2.2.1). When the number of tasks and VMs increases, the task scheduling process becomes a challenge because the complexity of mappings is increased and this complexity is increased further if many objectives are evaluated. As argued in Section 2.4.6, this research will focus on optimizing the scheduling algorithms to handle many objectives in a short time (compared to current solutions).

However, the process of finding the best mapping is based on simultaneously meeting the objectives of both consumers and providers. For the consumer, QoS in terms of reduced waiting time and less cost is preferable, while for providers the utilization of the system and profit are the main objectives. As presented in Chapter 2, MOO has been developed with PSO to deal with multiple objectives. The available solutions are effective with two or three objectives; however when the number of objectives is greater than three new or modified methods are required (as discussed in Section 2.2.2.2). To address this issue, MaOPSO was developed to deal with many objectives in a short time. It improves the methodology of evaluating multi-objective based on simple ranking that are presented in (Alkayal et al. (2016)). Thus, a modified ranking methodology is presented that will evaluate the objectives in less time than the Pareto set and weighted sum methods (as discussed in Section 2.2.2.2). Specifically, the presented MaOPSO algorithm aims to improve the efficiency of scheduling tasks over VMs in each selected data center to minimize mapping time. This, in turn, will improve the waiting time and increase the throughput of the system. The goal of scheduling in our algorithm is to submit each task to VMs inside the data center to minimize waiting time and execution cost and at the same time, increase the throughput and providers' profit by increasing the number of successfully executed tasks within the deadline limits.

Thus, after the PPSO SLA negotiation phase has finished, as shown in Algorithm 4.1 (see Chapter 4), each task is dispatched to the selected data center through the Dispatcher Agent in the Manager Module. The second phase of the model then involves scheduling tasks over VMs inside selected data centers. In the data center Module, there is a DC Manager, which accepts tasks from the Manager Module and sends them to the Task Scheduler Agent, as shown in Figure 5.1. In the task scheduling phase, the Task Scheduler Agent in each data center schedules the tasks over VMs based on the many-objective PSO task scheduling algorithm (Algorithm 5.1).

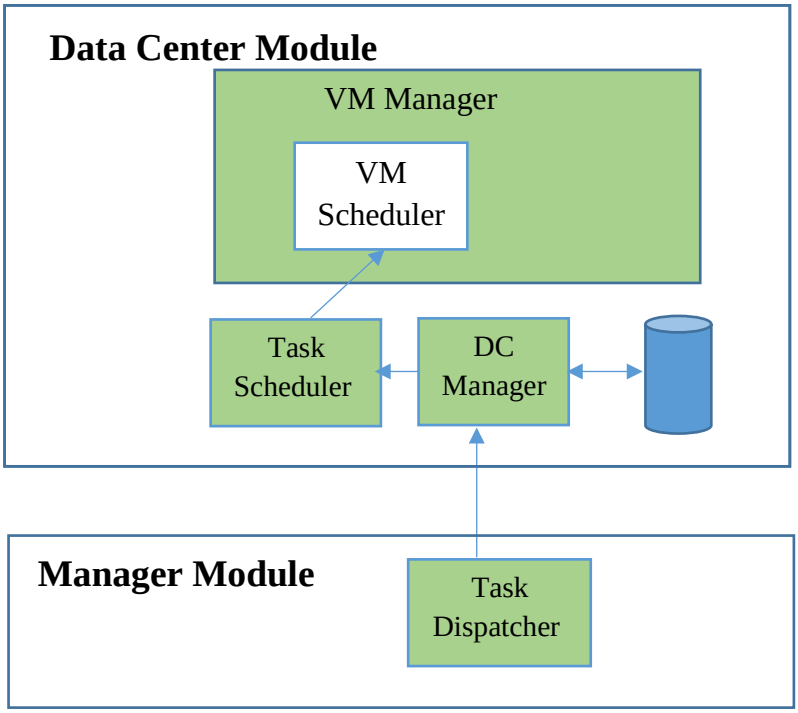

Figure 5.1: Task Scheduling Phase.

The many-objective PSO was therefore developed to deal with many-objective problems involving more than three objectives. From the literature discussed in Section 2.2.3, it is apparent that appropriate methods for applying many-objective PSO algorithms in our case including ranking and decomposition approaches. This is because a simple and quick strategy is required to evaluate many objectives in a short time. A new modified method will be used based on combining ranking strategies and the weighted sum. This will select a suitable VM for each task at a quicker time and a simpler process compared to Pareto set methods.

Thus, to address how to improve many-objective PSO, five objectives must be evaluated separately to find the best solution. These objectives are task execution time, task execution cost, data transfer time, data transfer cost, and the VM capacity (as discussed in Section 2.4). The first objective is to minimize task execution time (as shown in Equation 5.2). The second is to minimize TEC, as presented in Equation 4.5. The third is DTT, which involves minimizing the data transfer time (as shown in Equation 5.3). The fourth is to minimize DTC, which is presented in Equation 5.4. The fifth is to maximize the VM capacity (as shown in Equation 5.5). The VM with the best fitness value according to these objectives is then selected and a task is assigned. In

the following section, the design and details of the MaOPSO algorithm based on the modified ranking strategy are presented.

5.2 MaOPSO Task Scheduling Algorithm

The task scheduling problem consists of n tasks and m virtual machines. Each task must be processed by one of the VMs such that the overall scheduling time is minimized. The proposed algorithm focuses on the QoS parameters and costs relating to execution time, execution cost, data transfer time, data transfer cost and VM capacity. The algorithm follows the same rules as the model presented in Chapter 3 whereby each task can be executed on just one VM at a time and each VM handles only one task at a time.

In general, the first step in applying a MaOPSO scheduling algorithm is to represent the problem, which involves converting it from continuous to discrete values. A commonly used method that has been applied in most research is to represent the particle as a 1 × n vector of n number of VMs associated with the number of dimensions based on the number of objectives (see Section 2.2.1), as shown in Table 5.1. The value inside the particle vector is a random integer number between 1 and M where M is the number of VMs. A matrix m × n is used to represent velocity and position, where m is the number of VMs and n is the number of tasks that need to be scheduled. The elements of the position matrix can have values of either 0 or 1 with the constraint that there must be a single element with the value 1 in each column. Similarly, velocity is represented in the form of a matrix m × n. In the proposed method, the initial population is generated at random. For this purpose, the algorithm generates a random integer between 1 and M, representing the number of VMs onto which the task is mapped. Randomness in PSO initialization helps maintain population diversity and means all particles have an equal chance of being selected (Al-maamari et al. (2015)).

	T_1	T_2	T_3	T_4	…	T_n
VM	VM_1	VM_3	VM_1	VM_2	…	VM_m

Table 5.1: Particle Vector Direct Representation.

Algorithm 5.1 describes the task scheduling steps taken in the proposed model, which represents the main processes involved in scheduling tasks inside each data center. The algorithm invokes the MaOPSO algorithm (Algorithm 5.2) which finds a suitable VM for the arrival tasks (Line 1). If there is a suitable VM for the tasks then the tasks are mapped to VMs (Lines 3-4). Otherwise, the tasks status are updated with failed (Lines 5-7). The list of tasks is then updated to check if there are arrival tasks (Line 8). The steps are repeated while there is tasks in task list (Line 9). In our model, we deal with dynamic online scheduling by scheduling arrival tasks and not queue the tasks to reduce the waiting time and provide real time scheduling. Thus, the particle includes the arrival tasks with supposed the limit is 10 tasks at each particle. If the tasks more than 10, it will be kept in task list for next iteration.

Algorithm 5.1: Task scheduling algorithm.

Procedure TASKSCHEDULING (T, VMs)
Inputs: T, VMs *// list of tasks and VMs*
Outputs: mapping (T, VM)
1. **for** all arrival task in T list
2. v=MaOPSO (t, VM) *// map tasks to specific VMs by algorithm5.2*
3. **If** (v != null) *// there is suitable VM for task*
4. mapping (t, v) // maps tasks to selected VMs
5. **Else**
6. Update t status to failed
7. **End else**
8. Update T list
9. **Repeat** for step 2 until T list is empty
10. **Return** mapping
End procedure

Algorithm 5.2 includes the standard PSO (presented in Algorithm 2.4); however, instead of using one objective, it uses five objectives. It starts by defining the number of particles and initializing other parameters (Line 1). It then calculates the TET, TEC, DTT, DTC and VMC for each task with available VMs in the data center using CTET, CTEC, CDTT, CDTC and CVMC functions (Lines 2-8). These functions apply

Equations 5.1, 4.5, 5.2, 5.3 and 5.4 to compute the five objectives' values, which will be discussed in the following sections. The steps from Lines 11 to 23 are the main functions that represent the MaOPSO algorithm used to schedule tasks over VMs. The MaOPSO algorithm is improved by using the modified ranking function (as shown in Algorithm 5.2). This includes the process of computing the objective function based on five objectives, using the modified ranking strategy. Specifically, the rank value for each particle (i.e.VM ID) is determined by computing the rank of each objective, and then the smallest value of the rank value for each particle is selected. Thus, the particle with the smallest rank among all the values of the corresponding objectives is selected as the best solution.

Algorithm 5.2: MaOPSO Task scheduling algorithm.

```
Procedure MaOPSO (t, VM)
1.  Initialize (V, P, pbest, gbest, best)
2.  For each p ϵ t                                   // for each task in the particle t
3.    For each v ϵ VMs do
4.       TEC (v) ← CTEC (v)                          // cost of each task
5.       ECT (v) ←CECT (v)                           //task execution time
6.       DTT (v) ←CDTT (v)                           //data transfer time
7.       DTC (v) ←CDTC (v)                           //data transfer cost
8.       VMC (v) ←CVMC (v)                           //VM capacity
9.    End for
10. End for
11. While t < Iteration do
12.   For each p ϵ t
13.        f=Ranking (p, TET, TEC, DTT, DTC, VMC)
14.        best =SelectFitness (f)
15.        If best <pbest (p)
16.              Pbest =best
17.        End if
18.   End for
19.   If best <gbest
20.       gbest =best
21.   End if
22.   Update (V, P)                                 //using Equations 2.4, 2.5
23. End while
24. Return gbest
End Procedure
```

The particle's best fitness values are calculated according to the five factors mentioned previously (i.e. the least TET, TEC, DTT, and DTC and the highest VMC). Details of these objectives are as follows:

1. Task Execution Time:

The algorithm utilizes the task execution time as a vector for each arrival task because the proposed scheduling algorithm is dynamic; the expected time for executing each task on VM(i) is therefore computed accordingly. The TET is computed for a task in each VM, represented in a 1× n matrix where n represents the number of VMs in the data center. For each task, the TET at each VM is computed by considering the following parameters: the task length measured in Machine Instruction (MI) and the VM processing speed in MIPS. The task execution time is calculated by dividing the task length measured in MI by the VM processing measured in MIPS, as shown in Equation 5.1.

$$\boldsymbol{TET}\ (\boldsymbol{i},\boldsymbol{j})\ =\ \boldsymbol{TL}\ (\boldsymbol{i})\ /\ \boldsymbol{PSV}\ (\boldsymbol{j}) \tag{5.1}$$

where:
TET (i) is the execution time for task i
TL (i) is the length of task i measured in MI
PSV (j) is the processing speed of VM j measured in MIPS

2. Task Execution Cost:

This has been defined in Equation 4.5**.**

3. Data Transfer Time

The data transfer time for each task is computed according to the size of the task's input and output files and the bandwidth for each VM, depending on the VM type.

$$\boldsymbol{DTT}\ (\boldsymbol{i},\boldsymbol{j})\ =\ \boldsymbol{FS}\ (\boldsymbol{i})\ /\ \boldsymbol{VMB}\ (\boldsymbol{j}) \tag{5.2}$$

where:
DTT (i, j) is the data transfer time of task i in VM j
FS (i) is the size of the task i input and output files in MI
VMB (j) is the bandwidth of VM j

4. Data Transfer Cost

The data transfer cost is computed according to data transfer time which is computed in Equation 5.2 and the cost of bandwidth of VM, as shown in Equation 5.3.

$$\boldsymbol{DTC}\ (\boldsymbol{i},\boldsymbol{j})\ =\boldsymbol{DTT}\ (\boldsymbol{i})\ *\ \boldsymbol{CostBW}(\boldsymbol{j}) \tag{5.3}$$

where:
DTC (i, j) is the data transfer cost of task i in VM j

DTT (i, j) is the data transfer time of task i in VM j, as shown in Equation 5.2
CostBW (j) is the cost of bandwidth per second using VM j

5. VM Capacity

VM capacity is computed based on the utilization of the VM in terms of CPU, memory, storage size and bandwidth, as shown in Equation 5.4.

$$\boldsymbol{VMC}\ (\boldsymbol{j})\ = (\sum_{\boldsymbol{i=0}}^{\boldsymbol{m}} \boldsymbol{Tlength}\ (\boldsymbol{i}))/\ \boldsymbol{CMIPS}(\boldsymbol{j})\ \times \boldsymbol{Ncores}\ (\text{j}) \tag{5.4}$$

where:
VMC (j) is the capacity load of the VM j
Tlength (i) is the total length for all tasks assigned to VM
CMIPS(j) is the CPU speed of VM j in MIPS
Ncores (j) is the number of cores in VM j

The modified ranking strategy of the MaOPSO algorithm can now be considered. The novel contribution of this work concerns the method used to evaluate the objective functions, which involves ranking each objective to select the best solution using the MaOPSO algorithm. In this research, the modified ranking function is invoked to evaluate the objectives in MaOPSO. Using this function, the ranking strategy is applied to evaluate the solutions in each iteration, where each objective is represented by a two-dimensional matrix. Algorithm 5.3 illustrates the main steps for computing the fitness value, which involves ranking the solution according to the objective functions, thus improving the MaOPSO task scheduling algorithm.

The design of the fitness function is based on the sum ranking and minimum ranking strategies. The minimum ranking of the objectives is summarized in Equation 5.5, and the sum of ranking is presented in Equation 5.6. Finally, the weighted sum is computed for the two ranks to find the final rank of the solution x as shown in Equation 5.7.

$$\mathbf{F1}(\mathbf{x})\ = \mathbf{Min}\ (\mathbf{rank}\ (\mathbf{TEC}), \mathbf{rank}\ (\mathbf{TET}), \mathbf{rank}\ (\mathbf{DTT}),\ \mathbf{rank}\ (\mathbf{DTC}),\ \mathbf{rank}\ (\mathbf{VMC})) \tag{5.5}$$

$$\mathbf{F2}(\mathbf{x}) =\ \mathbf{rank}\ (\mathbf{TEC}) + \mathbf{rank}\ (\mathbf{TET}) + \mathbf{rank}\ (\mathbf{DTT}) + \mathbf{rank}\ (\mathbf{DTC}) + \mathbf{rank}\ (\mathbf{VMC}) \tag{5.6}$$

$$\mathbf{Min\ F}(\mathbf{x})\ =\ \mathbf{0.5}\ *\mathbf{F1}(\mathbf{x})\ +\ \mathbf{0.5}\ *\ \mathbf{F2}(\mathbf{x}) \tag{5.7}$$

Regarding the proposed ranking, Algorithm 5.3 illustrates the main steps for computing the fitness value based on the modified ranking of solutions according to the objective functions in order to improve the MaOPSO task scheduling algorithm. The algorithm begins by computing the minimum rank, following which the sum rank is computed as shown in Lines 2 and 3 for all objectives. The rank value for the particle is computed using the weighted sum method of the sum rank and minimum rank as shown in Line 4. Finally, the final rank for particle p is selected as a minimum rank as shown in Line 6, then it returned to continue the process of Algorithm 5.2 (Line 7).

Algorithm 5.3: Modified Ranking Strategy Algorithm.

Procedure MaOPSO (p, TET, TEC, DTT, DTC, VMC**)**
1. **For** each v ϵ p **do**
2. f1 (v) = Min (p, TET, TEC, DTT, DTC, VMC)
3. f2 (v) = Sum (p, TET, TEC, DTT, DTC, VMC)
4. rank (v) =f1 (v)*0.5 + f2 (v) *0.5
5. **End for**
6. r=min (rank)
7. **Return** r

End Procedure

To illustrate this approach, suppose there is one task with five VMs and the five objectives for each VM are computed using Equations 5.1, 4.5, 5.2, 5.3 and 5.4. The results are shown in Tables 5.2 - 5.6. The ranking of the VMs is based on the minimum rank and sum rank values, which involves sorting them in ascending order. The rank of the objectives is then computed by applying weighted sum, as shown in Equation 5.7, and the VM with the lowest rank value is selected, as shown in Table 5.7. All the shaded cells in Tables 5.2-5.7 represent the best solution, i.e. the best VM for the task.

The values of TET					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	10	12	8	15	12

The rank values after sorting					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	2	3.5	1	5	3.5

Table 5.2: Task Execution Time (TET).

The values of TEC					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	21	24	30	22	25

The rank values after sorting					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	1	3	5	2	4

Table 5.3: Task Execution Cost (TEC).

The values of DTT					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	10	14	8	12	16

The rank values after sorting					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	2	4	1	3	5

Table 5.4: Data Transfer Time (DTT).

The values of DTC					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	20	26	20	21	30

The rank values after sorting					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	1.5	4	1.5	3	5

Table 5.5: Data Transfer Cost (DTC).

The values of DTC					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	13	26	22	16	30

The rank values after sorting					
Task	**V1**	**V2**	**V3**	**V4**	**V5**
T1	5	2	3	4	1

Table 5.6: VM Capacity (VMC).

The value of the rank of each objective in a specific VM is then summed to find the rank of VM for this task. However, if two VMs are equal, for example VM1 = 6.25 and VM3= 6.25, then both have the smallest rank. The strategy to adopt in this case is that if there are many VMs with an equal smallest value, the first in the sequence is chosen which in this case is V1.

Task	**V1**	**V2**	**V3**	**V4**	**V5**
Min rank	1	2	1	2	1
Sum rank	2+1+2+1.5 +5 =11.5	3.5+3+4+4 +2 =16.5	1+5+1+1.5 +3 =11.5	5+2+3+3+ 4 =17	3.5+4+5+ 5+1 =18.5
Total	1*0.5+11.5 *0.5=6.25	0.5*2+0.5* 16.5=9.25	1*0.5+0.5* 11.5=6.25	2*0.5+0.5 +17=9.5	1*0.5+18. 5*0.5=9.7 5

Table 5.7: VMs Rank Values.

5.3 MaOPSO Task Scheduling Implementation

MaOPSO is implemented using CloudSim 3.0.3. The simulation model involves many data centers with different specifications, as shown in Chapter 3. Virtual machines are created to provide cloud services from the data center and many tasks are simulated using data from the dataset file. The details of these resources are presented in Section 3.4.1 in Tables 3.1, 3.2, and 3.3. The value of parameters used to apply the MaOPSO algorithm are listed in Table 5.8, which includes the number of particles, the maximum number of iterations and other parameters of PSO algorithms.

PSO Parameters	Values	Reason
# particles	10	Depends on the number of data centers
# iterations	5-500	Based on the results of the tests of fitness function , there is no further change after 500 iterations
w	[0-1]	Specified based on Equation 2.8
c1 ,c2	2.0,2.0	As reported in (Li-Ping et al. (2005))
r1,r2	[0-1]	

Table 5.8: The Setting of the Parameters for PSO.

To implement MaOPSO, several classes from CloudSim were used and modified to represent cloud environment such as classes for defining Data center, Host, VM, and Cloudlet. In addition, classes for managing resources such as CloudletScheduler, VMScheduler, and DatacenterBroker were modified to run in the proposed model. Moreover, new classes such as Task Dispatcher, Ranking, MaOPSO, Particle, and Problem Set classes were also added.

5.4 Experimental Evaluation

It is important to point out that the aim in this chapter is testing the effectiveness of the MaOPSO in scheduling task only before discussing the methods that will used to conduct the experimental tests and evaluate the effectiveness of the MaOPSO in task scheduling by comparing it with other methods. It can therefore be assumed in this case that after the selection of SLA negotiation, the tasks are scheduled inside the data center; the VM is then scheduled to available hosts based on FCFS.

In terms of evaluation, several experiments were conducted to measure the objectives of our model.

- **First**, several experiments were performed to analyze the effectiveness of MaOPSO and determine the best parameters for running the algorithm by changing the number of iterations.
- **Second**, experiments are performed to evaluate the performance of the modified ranking strategy in the task scheduling algorithm, the results of which are compared with the Pareto set and weighted sum approaches.
- **Third**, experiments are conducted to compare the scheduling task algorithm with state of the art algorithms used in task scheduling in cloud systems, which were previously discussed in Chapter 2.
- **Fourth,** experiments are performed to compare the modified ranking with the simple ranking strategy that developed in (Alkayal et al. (2016))**.**

Each experiment was run ten times and the average score across all experiments was then calculated with 95% confidence intervals. After running the simulation, the following parameters are measured and used as indicators to test the effectiveness of the proposed algorithm. To evaluate the performance of the MaOPSO task scheduling, the following factors were computed:

- **Average Waiting Time:** is defined in Equation 3.1**.**
- **Average Completed Time:** is defined in Equation 3.2.
- **Throughput:** is defined in Equation 3.3.
- **Average VM Utilization**: is defined in Equation 3.8**.**
- **Execution Cost:** is defined in Equation 3.11.
- **Profit:** is defined in Equation 3.12.
- **Imbalance Factor:** is defined in Equation 3.19.
- **Average Fitness Values:** is defined in Equation 3.20.

5.4.1 The MaOPSO Algorithm: Analysis Results

The model that has been developed aims to minimize the waiting time, completed time, and the cost. The results of the modified ranking (denoted as Ranking in the evaluation) strategy were normalized to show the effectiveness of the algorithm. The results are normalized to be represented in one figure as shown in Figure 5.2.The results in Figure 5.2 show that the ranking strategy minimizes the objectives with an increasing number of iterations until it reaches 100, after which there is no significant difference in the quality of the results. In terms of maximizing the throughput and the resource utilization, good results are shown after 50 iterations (see Figure 5.3). Thus, the maximum number of iterations in task scheduling evaluation results was selected as 100 to balance the objectives of minimization and maximization.

#Iteration	Minimizing Objectives			Maximizing Objectives	
	Waiting Time	Completed Time	Cost	Throughput	Utilization
5	0.3	0.58	0.3	0.4	0.19
10	0.25	0.38	0.3	0.42	0.3
50	0.108	0.33	0.29	0.41	0.31
100	0.102	0.29	0.28	0.45	0.29
200	0.095	0.33	0.27	0.51	0.32
300	0.11	0.35	0.25	0.52	0.33
400	0.13	0.35	0.23	0.51	0.35
500	0.13	0.38	0.2	0.54	0.35

Table 5.9: Results of MaOPSO in terms of Minimizing and Maximizing Objectives.

Figure 5.2: Results of MaOPSO in terms of Minimizing Objectives (data shown with 95% confidence intervals).

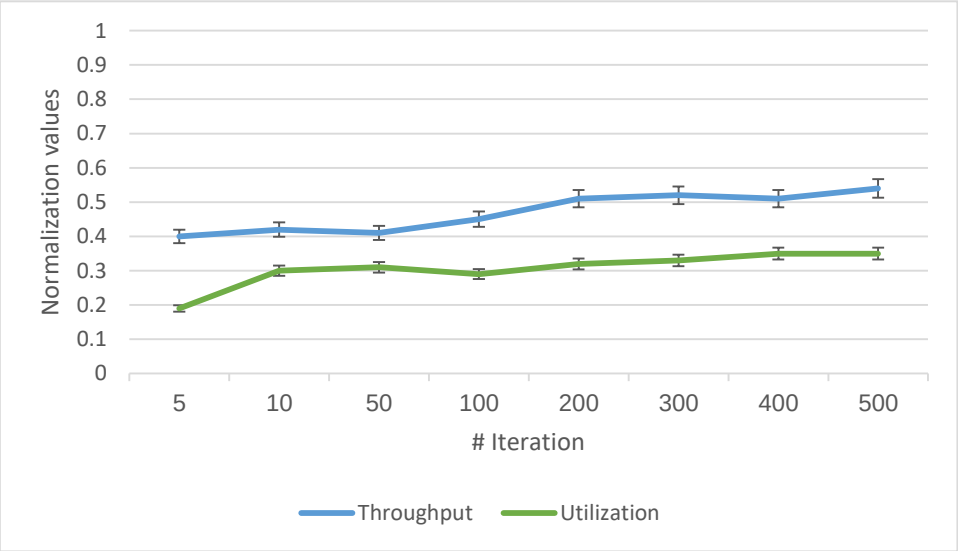

Figure 5.3: Results of MaOPSO in terms of Maximizing Objectives (data shown with 95% confidence intervals).

5.4.2 Evaluating Ranking Method Efficiency in MaOPSO Task Scheduling

In this section, the effectiveness of the ranking in dealing with five objectives will be assessed and compared to the weighted sum of objectives and the Pareto optimal solution using a non-dominant set. In this research, the weighted sum is computed by multiplying each objective with the weight that represents its importance in the system. The values of the objectives that need to be maximized are positive and those that should be minimized are negative. For simplicity, equal weights are used for each objective because they all have the same importance in this model, as shown in Equation 5.8. To analyze the performance of the method, the results of processing time and average fitness will be compared. These results are presented in Figures 5.4 and 5.5.

$$\boldsymbol{Rank}\ (\boldsymbol{i}) = \sum_{\boldsymbol{k=1}}^{\boldsymbol{5}} (\boldsymbol{obj}\ (\boldsymbol{i}, \boldsymbol{k}) \times \boldsymbol{w}\ (\boldsymbol{k})) \tag{5.8}$$

where:
k represents the objective value, and k=1, 2, 3, 4 and 5
obj (i, k) is the value of the objective k in the solution i
w (k) is a non-negative weight value such that $\sum w(k) = 1$, k=1, 2, 3, 4 and 5.

Specifically, the ranking strategy gives the best results in terms of reducing the processing time when compared to the weighted sum and the Pareto optimal set approaches, as shown in Figure 5.4. The processing time is the time that be consumed to finish evaluation objectives to find the results of scheduling process. In addition, with respect to the average fitness value, the results of ranking strategy converge on a specific solution, which is a weakness because the algorithm falls in to local optima. Therefore, to solve this problem, the VM utilization should be used as a factor to balance the tasks over VMs.

# Tasks	Weighted sum	Pareto	Ranking
5	366	1050	403
10	856	1520	990
50	1485	3907	1672
100	4567	8143	3673
150	6597	10068	4684
200	7113	12911	5515
250	9677	14156	6783
300	11978	15762	8175
350	13537	16600	8758
400	14713	18710	10317
450	15149	20760	12353
500	17714	22450	13405

Table 5.10: Processing Time Results.

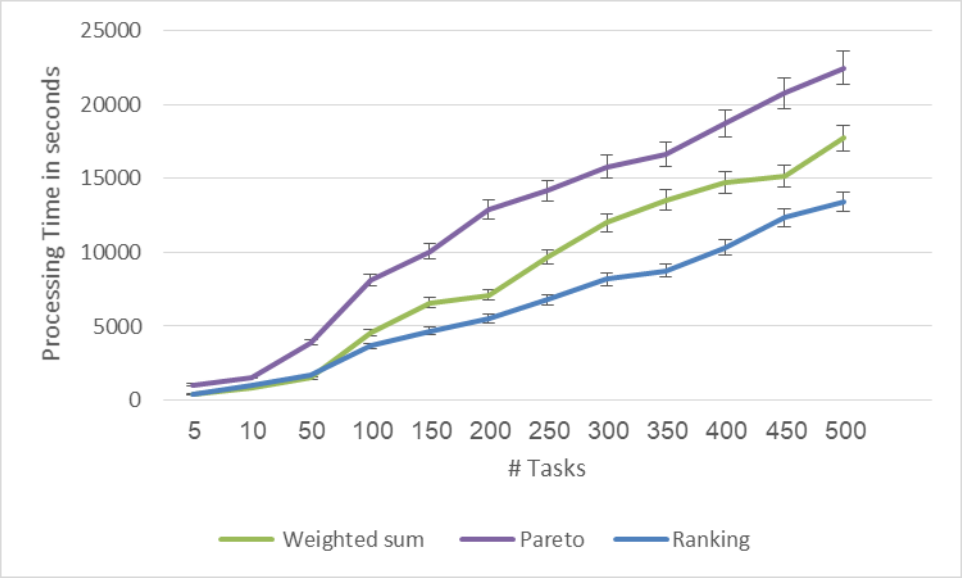

Figure 5.4: Processing Time Results (data shown with 95% confidence intervals).

# Tasks	Weighted sum	Pareto	Ranking
5	39.4	85.1	99.88
10	31.65	64.5	112.28
50	37.5	75.7	120.9
100	42.2	68.2	122.56
150	32.9	74	122.36
200	37.3	81	124.7
250	43.9	88	125.8
300	34.1	94.7	127.88
350	30.5	97.2	129.89
400	34.98	80.1	131.5
450	32.4	76.5	133.2
500	32.79	78	135

Table 5.11: Average Fitness Results.

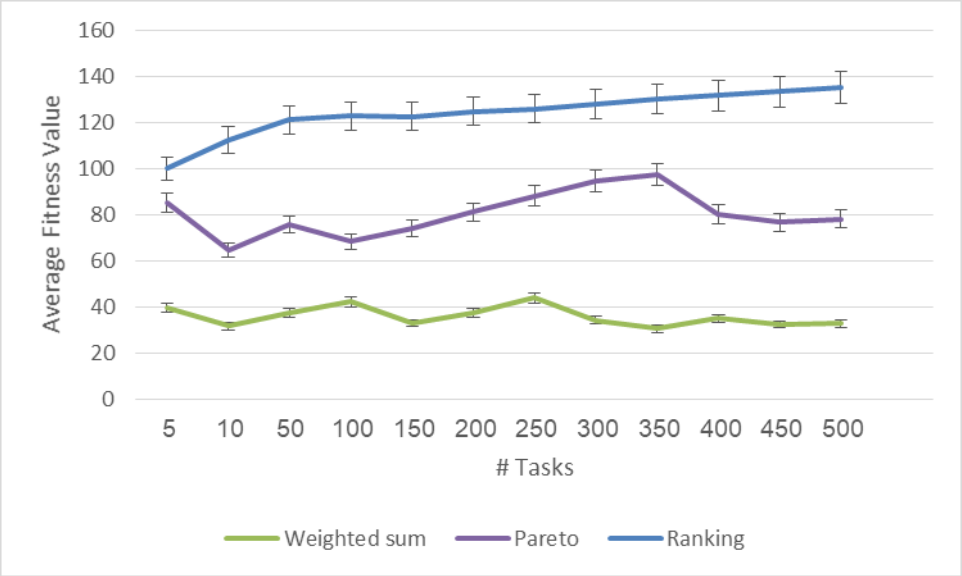

Figure 5.5: Average Fitness Results (data shown with 95% confidence intervals).

5.4.3 Evaluating MaOPSO Task Scheduling Compared to Other Scheduling Algorithms

In the research literature, different heuristic and meta-heuristic algorithms have been utilized to perform task scheduling in various fields (as discussed in sections 2.4.3 and 2.4.4). The performance of the algorithm used in this research is therefore compared with three task scheduling techniques Genetic, ACO, Max-min and Min-min algorithms

(described in sections 2.4.2, 2.4.3 and 2.4.4 respectively). Their performance is compared in terms of waiting time, completed time, average utilization of VMs, and the VM imbalance factor.

The results show an improvement in ranking strategy in terms of waiting time compared to other algorithms, especially when the number of tasks increases to more than 250. This is because the ranking strategy reduces the mapping time. With a smaller number of tasks, for example between 50-100, the Genetic algorithm gives a short waiting time compared to Minimax and Minimin algorithms, as shown in Figure 5.6. However, ACO reduces the waiting time less than the Genetic algorithm. The ranking algorithm reduces the waiting time by about 15%, and this is because it reduces the mapping time of tasks.

# Task	Ranking	Genetic	ACO	Max-min	Min-min
50	27	26.6	30	40	30
100	35	34.15	87	68	76
150	93	138	119	108	126
200	140	189	169	189	144
250	177	290	224	206	227
300	226	305	298	278	255
350	245	329	305	318	310
400	280	346	332	331	326
450	315	393	420	443	386
500	410	483	470	515	490

Table 5.12: Waiting Time Results.

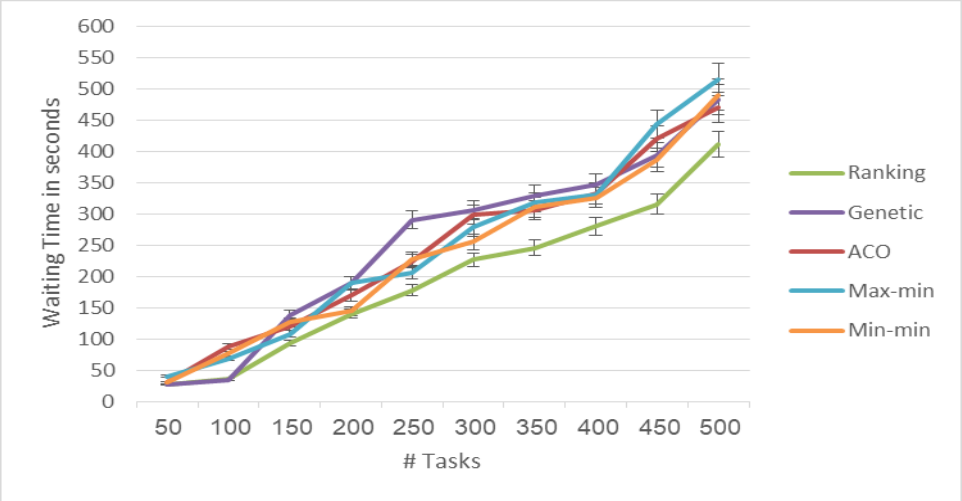

Figure 5.6: Waiting Time Results (data shown with 95% confidence intervals).

Reducing waiting time will reduce the time taken to complete tasks, as shown in Figure 5.7. The results show a decrease in completed time for the ranking strategy over other algorithms. In addition, Max-min and Min-min algorithms take a shorter amount of time compared to the Genetic and ACO algorithms.

# Task	Ranking	Genetic	ACO	Max-min	Min-min
50	330	350	345	291	304
100	348	398	372	339	351
150	382	436	417	383	368
200	419	492	455	454	437
250	454	524	487	497	477
300	483	597	524	549	525
350	510	621	566	595	577
400	536	683	583	635	592
450	551	705	628	656	632
500	590	788	657	693	677

Table 5.13: Completed Time Results.

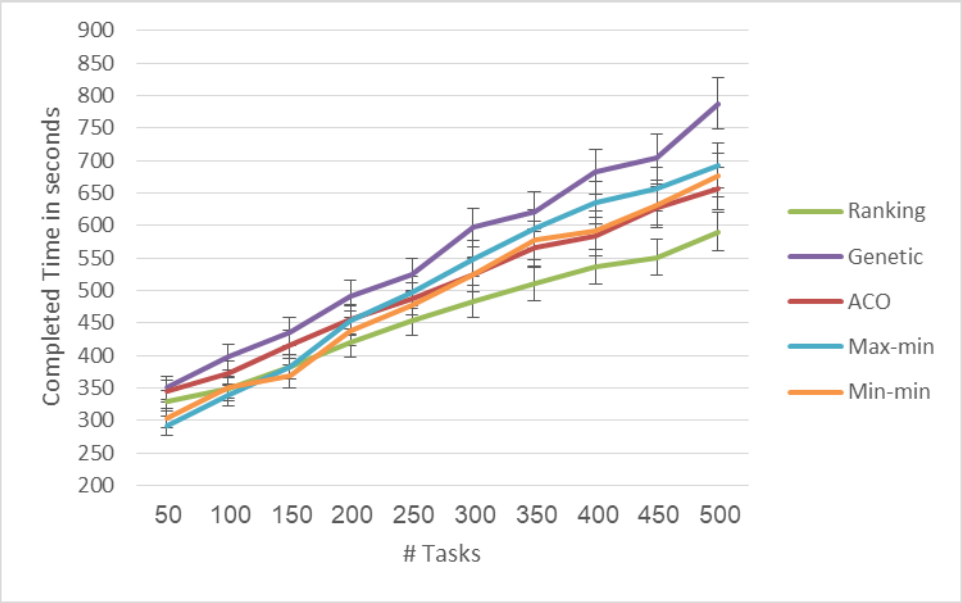

Figure 5.7: Completed Time Results (data shown with 95% confidence intervals).

With respect to the utilization of resources, the results in Table 5.14 show the best resource utilization in applying ranking strategy compared to other algorithms. Moreover, the ACO algorithm offers better resource utilization than the others; this is because they distribute the tasks over VMs during the scheduling process in an efficient

method. However, Figure 5.8 shows that the improvement in resource utilization is limited to about 10% because in this phase there is no focus on improving utilization. In the next chapter, the method of improving resource utilization associated with reduced power consumption will be considered.

# Task	Ranking	Genetic	ACO	Max-min	Min-min
50	0.53	0.38	0.39	0.22	0.35
100	0.56	0.46	0.45	0.31	0.39
150	0.58	0.49	0.48	0.35	0.42
200	0.64	0.52	0.52	0.39	0.48
250	0.67	0.56	0.63	0.43	0.52
300	0.69	0.6	0.67	0.49	0.55
350	0.73	0.65	0.69	0.53	0.57
400	0.75	0.68	0.72	0.57	0.63
450	0.78	0.72	0.75	0.62	0.65
500	0.85	0.74	0.79	0.67	0.68

Table 5.14: Average Utilization of Resources.

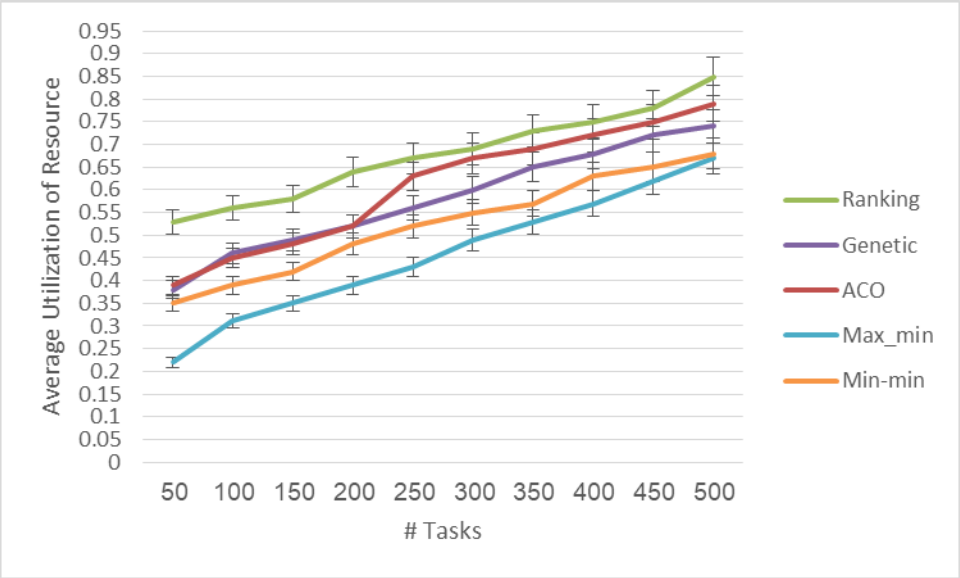

Figure 5.8: Average Utilization of Resources (data shown with 95% confidence intervals).

Regarding the results of total profits presented in Figure 5.9, the ranking strategy provides the highest profits compared to other algorithms. This is because it reduces the waiting time and executes more tasks within specific times. The profit is increased

by about 15% more than for other algorithms as a result of improving the throughput of the system.

# Task	Ranking	Genetic	ACO	Max-min	Min-min
50	6370	5269	5423	4096	4982
100	12896	10405	11754	10164	10696
150	20607	14383	17830	15395	14041
200	22636	16993	20145	19390	19556
250	27181	21788	24930	22389	23157
300	36816	26118	30150	27306	28350
350	42438	30289	36120	29470	32970
400	45285	34157	39784	31818	35038
450	48224	39810	43453	33366	38165
500	51956	44265	47403	36900	42174

Table 5.15: Total Profits Results.

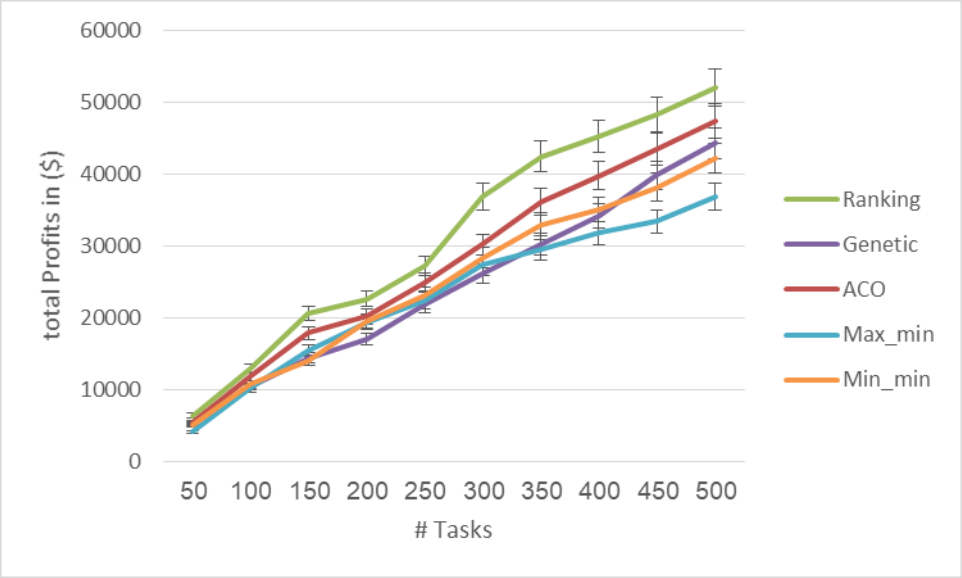

Figure 5.9: Total Profits (data shown with 95% confidence intervals).

5.4.4 Evaluating MaOPSO Task Scheduling Compared to Simple Ranking Algorithms

In this section, the proposed task scheduling algorithm is compared with the simple ranking algorithm that was presented in (Alkayal et al. (2016)). The simple ranking is

evaluated with three and five objectives to test its effectiveness with increased number of objectives.

Regarding the results of processing time presented in Figure 5.10, the ranking strategy provides the smallest processing time comparing to other algorithms. This is because it reduces the waiting time by about 15% less than for other algorithms. The simple ranking provides better results with three objectives than with five ones.

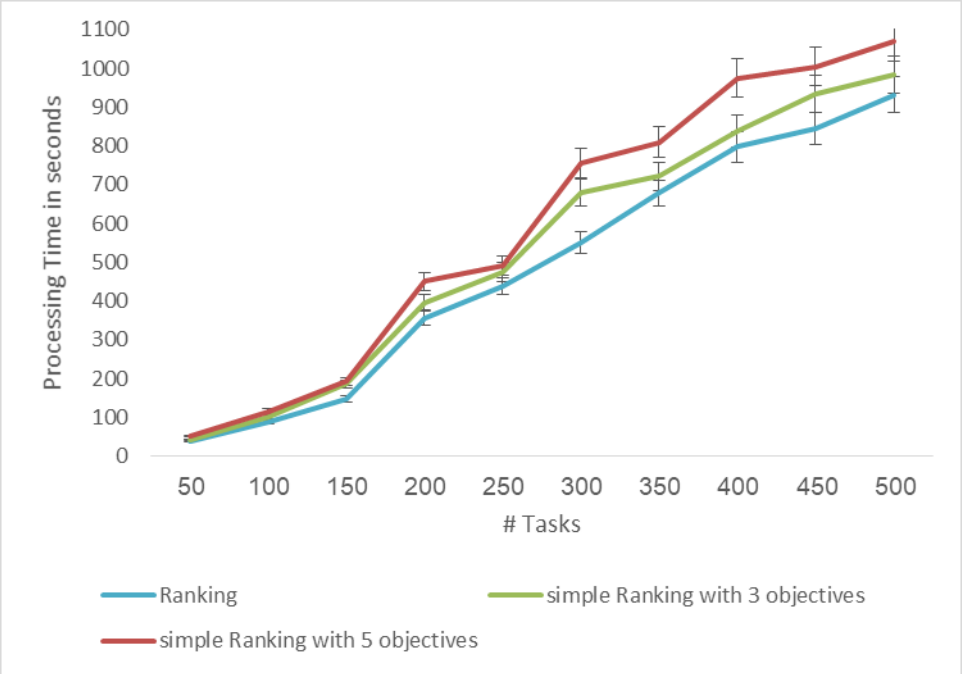

Figure 5.10: Processing Time Results (data shown with 95% confidence intervals).

Regarding the results of average waiting time presented in Figure 5.11, the ranking strategy provides the lowest compared to other algorithms. This is because it reduces the processing time as shown in Figure 5.10. The waiting time is decreasing by about 15% more than for other algorithms.

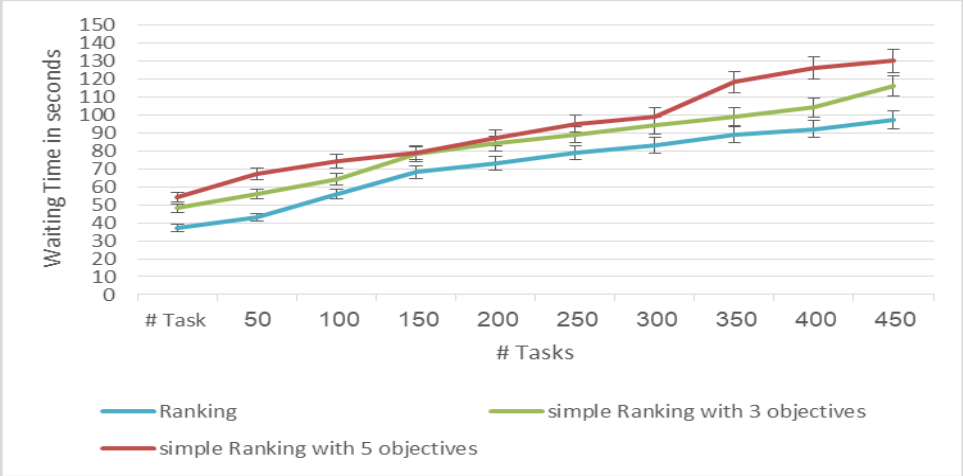

Figure 5.11: Waiting Time Results (data shown with 95% confidence intervals).

Regarding the results of total throughput presented in Figure 5.12, the ranking strategy provides the highest throughput compared to other algorithms. This is because it reduces the waiting time and execute more tasks within specific times. The throughput is increased by about 25% more than for other algorithms. The simple ranking provides higher throughput with three objectives than with five objectives.

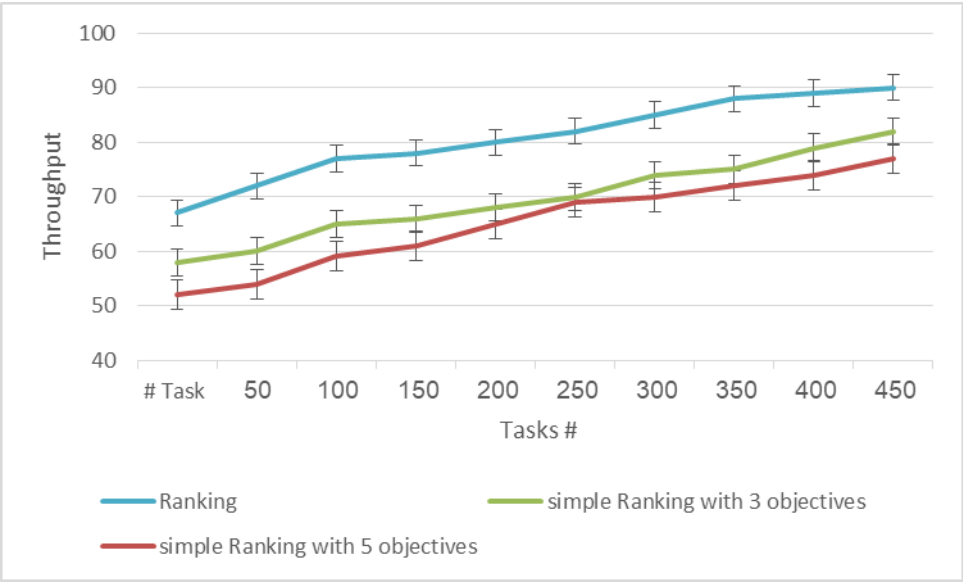

Figure 5.12: Throughput Results (data shown with 95% confidence intervals).

5.5 Summary

MaOPSO task scheduling in cloud computing is applied to distribute tasks over VMs inside data centers. It was optimized by applying a many-objective PSO algorithm using a modified ranking strategy. This modified strategy is an improvement and extension of the simple ranking strategy that presented in (Alkayal et al. (2016)). In so doing, this work advanced the field by improving task scheduling in terms of QoS, resource utilization, and profits. Thus, a new modified ranking strategy has been devised to evaluate the five objective functions: TET, TEC, DTT, DTC and VMC in a short space of time. The modified ranking method combines the weighted sum approach with ranking strategies to provide an evaluation method that can be adapted in line with an increase in the number of objectives.

The results of proposed ranking strategy were evaluated and compared with the simple ranking strategy, weighted sum and Pareto set approaches. The results from the ranking strategy are used in MaOPSO task scheduling to find the most appropriate VM for each task. This improvement accelerates and simplifies the process of evaluating objectives in MaOPSO algorithms when compared to other approaches. Because of these improvements, this work has increased performance by increasing the throughput and reducing waiting time. Moreover, it has provided an improvement of up to 20% in completion time compared to the weighted sum approach.

The main findings from the experimental analysis are as follows:

- Comparing the ranking results with the weighted sum results and Pareto set shows an improvement in processing time of up to 10% and in completed time of up to 15%.
- The throughput in the ranking algorithm increased by about 10% compared to the Min-min and by approximately 15% compared to the Genetic Algorithm.
- The completed time for ranking decreased by about 20% compared to the other benchmarks.
- The average fitness values of the ranking converged more quickly than the Genetic algorithm.
- The profits increased by about 15% more using the ranking strategy when compared to other algorithms.

Chapter 6

Virtual Machine Allocation using Particle Swarm Optimization

The VM allocation using the PSO algorithm, which is the third phase of the proposed resource allocation model described in Chapter 3, is discussed in this chapter in detail. In the first section, an overview of the VM allocation process and key objectives are presented. Section 6.2 then presents and describes the clustering algorithm based on K-means and PSO that was used to cluster hosts in the data centers. In Section 6.3, the VM scheduling algorithm based on a many-objective PSO algorithm and modified ranking strategy is discussed. An overview of the migration algorithm used to balance the load among hosts is presented in Section 6.4. Section 6.5 discusses the experimental procedures and parameters used to evaluate VM allocation and compares the results with state of the art algorithms in this field. Finally, Section 6.6 summarizes the main results and conclusions arising from this phase of the proposed model.

6.1 Overview

As discussed in Section 2.5, the virtual machine allocation problem is a crucial research issue in cloud computing. VM allocation involves finding an appropriate means of mapping VMs to hosts so that cloud resources are utilized efficiently to reduce power consumption and increase profits. Optimization of the current allocation of VMs requires the development of VM migration methodology, which distributes the load between hosts to satisfy the goals of balancing load and reduces power consumption by switching off unused hosts. At the same time, VM allocation algorithms aim to satisfy the QoS and prevent SLA violation.

In this thesis, a clustering mechanism was developed to divide the hosts into different classes based on the K-means and PSO algorithm. Based on the results of the clustering algorithm, the VM scheduling and VM migration processes were then conducted. A PSO based K-means algorithm was proposed by combining the globalized search ability of PSO and the fast convergence of K-means. This algorithm consists of two modules, namely PSO and K-means algorithms. In K-means, the convergence rate is quicker at finding a local optimum solution but slower at finding a global solution and thus justifies the combination of PSO and K-means by combining the advantages of both algorithms when clustering hosts in data centers (Neshat et al. (2012)).

The VM scheduling problem is an NP-hard problem and can be treated as a many-objective optimization problem (Panchal and Kapoor (2013)). The key goals of optimizing VM allocation involve simultaneously improving resource utilization, power consumption, QoS and profits. One of the main drawbacks of current research on VM migration solutions is that it only focuses on one or two major goals such as power consumption or resource utilization and ignores other objectives such as QoS performance, profits and SLA violation (see the discussion in Section 2.5.2). However, some researchers have proposed the use of a multi-objective VM allocation in cloud data centers. The aim of this study is to satisfy five objectives and, thus, a novel modified ranking strategy is applied to deal effectively with many objectives by using a modified ranking methodology. Moreover, MaOPSO not only addresses the aims of power consumption and reductions in SLA violation, it also strives to reduce the number of VM migrations and migration time to maintain QoS performance. To satisfy these objectives, five important criteria are considered, namely: power consumption by the host during the allocation of a VM, the host capacity, the host utilization, the data transfer time and the data transfer cost. In most research studies, only two dimensions of host utilization are considered, namely CPU capacity and memory size. In our research, other factors are taken into consideration when computing host utilization such as the network bandwidth and the storage size. This is because there are several applications that require large storage size and the distribution of data centers means that network bandwidth is an additional concern.

The proposed model used for allocating VMs to available hosts involves several processes. Specifically, the VM allocation methodology can therefore be divided into

three steps, which run in each data center. These steps are summarized in Algorithm 6.1. The first function is carried out by the Load Balancer Agent, which is responsible for clustering available hosts based on their utilization and capacity by using PSO and K-means algorithms (as described in Section 3.3). The clustering results produce four distinct lists of hosts namely high-loaded, over-loaded, under-loaded and unloaded (see Algorithm 6.2) (Line 2). The VMs are then scheduled over unloaded hosts based on a many-objective PSO algorithm that satisfies the required VM specification and reduces migration time as shown in Algorithm 6.5 (Line 3). The migrating process is conducted to move all VMs from unloaded hosts, and some VMs from high-loaded and over-loaded hosts (Line 4).

Algorithm 6.1: VM Allocation Algorithm.

Input: VMs, Hosts lists
Output:
Start Procedure VMAllocation (VMs, Hosts)
1. Initialize unloaded, over-loaded, under-loaded, high-loaded lists
2. Clustering (unloaded, over-loaded, under-loaded, high-loaded, c)
3. VMScheduling (VMs, unloaded, over-loaded)
4. VMmigration (unloaded, over-loaded, under-loaded, high-loaded)

End procedure

In the following sections, each step involved in allocating VMs will be described in detail.

6.2 Clustering Hosts Based on PSO and K-means

The novelty of the VM allocation strategy in this model is that it begins by clustering the hosts in each data center into four classes, which determine the status of the hosts based on their utilization. The main goals of applying clustering in the VM allocation are to automate the process of detecting the states of hosts rather than using static threshold points. In addition, using clustering techniques provides a more accurate and dynamic method for detecting the hosts' status, reflecting the current load of the data center. Once the clustering process is finished, the many-objective PSO algorithm assigns a VM to the unloaded hosts, a process that will be discussed in the next section.

In this thesis, a combination algorithm based on K-means and PSO clustering algorithms was therefore used to cluster hosts into four distinct groupings. As discussed in Section 2.2.5, the K-means algorithm is easy to implement and offers high processing performance. In addition, its computations are low in complexity and it efficiently deals with the collection of large amounts of data.

According to (Neshat et al. (2012)) one of the weaknesses of K-means is that it does not handle noisy data, which means that if K-means alone is applied in VM allocation it may cause imbalances among the hosts. The K-means algorithm also takes more iterations than other algorithms to initialize cluster centroids. Given this, (Baswade and Nalwade (2013)) demonstrated that if the centroid point is initially taken by a modification in the K-means algorithm, rather than random selection, it could increase performance, accuracy and reduce the number of iterations in the algorithm. The PSO is less sensitive to initial conditions due to its sub-optimal population-based nature, and is therefore more likely to find a near optimal solution. Thus, the proposed method for improving K-means using a PSO algorithm is based on applying PSO to find the initial centroids of the clusters, after which the K-means is used to enhance the results of the PSO.

An improved PSO-based K-means algorithm was developed by (Zheng and Jia (2011)) to overcome the problem of local optima in K-means clustering. Naik et al. (2012) then proposed a hybrid K-means and PSO (KPSO) clustering algorithm to obtain optimal centers for cluster analysis. In the proposed clustering model, unlike previous research, a PSO algorithm will be used to initialize the centroids of clusters. The K-means algorithm is used to refine the clustering results. Thus, by applying PSO and K-means with different objective functions, a strategy similar to that used by (Ahmadyfard and Modares (2008)) has been adopted.

The proposed clustered algorithm is summarized in Algorithm 6.2. It begins by initializing of the centroids (Line 1). Then the new values of centroids are computed using PSO in Algorithm 6.3 (Line 2). These values are then used as threshold limits to determine the status of the hosts (unloaded, over-loaded, under-loaded, and high-loaded). The hosts are assigned to clusters by comparing their utilization with the values of centroids, as shown in Lines 3-14.

Algorithm 6.2: Clustering Hosts Algorithm.
Inputs: unloaded, over-loaded, under-loaded, high-loaded**, c** *// c is the number of clusters*
Outputs: Clusters of Hosts
Procedure Clustering (unloaded, over-loaded, under-loaded, high-loaded, c**)**
1. Initialize centroid list with size of c
2. centroid =KPSOClustering (Hosts, c) *// using Algorithm 6.3*
3. **For** all host in Hosts
4. **If** host utilization < centroid [0] **then**
5. Add host into under-loaded list
6. **Else if** centroid [0] ≤ host utilization < centroid [1] **then**
7. Add host into un-loaded list
8. **Else if** centroid [1] ≤ host utilization < centroid [2] **then**
10. Add host into over-loaded list
11. **Else if** centroid [2] ≤ host utilization **then**
12. Add host into high-loaded list
13. **End if**
14. **End For**

End Procedure

The process of clustering hosts based on PSO is summarized in Algorithm 6.3. It begins by computing the utilization of hosts based on four attributes (CPU processing, memory size, storage size, and network bandwidth), as described in Equation 3.9 (Line 1). The host capacity is calculated as shown in Line 2. The PSOClustering module is initially conducted using the utilization of hosts as data points and the number of clusters to find each clusters' centroid (Line 3). The centroid values are then used in the K-means module as inputs to refine the centroids and generate the final clustering solution (Line 4). The K-means utilizes the centroid results from the PSO, following which the distance for all hosts is computed and the hosts are reassigned to the new clusters as shown in Equation 6.1, until there is no change in the distance results (Lines 5-9). Then the centroids are updated with the new cluster points as shown in Line 10 by using Equation 6.2, which computes the point's values in each cluster to find the new centroid. Finally, the centroid values are sorted in ascending order so that they can be used in the VM migration and monitoring as thresholds to determine the status of hosts (Line 11).

$$\boldsymbol{d}\,(\boldsymbol{xi}, \boldsymbol{cj}) = \left| \boldsymbol{xi} - \boldsymbol{cj} \right| \;,\; \forall\, \boldsymbol{xi} \in \boldsymbol{cluster\ j} \tag{6.1}$$

where:
d (xi, cj) is the distance between point x and centroid of cluster j
xi is the point x in the space
cj is the centroid of cluster j

$$\boldsymbol{cj} = \sum_{i \in j} \boldsymbol{xi}\ / \boldsymbol{n} \tag{6.2}$$

where
n is the number of points in the cluster j
xi is the point x in the space
cj is the centroid of cluster j

Algorithm 6.3: KPSO Clustering Algorithm.

Inputs: List of Hosts, c *// c is the number of clusters*
Outputs: Centroids of Clusters
Procedure KPSOClustering (Hosts, c)
1. U= Calculate utilization(Hosts) *// using Equation 3.9*
2. N=HostCapacity (Hosts) *// using Equation 6.4*
3. centroid=PSOClustering (U, n, c) *// using Algorithm 6.4 to create clusters by PSO*
4. **Repeat**
5. **For** all n clusters.
6. For all Hosts
7. Compute the distance of host from cluster centroids *// using Equation 6.1*
8. Assign host to the cluster that have closer centroid
9. **End for**
10. Compute the new centroid of all cluster *// using Equation 6.2.*
11. **End for**
12. **Repeat until** the centroid distance does not change.
13. Sort the clusters centroids *// sort the centroid in ascending order*
14. **Return** the clusters centroids

End

Specifically, the steps taken by the clustering algorithm based on PSO are shown in Algorithm 6.4. PSO deals with the clustering problem like any other problem by defining particles with the centroids of the clusters as discussed in Section 2.2.4. It begins with the initialization of the particles with the centroid of clusters, whereby the number of dimensions in the particles forms the number of the cluster (Line 1). For all particles, the distance of all hosts to all clusters is then computed as shown in Line 6. Each host is assigned to the closest cluster, which therefore involves a short distance as shown in Line 7. Following this, the fitness function is computed according to the new distribution of the hosts based on Equation 6.3 (Line 8). The new centroids are then computed after the velocity and position have been updated, as shown in Lines 11 and 12. Finally, the gbest is returned which includes the centroid values of the cluster (Line 15).

The fitness function of the PSO algorithm is computed according to the utilization and capacity of the hosts as shown in Equation 6.3. The objective function aims to minimize the inter cluster distance of the utilization and capacity because the hosts are clustering based on the both utilization and the capacity. The host capacity is computed based on the available resources in the host in terms of available CPU MIPS and Memory size as shown in Equation 6.4 and the utilization is computed as shown in Equation 3.9.

$$\boldsymbol{f(x) = \sqrt{(HU(x) - AHU\,(x))^2 - (HC(x) - AHC\,(x))^2}} \qquad (6.3)$$

where:
HU is the utilization of the host x as shown in Equation 3.9
AHU is the average utilization of all hosts
HC is the capacity of the host x as shown in Equation 6.4
AHC is the average capacity of all hosts

$$\mathbf{HC(x) = (ACPU(x) - UCPU(x)) + (AM(x) - UM(x))} \quad (6.4)$$

where:
HC(x) is the capacity of the host x
ACPU(x) is the available CPU in the host x
UCPU(x) is the used amount of CPU in the host x
AM(x) is the available memory in the host x
UM(x) is the amount of memory used in the host x

Algorithm 6.4: PSO Clustering Algorithm.

Inputs: List of utilization of the hosts U, c *// c is the number of clusters*
Outputs: Centroids of Clusters
Procedure PSOClustering (U, n, c)
1. Initialize each particle with c random cluster centers.
2. For iteration = 1 to max-iterations
3. for all particles
4. for all n hosts
5. for all c clusters
6. Compute the distance of host from all cluster centroids.
7. Assign host to the cluster that have closer centroid
8. Calculate the fitness function *// using Equation 6.3*
9. End for
10. End for
11. Find the Pbest, Gbest position of each particle.
12. Update the cluster centroids according to velocity updating
13. End for
14. Iteration ++
15. Return Gbest *// return the best centroid values*
End

6.3 Virtual Machine Scheduling Algorithm

In this phase, the VM is mapped to the unloaded hosts using a many-objective PSO (MaOPSO) VM scheduling algorithm, which is based on five objectives. The new aspect contained in the proposed model is that VMs are allocated to idle, under-loaded, or over-loaded hosts only to save power. This algorithm runs in each data center and selects the most appropriate host to run the VM in order to increase utilization and profits, whilst at the same time reducing power consumption and waiting time.

After determining the state of hosts using clustering algorithms (as shown in Algorithm 6.2), the scheduling of VMs over unloaded hosts is conducted. The algorithm starts mapping VMs to unloaded hosts based on the many-objective PSO. After the VM is assigned to the selected host, the unloaded host's list is updated and re-sorted after each mapping. Finally, the algorithm returns the mapping matrix, which contains the best host for each VM. Through modification of the MaOPSO algorithm, the unloaded hosts are ranked according to five objectives.

Specifically, Algorithm 6.5 summarizes the main steps involved in scheduling VMs inside the data center. The algorithm starts by checking the unloaded host's and over-loaded lists, if they contain hosts then the list and the VMs are sent to the MaOPSO for scheduling as shown in Line 3. If the unloaded list is empty, the algorithm then searches in the over-loaded list (Lines 4,5). If the over-loaded list is empty, the allocation will be carried out in one of the hosts in the sleep mode (Line 7). Finally, the selected host is then updated in term of utilization after allocation using Algorithm 6.7 (Line 10).

Algorithm 6.5: VM Scheduling Algorithm.

Input: List of VMs and List of unloaded, over-loaded
Output: the mapping results of VMs and Hosts
Procedure VMScheduling (VMs, unloaded, over-loaded)
1. **For** all VMs
2. **If** unloaded is not empty **then**
3. Host = MaOPSO_VMScheduling (VM, unloaded)
4. **Else if** over-loaded is not empty **then**
5. Host = MaOPSO_VMScheduling (VM, over-loaded)
6. **Else**
7. Assign VM to the first host in the sleep mode list // list already sorted
8. **End if**
9. UpdateUtilization (Host ,VM) *//using Algorithm 6.7*
10. **End for**
End Procedure

The VM scheduling problem formulation is now considering. The algorithm for allocating VM is based on input values with specific constraints to satisfy the objectives. The specification of these inputs and outputs will now be presented along with details of the objectives and their constraints. The input data will be considered first. There are several hosts with different specifications in each data center. In addition, each host can run one or more virtual machines based on the cores in each host. In the proposed algorithm, the VMs are allocated to unloaded, over-loaded and idle hosts.

Second, the output data is considered. The output of the VM scheduling algorithm is the placement matrix, which incorporates the mapping of hosts for each VM. A two-dimensional encoding scheme is used for solving the VM scheduling problem. The first dimension of a particle is an n vector where n is the number of unloaded hosts, while the second dimension represents the set of m VMs to be mapped to these hosts. Next, the constraints are considered. The VM placement is conducted according to the following conditions:

- The VM is running on one host at a time. Supposing that h matrix is used to represent the hosts in the data center with m × n elements where m is the number of VMs and n is the number of hosts, and then the total number of each row should be 1, $\sum_{i=1}^{n} h_{ij} \leq 1 \; \forall j \; \in m \, , where \; h_{ij} \; is \; a \; binary \; value \; 0 \; or 1$.
- The host should be sufficient for all VMs allocated to it such that $\sum_{1=1}^{m} VCPU_i \; \leq HCPU_j, \sum_{1=1}^{m} VMemory_i \; \leq HMemory_j, \sum_{1=1}^{m} VSt_i \; \leq HSt_j$

  where:
  $VCPU_i$ is the CPU processing speed of VM i
  $HCPU_j$ is the CPU processing speed of Host j
  $VMemory_i$ is the Memory size of VM i
  $HMemory_j$ is the Memory size of Host j
  VSt_i is the Storage size of VM i
  HSt_j is the Storage size of Host j

Finally, the objective functions are considered. The VM scheduling algorithm involves five objectives; the details of each of these are described as follows:

- **Power Consumption of the host after allocation (PH):** the power consumption by VMs and the host inside the data center is mainly computed in terms of CPU utilization of VMs and PMs (in MIPS) based on the power of active hosts and idle hosts. According to (Beloglazov et al., 2012), the power that has been consumed by the host in ideal condition is the 70% of their total power consumed by the host when it is 100% utilized.

$$\mathbf{PH} \; = \; \mathbf{Pidle} \; + \; \mathbf{U} \; * \; (\mathbf{Pmax} - \mathbf{Pidle}) \qquad (6.5)$$

  where:
  PH is the total power consumed by a host
  Pidle is the power consumed by idle hosts (70% Pmax (Beloglazov et al., 2012))
  Pmax is the maximum power consumed by a host, (suppose 250 W)
  U is the CPU utilization when the host is active

2. **Host Capacity (HC):** is computed as shown in Equation 6.4.

3. **Host Utilization (HU):** the utilization of the host in terms of CPU, memory, storage, and bandwidth, unlike previous work (see Section 2.5.1) which includes only CPU and memory utilization. This objective is computed as shown in Equation 3.9.

4. **Data Transfer Time (DTT):** the time taken to transfer the data files; this is based on the bandwidth of the host as shown in Equation 5.3.

5. **Data Transfer Cost (DTC):** the cost of transferring the data files to the host; this is based on the cost of bandwidth as shown in Equation 5.4.

Based on these objectives, the fitness function is computed using the modified ranking algorithm discussed in Chapter 5. Thus, the fitness function is computed by determining the rank for each objective after which the modified ranking is applied. The minimum rank is calculated as shown in Equation 6.6 and the sum ranking as shown in Equation 6.7. Finally, the weighted sum of the minimum ranking and the sum ranking is computed as shown in Equation 6.8.

$$\mathbf{F1(x)} = \mathbf{min}\,(\mathbf{rank(PH)},\ \mathbf{rank(HC)},\ \mathbf{rank(HU)},\ \mathbf{rank(DTT)},\ \mathbf{rank(DTC)}) \tag{6.6}$$

$$\mathbf{F2(x)} = \mathbf{rank(PH)} + \mathbf{rank(HC)} + \mathbf{rank(HU)} + \mathbf{rank(DTT)} + \mathbf{rank(DTC)} \tag{6.7}$$

$$\mathbf{minF(x)} = \mathbf{0.5} \times \boldsymbol{F1(x)} + \mathbf{0.5} \times \boldsymbol{F2(x)} \tag{6.8}$$

Algorithm 6.6 outlines the steps involved in scheduling VM over hosts using PSO. It begins by setting the number of elements in the particles with the number of hosts (Line 1). It then initializes the particles randomly with hosts (Line 2). For each particle, it calculates the fitness function using Equation 6.8 (Line 4). The values of pbest and gbest are updated (Lines 5-10), following which the velocity and the position for each particle are updated as shown in Lines 13 and 14. These steps are repeated until the iteration maximum is reached, after which the gbest is returned which includes the host number (Line 17).

Algorithm 6.6: MaOPSO VM Scheduling Algorithm.
Input: VM and List of Hosts in the data center
Output: the best host for VM
Procedure MaOPSO_VMScheduling (VM, Hosts)
1. Set elements in the particles = number of Hosts, t=1, tmax=10
2. **Do**
3. **for** each particle
4. Calculate solution fitness value *// using Equation 6.8*
5. **If** the fitness value is better than Pbest
6. Set Pbest = current fitness value
7. **End If**
8. **If** Pbest is better than Gbest
9. Set Gbest = Pbest
10. **End If**
11. **End for**
12. **for** each particle
13. Update particle Velocity *// use Equation 2.4*
14. Use Velocity to update particle Position *// use Equation 2.5*
15. **End for**
16. **while** (t < tmax)
17. Return Gbest *// return the best host for VM*
End Procedure

After VM scheduling is finished for each VM, the utilization of the selected host and the host cluster's lists are updated as shown in Algorithm 6.7. The utilization is computed according to the number of VMs in the host using Equation 3.8. The algorithm starts by setting the utilization as zero. Then if the host does not contain VM then only the utilization of the allocated VM is computed (Lines 1-3). Otherwise, the utilization of the available VMs is computed as shown in Lines 6-8. The utilization of the host is updated with the new value and the host's cluster will update (Lines 10 and 11).

Algorithm 6.7: Updating Utilization after VM Scheduling.
Input: Host, VM
Output: update the utilization of Host
Procedure UpdateUtiliazation (Host, VM)
1. Set utilization = 0
2. **If** Host contains no VM **then**
3. Set utilization = utilization of VM *// using Equation 3.8*
4. **Else**
5. **for** each v in the Host **do**
6. utilization = utilization + utilization of v
7. **End for**
8. **End else**
9. Set Host utilization with utilization
11. UpdateHostcluster (Host)
End Procedure

6.4 Virtual Machine Migration Algorithm

VM migration in cloud computing is one of the techniques that has recently drawn researchers' attention and is now an active field of research (see discussion in Section 2.5.2). Power consumption can be reduced considerably because an inactive host or a host in sleep mode consumes minimal energy (Pietri and Sakellariou (2016)). Reducing power consumption, which is one of the benefits of VM migration, can be achieved by switching idle resources to inactive mode when the load is low, and then turning them on again when the load is high (Wood, 2007). However, to achieve this outcome, VMs need to be migrated VMs from one host to the other, which may have a negative effect upon performance or take a long time, which may then lead to an SLA violation. Bad migration decisions may therefore waste resources and take time, increasing the waiting time and decreasing the performance in terms of throughput. The migration algorithm must therefore be designed in a way that not only reduces power consumption, but also satisfies QoS performance to prevent SLA violation especially in tasks with limited deadlines, as is the case in this research. In the VM migration method, selecting which VM to migrate to is another challenging task for which researchers have proposed several different solutions (see discussion in Section 2.5.2.2).

There are several different approaches to applying VM migration. For example, one way is to migrate all the VMs in the over-loaded machine to the under-loaded machine whose residual capacity is big enough to hold them (Lin et al. (2011)). In research conducted by Beloglazov and Buyya (2010), the under-loaded machines were migrated to free the resources and then switched off to save power. Such migration can result in performance degradation of the resources if it is applied without taking into consideration system load and time as migration is often time consuming. Therefore, a continuous monitoring scheme can be applied to minimize the number of VM migrations and ensure optimal performance.

In the proposed model, based on the results of clustering (Section 2.5.2.1), the load balancer Agent responsible for clustering the hosts checks the lists and if there are hosts in the high-loaded and under-loaded lists, it calls the VM Manager Agent to start the migration process. The under-loaded hosts are then placed in sleep mode to save power. VMs from the hosts that are over-loaded are migrated to other hosts to reduce the waiting time of the tasks. In addition, all VMs from the under-loaded hosts are migrated

to other unloaded hosts and the corresponding hosts are then switched off. The aim is to balance the load to prevent SLA violations due to long migration time in cases if migrating many VMs.

The migration algorithm in this model are conducted in the level of hosts by migration VMs from hosts to another inside data center. This make limitation of applying migration process and to overcome this limitation, the network communication between data center is should developed by improving the topology of VMs network to simplify the communication among VMs which can be improved to developed the proposed model in future work.

In more detail, Algorithm 6.8 outlines the steps undertake in VM migration. It begins by checking the high-loaded list if it is not empty it calls the MigrationLoaded procedure to transfer the VMs from these hosts to unloaded hosts (Lines 1 and 2). It then checks the over-loaded list to transfer VMs (Lines 3 and 4). Finally, VMs are migrated from the under-loaded list by calling the MigrationUnderloaded procedure (Lines 5 and 6).

Algorithm 6.8: VM Migration Algorithm.

Inputs: high-loaded, over-loaded, unloaded, under-loaded lists
Outputs: migration of VMs
Procedure VMmigration (high-loaded, over-loaded, unloaded, under-loaded)
1. **If** high-loaded is not empty **then**
2. MigrationLoaded (high-loaded)
3. **If** over-loaded is not empty then
4. MigrationLoaded (over-loaded)
5. **If** under-loaded is not empty then
6. MigrationUnderloaded (under-loaded)
7. **End if**

End Procedure

The procedures involved in VM migration are similar but the difference is that from under-loaded lists all VMs are migrated to switch off the hosts whilst in the over-loaded and high-loaded lists one VM is transferred to avoid high-loaded hosts. In Algorithm 6.9, the migration steps start by setting the flag value to false then, for all hosts, it tries to transfer the VMs to unloaded or under loaded hosts (Lines 4-8). If all VMs transfer from one host, it is then switched off and removed from under-loaded hosts (Lines 9-11).

Algorithm 6.9: Migration from under-loaded host.

```
Procedure MigrationUnderloaded (under-loaded)
1.  Flag=false;
2.  For each host in under-loaded list
3.      For each vm in the VM list of the host
4.      If (Migratedto (vm, unloaded)) or (Migratedto (vm, under-loaded)
5.        Flag = true
6.        Continue for all vm
7.      Else go to next host
8.      End if
9.      If flag=true
10.       add host to sleep list
11.       Remove host from under-loaded list
12.     End if
13.  End for
End Procedure
```

If VM migration takes place from high-loaded and over-loaded lists, then Algorithm 6.10 is applied. The algorithm starts by selecting one VM from the host for transfer based on the minimum migration time, which is computed in Equation 6.9 (Line 2). It is then migrated to the unloaded list (Line 3). The utilization of the host and the cluster are then updated (Lines 4-5) and the host is removed from the high-loaded list. The same process for over-loaded lists is followed in Algorithm 6.11 where, instead of high-loaded lists, the over-loaded lists are used.

Algorithm 6.10: Migration from high-loaded host.

```
Procedure MigrationLoaded (high-loaded)
1.  For each host in high-loaded list
2.          vm = selectvm (host)
3.      If (Migratedto (vm, unloaded))
4.        Computehost utilization(host)
5.        Updatehostcluster(host)
6.        Remove host from high-loaded list
7.      Else
8.          go to next host
9.      End if
10.  End for

End Procedure
```

The minimum migration time approach is used to select the VM for migration. This approach depends on selecting the VM with the least memory to be migrated. This because a VM with less memory can be migrated faster depending on the bandwidth of the host. The migration time for each VM is computed by dividing the memory size of the VM by the bandwidth of the host as shown in Equation 6.9.

$$\mathbf{MT} = \mathbf{RAM}(\mathbf{v})/\mathbf{BW}(\mathbf{i}) \quad (6.9)$$

where:
MT is the migration time in seconds
Ram (v) is the size of VM v in MB
BW (i) is the bandwidth of the hosts i

Algorithm 6.11: Migration from high-loaded host.

```
Procedure MigrationLoaded (over-loaded)
1.  For each host in high-loaded list
2.      vm = selectvm (host)
3.      If (Migratedto (vm, unloaded))
4.          Computehost utilization(host)
5.          Updatehostcluster(host)
6.          Remove host from over-loaded list
7.      Else
8.          go to next host
9.      End if

End Procedure
```

The processes involved in VM migration from host are listed in Algorithm 6.12. It starts by initializing the selected host with a null value (Line 1). The MaOPSO algorithm is then applied to select the host for VM (Line 3). If the process of allocation succeeds, the utilization of the selected host and the cluster are updated (Lines 7-9).

Algorithm 6.12: Select Host for migration VM.

```
Procedure migratedto (VM, hosts)
1.  selectedhost=null
2.  For each host in hosts
3.      selectedHost =MaOPSO_VMScheduling (VM, hosts)
4.      If selectedHost=null
5.          return false
6.      Else
7.          update selectedhost utilization
8.          Updatehostcluster(selectedhost)
9.          return true
10.     End if
11. End for
End procedure
```

6.5 Experimental Evaluation

To evaluate the proposed model of VM allocation algorithms the CloudSim simulator is used. Several classes such as VM_migration, VMScheduling, MaOPSO, modified Ranking and Clustering classes are added to the simulator to execute our model. In

addition, some classes of CloudSim are modified such as Datacenter, Broker, VM and cloudlet classes.

The specification and characteristics of resources and tasks presented in Section 3.4.1 are used. In the following sections, the experimental methodology and the results of the evaluation of our proposed model are discussed.

6.5.1 Experimental Methodology

In this section, the methodology for evaluating the proposed model will be discussed. To evaluate the strategy of VM allocation developed in this research, it will be compared with single and double thresholds, and the First Fit algorithm (see Section 2.5.2.1) to demonstrate the effectiveness of our approach and determine whether there is a difference in the results. The First Fit is based on choosing the first host that can fit the VM without considering any other objectives. In the next section, the results of these algorithms are compared with the proposed algorithm.

Each experiment was run ten times and the average score across all experiments was then calculated with 95% confidence intervals. After running the simulation, the following parameters are measured and used as indicators to test the effectiveness of the proposed algorithm. To evaluate the performance of the MaOPSO VM allocation algorithm, the following factors were computed:

- **Average Waiting Time:** is defined in Equation 3.1**.**
- **Average Completed Time:** is defined in Equation 3.2.
- **Throughput:** is defined in Equation 3.3.
- **Average Resource Utilization**: is defined in Equation 3.9**.**
- **Profit:** is defined in Equation 3.12.
- **Power Consumption:** is defined in Equation 3.14.
- **SLA Violation:** is defined in Equation 3.18**.**
- **Imbalance Factor:** is defined in Equation 3.20.

6.4.2 Experimental Results

In this section, the experimental results will be presented. The model was evaluated according to several factors that reflect the research goals. These factors include completion time, waiting time, resource utilization, throughput, profit, power consumption, migration time, and the number of migrations.

First, the completion time and waiting time are considered. The VM allocation completion time was tested by determining the submission and finishing time of the task. The test was repeated using different numbers of tasks in each experiment (see Table 6.1). As can be seen in Figure 6.1, the completion time for the proposed algorithm is less than for other algorithms. This is because the selection of hosts to allocate VMs based on MaOPSO, which depends on five objectives, was improved. The enhancement in the model is based on selecting the host from an unloaded cluster, which leads to a reduction in allocation and migration time. The proposed model reduces the completion time by about 15% compared to other algorithms.

# Task	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	88	100	100.2	102
100	88	102	101.8	104
150	90	97	102.6	104
200	93	100	100	105
250	97	99.5	101.5	107
300	98	101.3	103.5	107.3
350	98	100.5	104	108
400	99	102.5	105	109
500	101.3	105.4	106	110

Table 6.1: Average Completion Time Results (in seconds).

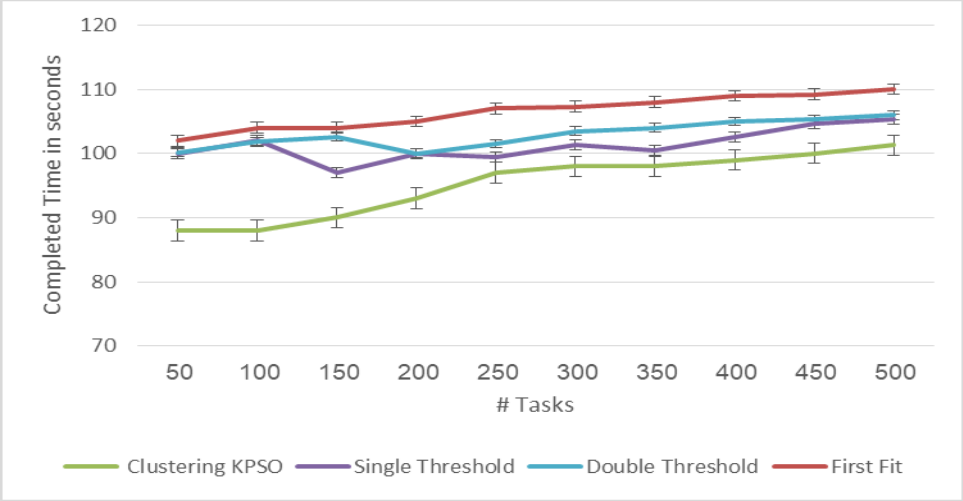

Figure 6.1: Average Completion Time (data shown with 95% confidence intervals).

The results of the waiting time experiment are presented in Table 6.2 and Figure 6.2. These show that an increase in the number of tasks causes the waiting time to increase. Waiting time includes the mapping time, the time spent queuing the tasks and migration time prior to the start of execution. This model gives the shortest waiting time compared to other algorithms because it searches for an optimal allocation of VM. The First Fit algorithm gives the longest waiting time because it allocates VM to the first fitting host.

# Tasks	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	5.063	15.002	10.054	15.114
100	10.028	15.606	12.854	20.042
150	15.403	20.405	18.109	25.062
200	20.088	25.178	21.957	30.919
250	25.388	33.005	28.604	35.118
300	30.991	37.084	35.9	40.916
350	35.521	45.203	40.902	51.505
400	40.578	53.52	47.825	55.969
500	50.934	63.701	57.57	67.231

Table 6.2: Average Waiting Time Results (in seconds).

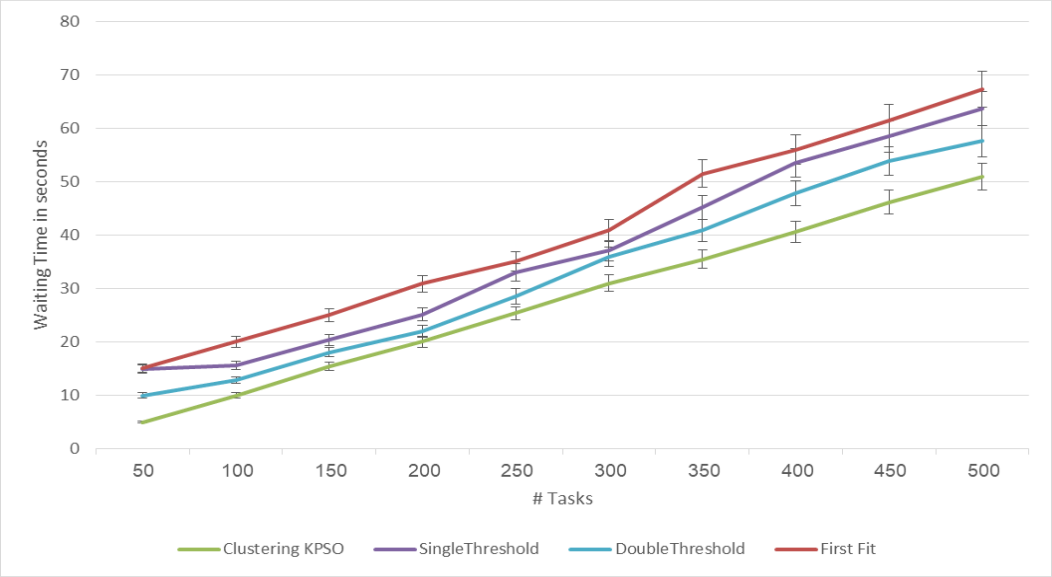

Figure 6.2: Average Waiting Time (data shown with 95% confidence intervals).

Second, the throughput factor is considered. It is used to measure the performance of the proposed model regarding the execution many tasks in a small amount of time. The results in Table 6.3 show that, when there is a large number of tasks, the model offers good performance regarding throughput. This is because our model tries to select the best host for each VM, which leads to an increase in the number of tasks executed in a short space of time. By comparing the results in Figure 6.3, it can be inferred that the throughput of our model is the highest, whereas the throughput of First Fit is the smallest. Thus, our model maps VMs to hosts in a way that requires less utilization and high CPU processing, thereby executing more tasks in a shorter space of time.

#Tasks	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	60.3	54.8	55.19	48.9
100	69	62.15	66.21	60.7
150	70.4	65.85	68	61.05
200	73.5	68.75	70.3	62.2
250	76.8	70.75	72.14	65.45
300	79.1	71.88	73.2	66.5
350	81.5	72.88	74.5	68.35
400	83	73.59	75	70.3
450	85.6	74.4	76.3	72.2
500	87.2	75.35	77	73.39

Table 6.3: Throughput Results.

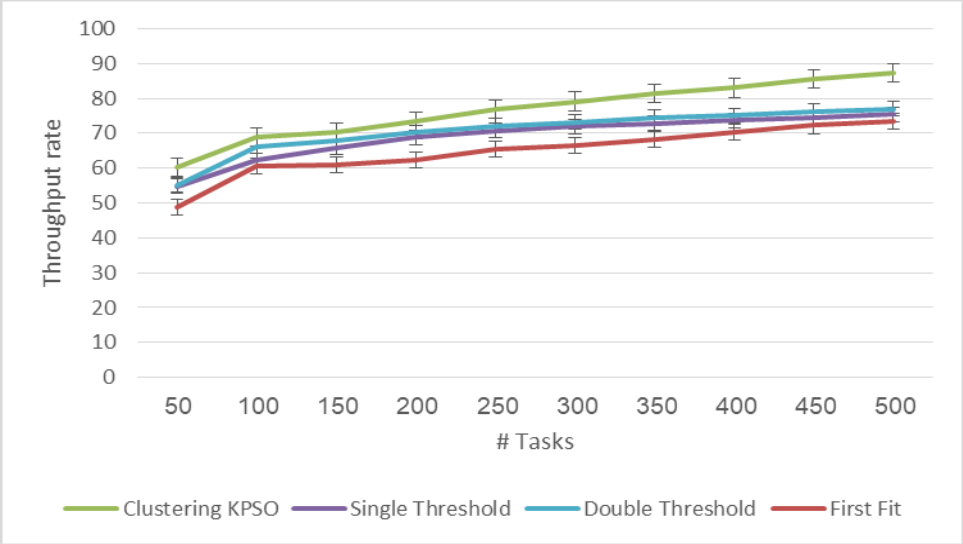

Figure 6.3: Throughput Results (data shown with 95% confidence intervals).

Third, the average resource utilization is considered. When the resource utilization in our system was evaluated and compared to the results of other algorithms, the proposed model gave the highest utilization. This is because the methodology for allocating hosts to VMs is taking the host utilization as a factor of evaluating hosts. Moreover, the VM migration algorithm aims to improve utilization by moving VMs from under-loaded hosts to unloaded ones to maximize utilization by hosts. Specifically, the resource utilization increased in line with an increase in the number of tasks (as shown in Table 6.4). Thus, the new algorithm improves utilization by up to 10% compared to single and double threshold algorithms and up to 20% compared to the First Fit algorithm.

# Tasks	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	0.22	0.2	0.21	0.15
100	0.2	0.2	0.22	0.18
150	0.35	0.31	0.29	0.25
200	0.55	0.39	0.42	0.34
250	0.6	0.41	0.46	0.39
300	0.61	0.47	0.52	0.41
350	0.65	0.49	0.51	0.45
400	0.72	0.55	0.63	0.51
450	0.75	0.61	0.62	0.57
500	0.77	0.62	0.66	0.6

Table 6.4: Average Resource Utilization Results.

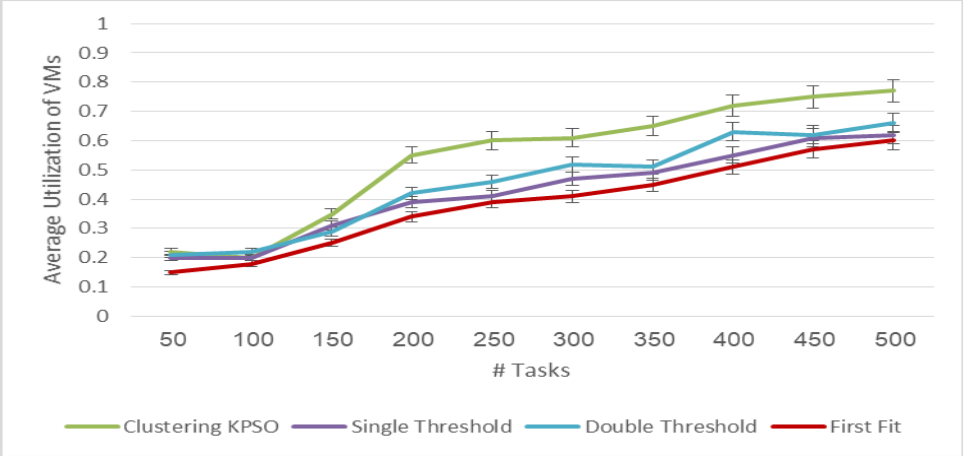

Figure 6.4: Average Resource Utilization (data shown with 95% confidence intervals).

Fourth, the profits are considered. These are computed as the difference between the cost of executing all tasks and the cost of power consumed during execution, and the penalty cost if an SLA violation occurs. Table 6.5 displays the profit results. From these, it is clear the profit in our model is the greatest. For example, running 300 tasks in our algorithm gives $38 while executing the tasks using a single threshold algorithm gives $28.6 and $26.2 when using double thresholds. This means that the profit increases by a ratio of approximately 30% compared to a single threshold algorithm and 40% compared to a First Fit. This is because our strategy improves the throughput by increasing the number of executed tasks, which indirectly improves profit. In addition, our model reduces the cost of power by moving the unused running hosts into sleep mode. The profits also benefit from a reduction in the number of failed tasks.

#Tasks	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	5.5	2.9	3.07	2.1
100	6.9	4.3	5.03	3.3
150	13	11.7	9.2	8.3
200	15.7	13.6	12.5	11.9
250	25	20.1	18.8	17.4
300	38	28.6	26.2	23
350	45	37.5	40.7	33.2
400	50	43.6	46.8	41.1
450	56	47.7	48	45
500	63	52.1	51	49.2

Table 6.5: Profit Results.

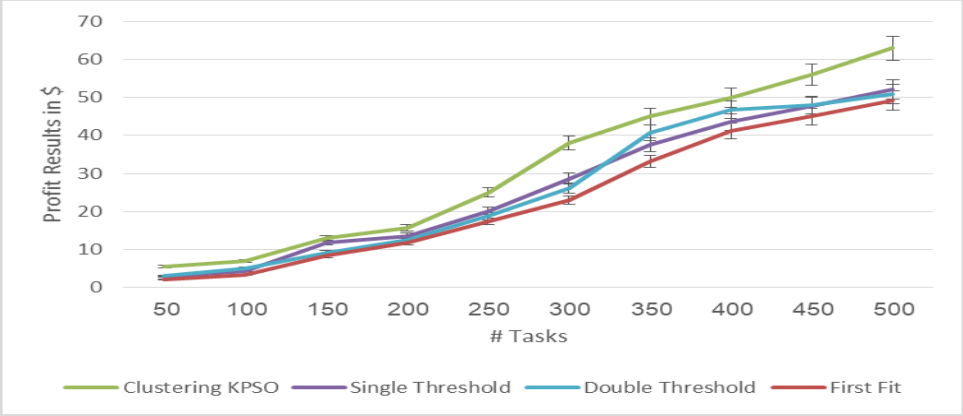

Figure 6.5: Profit Results (data shown with 95% confidence intervals).

Fifth, we consider power consumption. As shown in Table 6.6, there is a reduction in the power consumed when using the proposed algorithm compared to that consumed by others. This is because using the VM migration strategy increases the hosts' utilization and saves power by moving under-loaded hosts to sleep mode. In Figure 6.6, single and double thresholds give high ratios of power consumption compared to the proposed algorithm, as they depend on static values that do not reflect the status of the hosts' utilization for all cases. The clustering KPSO algorithm consumes the least amount of power compared to all other algorithms because it migrates VMs from under-loaded hosts to conserve the power of the host. It reduces the consumed power by 15% compared to the threshold approach and 25% compared to the First Fit algorithm.

# Tasks	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	0.05	0.09	0.07	0.1
100	0.07	0.1	0.08	0.12
150	0.08	0.12	0.09	0.14
200	0.09	0.14	0.1	0.17
250	0.1	0.16	0.12	0.19
300	0.11	0.18	0.14	0.23
350	0.12	0.21	0.16	0.25
400	0.14	0.23	0.18	0.27
450	0.15	0.27	0.23	0.32
500	0.2	0.31	0.25	0.35

Table 6.6: Power Consumption Results.

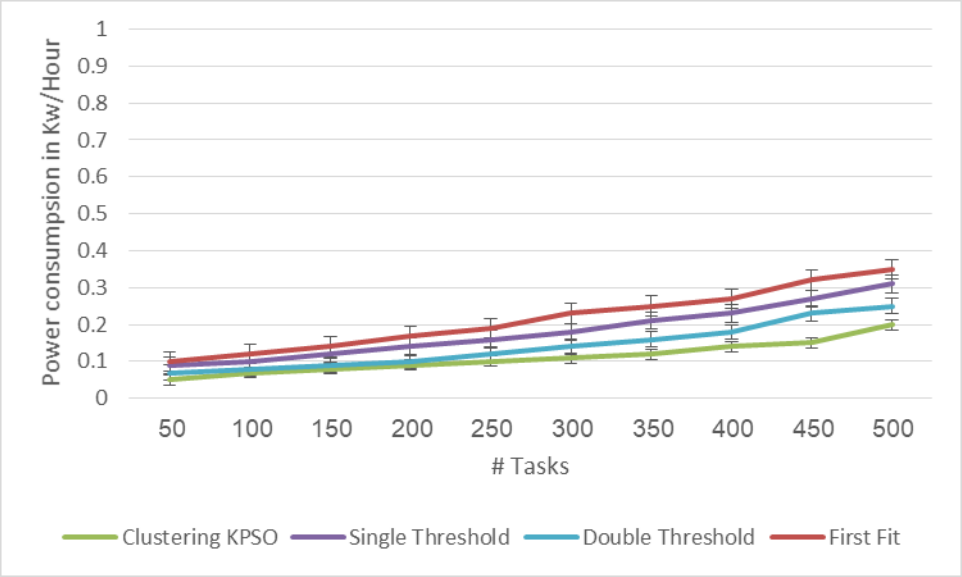

Figure 6.6: Power Consumption Results (data shown with 95% confidence intervals).

Next, the imbalance factor is considered. The imbalance factor results in Figure 6.7 indicate that the proposed algorithm balances the load better than any other algorithm. This because the load balancer was applied to balance the load and migrate VMs from over-loaded and high-loaded lists. The First Fit results yield high imbalance factors, which means that it unbalances the load between hosts because it does not consider the load balance as a factor during the allocation process.

# Tasks	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	5.3	7.01	8.7	9.09
100	5.8	7.97	9.41	10.2
150	6.6	8.57	9.57	11.5
200	7.5	9.75	10.6	12.4
250	8.01	10.09	11.29	13.6
300	9.3	10.7	12.59	15.9
350	9.8	12.31	14.7	18.3
400	11.3	13.2	15.66	20
450	12.5	14.9	17.34	23.5
500	15.3	18.2	23.87	27.5

Table 6.7: Imbalance Factor Results.

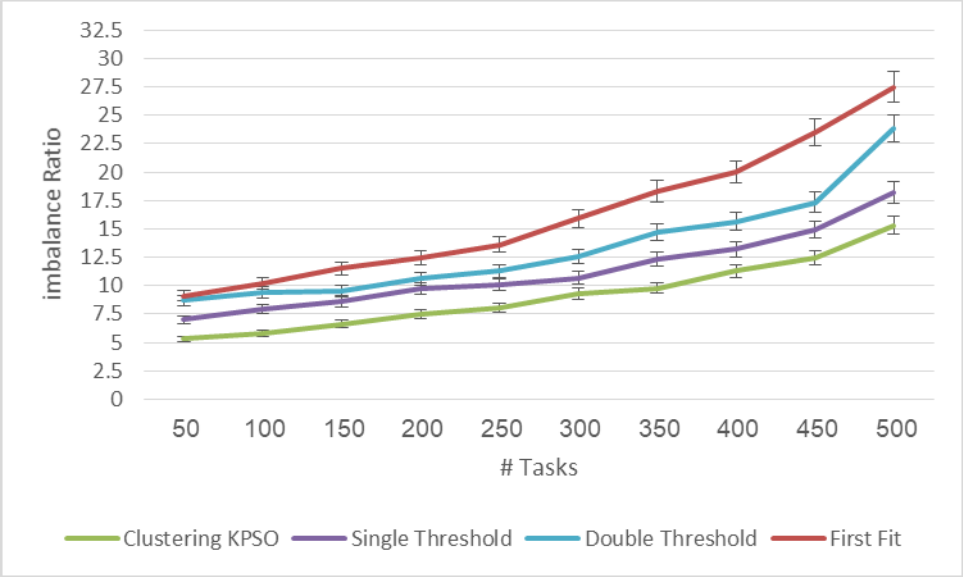

Figure 6.7: Imbalance Factor Results (data shown with 95% confidence intervals).

Finally, the SLA violation rate is considered. An SLA violation occurs if the tasks miss deadlines or if the VM is assigned a greater number of MIPS than was initially allocated, which increases the waiting time. The SLA violation comparisons in Figure 6.9 show that the proposed model decreases SLA violation by about 25% compared to the single and double thresholds and by 35% compared to the First Fit algorithm.

# Task	Clustering KPSO	Single Threshold	Double Threshold	First Fit
50	34.3	45.8	40.3	47.3
100	32.9	52.1	46.4	54.3
150	35.2	55.5	48.2	57.4
200	36.6	58.2	49.5	60.7
250	38.1	60.3	51.2	63.5
300	39.4	61.8	52.8	65.1
350	41.5	62.3	53.6	66.75
400	42.7	63.1	54	68.13
450	39.4	62.6	55.7	71.7
500	37.5	62.9	57.4	72.95

Table 6.8: SLA Violation Rate Results.

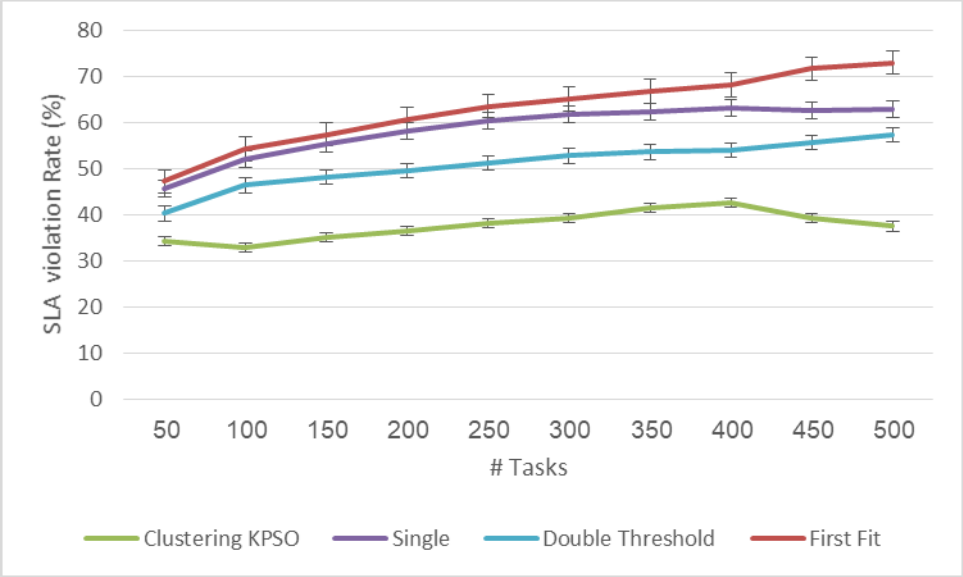

Figure 6.8: SLA Violation Rate Results (data shown with 95% confidence intervals).

The algorithm will now be evaluated in relation to clustering, where the results will be compared to PSO and K-means algorithms separately to demonstrate the effective of the proposed KPSO, which combined the two algorithms. The results were evaluated in term of clustering time, waiting time, throughput and power consumption. The results in Figures 6.9 and 6.10 show that the proposed algorithm consumes less in the way of clustering time and waiting time than the K-means and PSO algorithms because it uses PSO to initialize the cluster centroids. Consequently, the proposed model executes more tasks and therefore gives high throughput as shown in Figure 6.11.

# Task	KPSO	K-means	PSO
50	0.17	0.28	0.21
100	0.3	0.44	0.37
150	0.4	0.6	0.46
200	0.45	0.72	0.57
250	0.6	0.8	0.65
300	0.71	0.84	0.79
350	0.82	0.92	0.87
400	0.85	1.2	0.96
450	0.9	1.5	1.13
500	1.2	1.6	1.54

Table 6.9: Waiting Time Results.

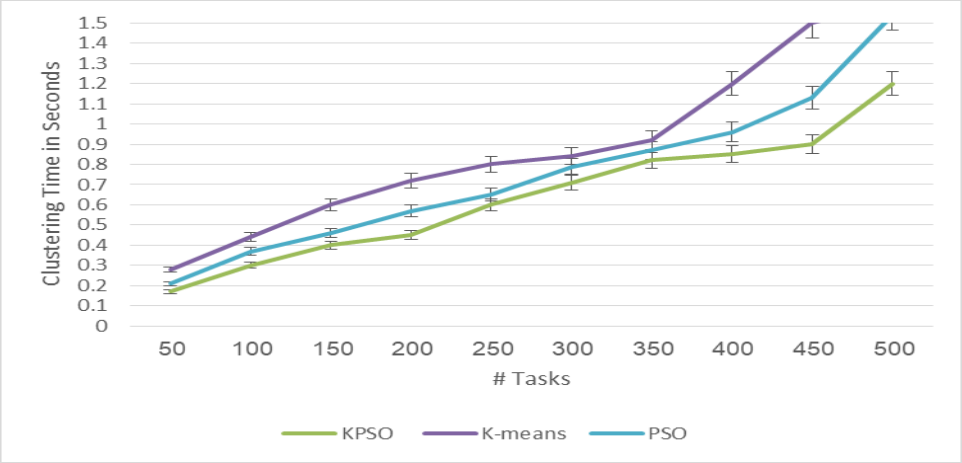

Figure 6.9: Clustering Time Results (data shown with 95% confidence intervals).

# Task	KPSO	K-means	PSO
50	0.61	0.73	0.65
100	0.73	0.85	0.78
150	0.82	0.94	0.8
200	0.95	1.5	0.98
250	1.3	1.7	1.5
300	1.6	1.93	1.76
350	1.8	2.3	2.13
400	2.2	2.67	2.46
450	2.4	2.91	2.87
500	2.6	3.4	3.07

Table 6.10: Average Waiting Time Results.

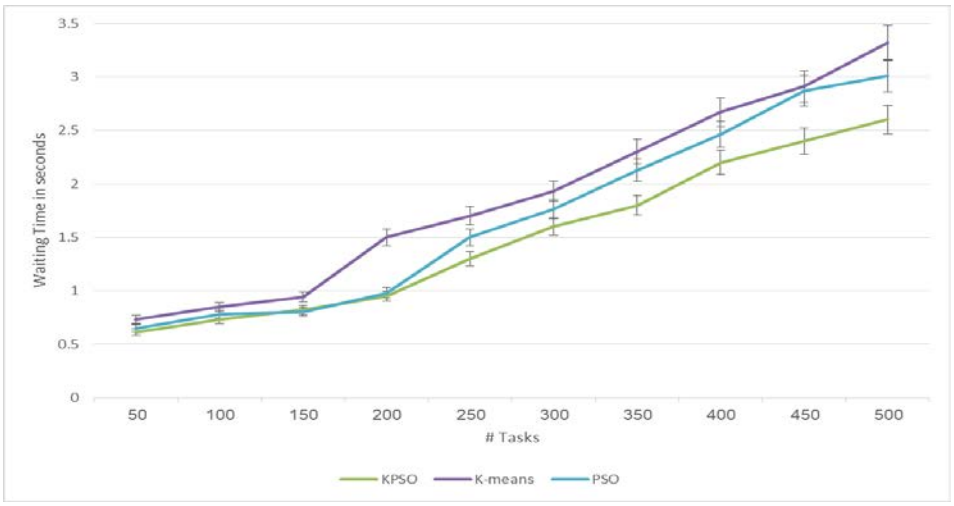

Figure 6.10: Average Waiting Time (data shown with 95% confidence intervals).

# Task	KPSO	K-means	PSO
50	60.3	40.8	52.19
100	69	42.7	57.3
150	70.4	51.5	61.2
200	73.5	56.4	65.5
250	76.8	60.5	68.43
300	79.1	63.2	70.1
350	81.5	66.81	72.35
400	83	69.3	74.7
450	85.6	70.4	77.5
500	87.2	73.4	79.2

Table 6.11: Throughput Rate Results.

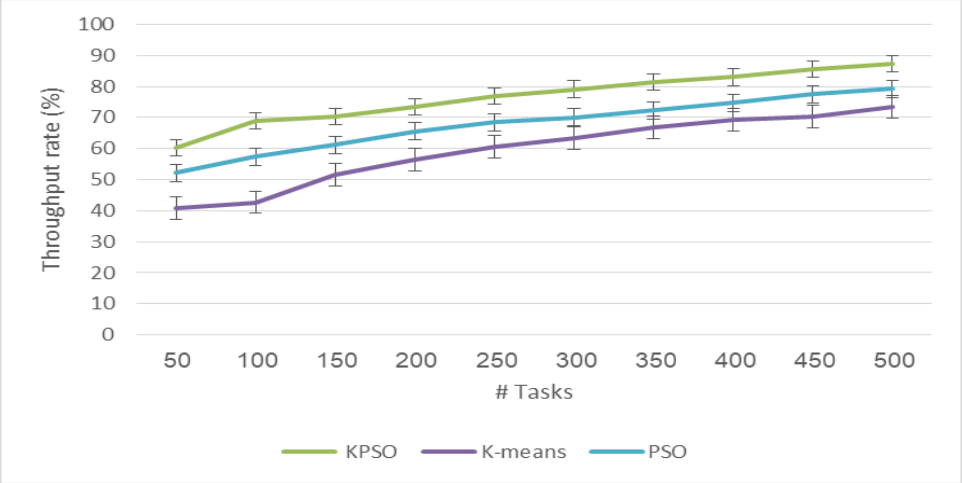

Figure 6.11: Throughput Results (data shown with 95% confidence intervals).

6.6 Summary

In this chapter, the specification of VM allocation was presented and discussed in detail. Essentially, the proposed model of VM allocation consists of four steps:

- Clustering hosts based on the host load and utilization using K-means and PSO algorithms
- Scheduling the VMs to unloaded hosts by finding the appropriate host for each VM using a MaOPSO algorithm and a modified ranking strategy
- Migrating VMs from high-loaded, over-loaded and under-loaded hosts based on the minimum migration time

- Monitoring the load of the hosts to avoid high-loaded and under-loaded hosts.

The main findings from the experimental analysis are as follows:

- The proposed algorithm provides an improvement in waiting time of up to 15%, in completion time of up to 15%, and in throughput of up to 15%.
- The utilization of the resources in our model is increased by about 20% compared to the K-means algorithm due to the effective use of VM migration.
- A noticeable enhancement in profits in our model was observed when increasing the number of tasks. This is because improving resource utilization indirectly improves profit by about 20% compared to the single threshold algorithm and by about 30% compared to the First Fit algorithm.
- Power consumption in our model is reduced by 20% compared to other algorithms because the VM migration algorithm has been improved.
- The proposed model also provides a small imbalance factor, which means that it balances loads by about 40% more effectively than other methods.
- Our model reduces SLA violation rates by about 35% compared to the First Fit algorithm and 25% with thresholds.
- The proposed algorithm reduces waiting time and clustering time compared to PSO and K-means algorithms.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

A model for allocating resources in cloud computing was proposed in this thesis, which involved improving three aspects: SLA negotiation, task scheduling and VM allocation. Particle swarm optimization was used to optimize these modules with different variants depending on the nature of the problem in each case.

For SLA negotiation, parallel PSO is used to improve the process of negotiation between consumers and multiple distributed data centers with different resources. The parallel PSO algorithm is used to quicken the process of negotiation and automate the process to reach agreement in a short amount of time with a reasonable quality solution. The proposed algorithm for SLA negotiation reduces the waiting time by about 30% compared to PSO and 20% compared to SPPSO. The throughput increases by about 20% compared to the PSO algorithm. SLA violation rates are improved by about 25% compared to PSO.

The task scheduling algorithm was improved using a many-objective PSO algorithm based on a modified ranking strategy. In so doing, this work advanced the state of the art by improving task scheduling using MaOPSO based on five objectives, and it devised a modified ranking strategy to evaluate the objective functions. The results of the modified ranking strategy were used in MaOPSO task scheduling to find an appropriate VM for each task. This improvement accelerates and simplifies the process of evaluating objectives in MaOPSO algorithms compared to using Pareto set method or weighted sum. The work was evaluated empirically by comparing it to a Genetic algorithm and ACO algorithm for optimization and two heuristic algorithms (Min-min and Max-min). Our results show that the new algorithm outperformed these algorithms

in terms of profits, performance (waiting time, execution time, and throughput), resource utilization and power consumption. Specifically, MaOPSO maximized resource utilization and minimized waiting time, demonstrating improvements of up to 40% compared to Min-min. In addition, the VM migration reduced power consumption by 25% compared to a Genetic algorithm. Profits were improved indirectly by improving the utilization, as much as 40% in some cases. The improved task scheduling aimed to reduce the completion time of the tasks, which improves both throughput and profits. Additionally, the SLA needed be satisfied to reduce any SLA violation caused by missing task deadlines. Finally, the MaOPSO task-scheduling algorithm increases throughput, reduces waiting time and shows an improvement up to 30% greater than the weighted sum objective.

Regarding VM allocation and migration, the strategy for allocating hosts based on many-objective PSO was improved, enabling it to handle many objectives simultaneously. Clustering based on PSO and K-means algorithm was applied to define the threshold limits, improve the migration process and increase the utilization of resources. Additionally, the VM migration, based on clustering results was developed to decrease power consumption. The proposed VM allocation algorithm decreases both the waiting time and completed time by 15% and increased the throughput by about 15%. Moreover, by applying the migration process, it reduced the power consumption by about 25% and reduced the imbalance factor by 40%. Regarding SLA violation rates, it reduced violations by about 35%, which improved profits by about 20%.

Thus, from these improvements, performance in terms of waiting time and throughput was enhanced by reducing the negotiation time and mapping time. Moreover, utilization of resources was increased and balanced in an efficient way by applying VM migration. At the same time, power consumption was reduced by closing the under-loaded hosts during migration process. The profit was indirectly increased due to maximization of resource utilization and increasing the overall throughput.

7.2 Future Work

Based on the analysis of the literature in Section 2.6 and the results obtained from applying our model, the following issues need to be addressed in future work.

- **Improving the Initialization of PSO with Heuristic Algorithms**

The MaOPSO used in the scheduling task can be extended to explore different options for creating an initial population, for example using a Max-min or Min-min approach, as this will influence the quality of solutions and convergence speed of the algorithm (as discussed in Section 2.2.2). During the study, it became apparent that, in future research, further improvements can be made to the initialization of the PSO algorithm, improving the fitness function, and incorporating the advantages of other clustering algorithms to improve the efficiency and performance of clustering (see discussion in Section 2.2.4). It is relatively easy to find more effective algorithms for clustering in the pattern recognition domain as this depends on the problem and the PSO algorithm can be used to optimize the clustering algorithm parameters.

- **Improving the Load Balancing at The VM Level**

In our model, we balance the load of VMs in the hosts. Many benefits can be gain if we applying balancing load at the level of VMs by distributing the tasks between VMs. Another meta-heuristic can be combined with PSO to overcome the limitations of PSO. One suggestion would be to use the Cuckoo search algorithm (CSA), which is a meta-heuristic optimization algorithm inspired by the behavior of certain cuckoo species (Yang and Deb (2009)). CSA is mostly used as a single parameter and is well known for its simplicity and ease of implementation. CSA works with an initial population that represents a set of cuckoos. The population of cuckoos in the nest will depend on the host bird. Eggs that are like those of the host have more chance to grow and become a cuckoo and other eggs will be identified and destroyed by the host bird (Yang and Deb (2009)).

Compared to the PSO algorithm, which can converge to local optima prematurely, the CSA algorithm often converges to global optimal solutions. Moreover, the CSA algorithm can effectively balance the local and global search with the help of a switching parameter.

- **Automating the Process of Defining Consumers' Requirements**

In this research, it was assumed that the consumer specifies their requirements (see discussion in Section 3.1). However, a more intelligent method is needed to determine the consumers' requirements. Moreover, an assessment of the consumers' service requirements based on different types of application can improve resource utilization, such that services delivered to the consumer as the ones they require, no more or less. For example, if consumer's request is running tasks on a resource, the tasks are mapped to the required resource. In our model and most of the developed scheduling algorithms, the requirement is determined in terms of CPU, memory, storage and bandwidth rather than determined by the consumers, which is neither accurate nor precise.

- **Scheduling Workflow Applications**

The proposed model of task scheduling in Section 5.1 can be improved to handle applications that have dependencies between tasks, enabling it to deal with workflow scheduling. This is discussed in Section 2.4.1, and incorporating such functionalities would extend our approach and allow it to be applied in the scheduling of a wider range of applications including bioinformatics applications and computational biology.

- **Fault Tolerance Factors**

The notion of resource failure can be integrated with the allocation process to enhance reliability. At present this is absent from our approach, as discussed in Section 2.3.5 Resource performance variation should also be added to the monitoring services to predict the possibilities of failure before it occurs. Incorporating these features into the proposed model will build greater robustness into the scheduling process. Detecting faults and recovering the system after failure need to be improved and optimized because it is a significant requirement in developing any resource management model.

- **Dynamic SLA**

In the scheduling process, there is currently no mechanism to detect the changes in the SLA violation and adapt the changes in the SLA as discussed in Section 2.3.1. Our model deals with a static form of SLA that is determined at the beginning of negotiation, if the SLA can be changed during the execution with specific constraints then more benefits can be gained in terms of performance and resource utilization. The SLA form can be checked and evaluated while the resource allocation is proceeding to reflect changes in system utilization and performance. This suggestion can form part of a long-term agreement because more changes can be made after the agreement is signed.

- **Communication between Data centers**

Currently model of resource allocation there is one entity that manages all data centers and no communication between data centers. This architecture can therefore be applied to a small number of data centers. However, as the number of data centers increases, the management process through a centralized manager becomes more complicated and can be a single point of failure.

To improve the architecture and reduce the communication overheads a multi-agent system can be used to facilitate communication between data centers, enabling agents to negotiate to select the best data center for each task to develop distributed manager model. In addition, improving the strategies of communication between data centers can improve the balancing load among them and increasing the performance of the system.

- **Improve the network topology of the VM**

The main limitation of applying VM migration between data centers is the network communication, as discussed in Section 2.5.2. In our model, the migration of VMs from one host to another inside the data center was developed. In future research, VM migration between data center can be developed to manage the network VM more efficiently.

One of the suggested ways to improve this is to develop virtualized VM in data centers by optimizing network topologies among virtualized VMs, as discussed in Section 6.4. This will reduce network communication traffic during the migration of VMs from one data center to another.

References

Abdel-Kader, R. F. (2010, February). Genetically improved PSO algorithm for efficient data clustering. In *Machine Learning and Computing (ICMLC), 2010 Second International Conference on* (pp. 71-75). IEEE.

Abdi, S., Motamedi, S. A., & Sharifian, S. (2014, January). Task scheduling using Modified PSO Algorithm in cloud computing environment. In *International conference on machine learning, electrical and mechanical engineering* (pp. 8-9).

Abdullah, R., & Talib, A. M. (2012, October). Towards integrating information of service level agreement and resources as a services (RaaS) for cloud computing environment. In *Open Systems (ICOS), 2012 IEEE Conference on* (pp. 1-5). IEEE.

Adamuthe, A. C., Pandharpatte, R. M., & Thampi, G. T. (2013, November). Multiobjective virtual machine placement in cloud environment. In *Cloud & Ubiquitous Computing & Emerging Technologies (CUBE), 2013 International Conference on* (pp. 8-13). IEEE.

Adrian, B., & Heryawan, L. (2015, November). Analysis of K-means algorithm for VM allocation in cloud computing. In *Data and Software Engineering (ICoDSE), 2015 International Conference on* (pp. 48-53). IEEE.

Ahmad, R. W., Gani, A., Hamid, S. H. A., Shiraz, M., Yousafzai, A., & Xia, F. (2015). A survey on virtual machine migration and server consolidation frameworks for cloud data centers. *Journal of Network and Computer Applications*, *52*, 11-25.Aissi, H., Bazgan, C., & Vanderpooten, D. (2005). Complexity of the min–max and min–max regret assignment problems. *Operations research letters*, *33*(6), 634-640.

Ahmadyfard, A., & Modares, H. (2008, August). Combining PSO and k-means to enhance data clustering. In *Telecommunications, 2008. IST 2008. International Symposium on* (pp. 688-691). IEEE.

Abulkhair, M. F., Alkayal, E. S., & Jennings, N. R. (2017, September). Automated Negotiation using Parallel Particle Swarm Optimization for Cloud Computing Applications. In *Computer and Applications (ICCA), 2017 International Conference on* (pp. 26-35). IEEE.

Al-Ayyoub, M., Jararweh, Y., Daraghmeh, M., & Althebyan, Q. (2015). Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure. *Cluster Computing*, *18*(2), 919-932.

Alkayal, E. S., Jennings, N. R., & Abulkhair, M. F. (2016, November). Efficient Task Scheduling Multi-Objective Particle Swarm Optimization in Cloud Computing. In *Local Computer Networks Workshops (LCN Workshops), 2016 IEEE 41st Conference on* (pp. 17-24). IEEE.

Alkhashai, H. M., & Omara, F. A. (2016). An Enhanced Task Scheduling Algorithm on Cloud Computing Environment. *International Journal of Grid and Distributed Computing*, *9*(7), 91-100.

An, B., Lesser, V., Irwin, D., & Zink, M. (2010, May). Automated negotiation with decommitment for dynamic resource allocation in cloud computing. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1* (pp. 981-988). International Foundation for Autonomous Agents and Multiagent Systems.

Anuradha, V. P., & Sumathi, D. (2014, February). A survey on resource allocation strategies in cloud computing. In *Information Communication and Embedded Systems (ICICES), 2014 International Conference on* (pp. 1-7). IEEE.Attea, B. A. A. (2010). A fuzzy multi-objective particle swarm optimization for effective data clustering. *Memetic Computing*, *2*(4), 305-312.

Arfeen, M. A., Pawlikowski, K., & Willig, A. (2011, July). A framework for resource allocation strategies in cloud computing environment. In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual* (pp. 261-266). IEEE.

Awad, A. I., El-Hefnawy, N. A., & Abdel_kader, H. M. (2015). Dynamic Multi-objective task scheduling in Cloud Computing based on Modified particle swarm optimization. *Advances in Computer Science: an International Journal*, *4*(5), 110-117.

Al-maamari, A., & Omara, F. A. (2015). Task scheduling using PSO algorithm in cloud computing environments. *International Journal of Grid and Distributed Computing*, *8*(5), 245-256.

Al-Olimat, H. S. (2014). *Optimizing cloudlet scheduling and wireless sensor localization using computational intelligence techniques*. The University of Toledo.

Bagul Dhanashri, R., & Toris Divya, N. (2017). Implementation of Two Level Scheduler in Cloud Computing Environment. *International Journal of Computer Applications, Volum No.3,, 166*(3).

Banerjee, S., & Hecker, J. P. (2017). A Multi-Agent System Approach to Load-Balancing and Resource Allocation for Distributed Computing. In *First Complex Systems Digital Campus World E-Conference 2015* (pp. 41-54). Springer, Cham.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... & Warfield, A. (2003, October). Xen and the art of virtualization. In *ACM SIGOPS operating systems review* (Vol. 37, No. 5, pp. 164-177). ACM.

Baswade, A. M., & Nalwade, P. S. (2013). Selection of initial centroids for k-means algorithm. *IJCSMC*, *2*(7), 161-164.

Beegom, A. A., & Rajasree, M. S. (2014, October). A particle swarm optimization based pareto optimal task scheduling in cloud computing. In *International Conference in Swarm Intelligence* (pp. 79-86). Springer, Cham.

Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, *28*(5), 755-768.

Beloglazov, A., & Buyya, R. (2010, May). Energy efficient resource management in virtualized cloud data centers. In *Proceedings of the 2010 10th IEEE/ACM international conference on cluster, cloud and grid computing* (pp. 826-831). IEEE Computer Society.

Beloglazov, A., & Buyya, R. (2012). Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of

virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, *24*(13), 1397-1420.

Belal, M., & El-Ghazawi, T. (2004). Parallel models for particle swarm optimizers. *International Journal on Intelligent Cooperative Information Systems*, *4*(1), 100-111.

Benameur, L., Alami, J., & El Imrani, A. (2009, September). A new hybrid particle swarm optimization algorithm for handling multiobjective problem using fuzzy clustering technique. In *Computational Intelligence, Modelling and Simulation, 2009. CSSim'09. International Conference on* (pp. 48-53). IEEE.

Berkhin, P. (2006). A survey of clustering data mining techniques. *Grouping multidimensional data*, *25*, 71.

Blum, C., & Li, X. (2008). Swarm intelligence in optimization. In *Swarm Intelligence* (pp. 43-85). Springer Berlin Heidelberg.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, *35*(3), 268-308.

Bohra, A. E. H., & Chaudhary, V. (2010, April). VMeter: Power modelling for virtualized clouds. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on* (pp. 1-8). Ieee.

Bousselmi, K., Brahmi, Z., & Gammoudi, M. M. (2016, March). QoS-aware scheduling of workflows in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2016 IEEE 30th International Conference on* (pp. 737-745). IEEE.

Buyya, R., Beloglazov, A., & Abawajy, J. (2010). Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges. *arXiv preprint arXiv:1006.0308*.

Buyya, R., Vecchiola, C., & Selvi, S. T. (2013). *Mastering cloud computing: foundations and applications programming*. Newnes.

Cabrera, J. C. F., & Coello, C. A. C. (2010). Micro-MOPSO: A Multi-Objective Particle Swarm Optimizer That Uses a Very Small Population Size.

Cagnina, L., Esquivel, S. C., & Coello Coello, C. (2005). A particle swarm optimizer for multi-objective optimization. *Journal of Computer Science & Technology*, *5*.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, *41*(1), 23-50.

Chaisiri, S., Lee, B. S., & Niyato, D. (2009, December). Optimal virtual machine placement across multiple cloud providers. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific* (pp. 103-110). IEEE.

Chang, J. F., Roddick, J. F., Pan, J. S., & Chu, S. C. (2005). A parallel particle swarm optimization algorithm with communication strategies.

Chen, J., Han, X., & Jiang, G. (2014). A negotiation model based on multiagent system under cloud computing. In *The Ninth International Multi-Conference on Computing in the Global Information Technology* (pp. 157-164).

Chen, W. N., & Zhang, J. (2009). An ant colony optimization approach to a grid workflow scheduling problem with various QoS requirements. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *39*(1), 29-43.

Cho, K. M., Tsai, P. W., Tsai, C. W., & Yang, C. S. (2015). A hybrid meta-heuristic algorithm for VM scheduling with load balancing in cloud computing. *Neural Computing and Applications*, *26*(6), 1297-1309.

Chow, C. K., & Tsui, H. T. (2004, June). Autonomous agent response learning by a multi-species particle swarm optimization. In *Evolutionary Computation, 2004. CEC2004. Congress on* (Vol. 1, pp. 778-785). IEEE.

Copil, G., Moldovan, D., Salomie, I., Cioara, T., Anghel, I., & Borza, D. (2012, August). Cloud SLA negotiation for energy saving—A particle swarm optimization

approach. In *Intelligent Computer Communication and Processing (ICCP), 2012 IEEE International Conference on* (pp. 289-296). IEEE.

Dasgupta, K., Mandal, B., Dutta, P., Mandal, J. K., & Dam, S. (2013). A genetic algorithm (ga) based load balancing strategy for cloud computing. *Procedia Technology*, *10*, 340-347.

Dastjerdi, A. V., & Buyya, R. (2012, May). An autonomous reliability-aware negotiation strategy for cloud computing environments. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)* (pp. 284-291). IEEE Computer Society.

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, *1*(4), 28-39.

Durillo, J. J., García-Nieto, J., Nebro, A. J., Coello, C. A. C., Luna, F., & Alba, E. (2009, April). Multi-objective particle swarm optimizers: An experimental comparison. In *International Conference on Evolutionary Multi-Criterion Optimization* (pp. 495-509). Springer, Berlin, Heidelberg.

Eberhart, R., & Kennedy, J. (1995, October). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on* (pp. 39-43). IEEE.

Eberhart, R. C., & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on* (Vol. 1, pp. 84-88). IEEE

Emeakaroha, V. C., Netto, M. A., Calheiros, R. N., Brandic, I., Buyya, R., & De Rose, C. A. (2012). Towards autonomic detection of SLA violations in Cloud infrastructures. *Future Generation Computer Systems*, *28*(7), 1017-1029.

Esmaeili, A., & Mozayani, N. (2010, June). Improving multi-agent negotiations using multi-objective PSO algorithm. In *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications* (pp. 92-101). Springer, Berlin, Heidelberg.

Fan, S. K. S., & Chang, J. M. (2009). A parallel particle swarm optimization algorithm for multi-objective optimization problems. *Engineering Optimization*, *41*(7), 673-697.

Feng, M., Wang, X., Zhang, Y., & Li, J. (2012, October). Multi-objective particle swarm optimization for resource allocation in cloud computing. In *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on* (Vol. 3, pp. 1161-1165). IEEE.

Figueiredo, E. M., Ludermir, T. B., & Bastos-Filho, C. J. (2016). Many objective particle swarm optimization. *Information Sciences*, *374*, 115-134.

Foster, I., Zhao, Y., Raicu, I., & Lu, S. (2008, November). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08* (pp. 1-10). Ieee.

Gan, G., Ma, C., & Wu, J. (2007). *Data clustering: theory, algorithms, and applications*. Society for Industrial and Applied Mathematics.

Gao, Y., Guan, H., Qi, Z., Hou, Y., & Liu, L. (2013). A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, *79*(8), 1230-1242.

Gao, Y., Peng, L., Li, F., Liu, M., & Liu, W. (2014, August). Multi-objective cloud estimation of distribution particle swarm optimizer using maximum ranking. In *Natural Computation (ICNC), 2014 10th International Conference on* (pp. 321-325). IEEE.

Garg, H. (2016). A hybrid PSO-GA algorithm for constrained optimization problems. *Applied Mathematics and Computation*, *274*, 292-305.

Garza-Fabre, M., Pulido, G., & Coello, C. (2009). Ranking methods for many-objective optimization. *MICAI 2009: Advances in Artificial Intelligence*, 633-645.

Gautier, T., Besseron, X., & Pigeon, L. (2007, July). Kaapi: A thread scheduling runtime system for data flow computations on cluster of multi-processors. In *Proceedings of the 2007 international workshop on Parallel symbolic computation* (pp. 15-23). ACM.

Gendreau, M., & Potvin, J. Y. (2005). Metaheuristics in combinatorial optimization. *Annals of Operations Research*, *140*(1), 189-213.

Gonsalves, T., & Egashira, A. (2013). Parallel swarms oriented particle swarm optimization. *Applied Computational Intelligence and Soft Computing*, *2013*, 14.

Govindarajan, K., Somasundaram, T. S., & Kumar, V. S. (2013, July). Particle swarm optimization (PSO)-based clustering for improving the quality of learning using cloud computing. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on* (pp. 495-497). IEEE.

Grama, A. (2003). *Introduction to parallel computing*. Pearson Education.

Guo, L., Shao, G., & Zhao, S. (2012, September). Multi-objective task assignment in cloud computing by particle swarm optimization. In *Wireless Communications, Networking and Mobile Computing (WiCOM), 2012 8th International Conference on* (pp. 1-4). IEEE.

Gupta, G., Kumawat, V. K., Laxmi, P. R., Singh, D., Jain, V., & Singh, R. (2014, August). A simulation of priority based earliest deadline first scheduling for cloud computing system. In *Networks & Soft Computing (ICNSC), 2014 First International Conference on* (pp. 35-39). IEEE.

Hao, Y., Wang, L., & Zheng, M. (2016). An adaptive algorithm for scheduling parallel jobs in meteorological Cloud. *Knowledge-Based Systems*, *98*, 226-240.

Hashmi, K., Alhosban, A., Malik, Z., & Medjahed, B. (2011, July). Webneg: A genetic algorithm based approach for service negotiation. In *Web Services (ICWS), 2011 IEEE International Conference on* (pp. 105-112). IEEE.

Hatamlou, A., Abdullah, S., & Othman, Z. (2011, June). Gravitational search algorithm with heuristic search for clustering problems. In *Data mining and optimization (DMO), 2011 3rd conference on* (pp. 190-193). IEEE.

He, P., Liang, Y., & Chou, X. (2014, August). Resource Scheduling Algorithm in Embedded Cloud Computing and Application. In *Advanced Applied Informatics (IIAIAAI), 2014 IIAI 3rd International Conference on* (pp. 425-429). IEEE.

Hemalatha, M. (2013). Cluster Based Bee Algorithm for Virtual Machine Placement in Cloud Data Center. *Journal of Theoretical & Applied Information Technology*, *57*(3).

Huang, Lu, Hai-shan Chen, and Ting-ting Hu. "Survey on Resource Allocation Policy and Job Scheduling Algorithms of Cloud Computing1." *JSW* 8.2 (2013): 480-487.

Izakian, H., Ladani, B. T., Abraham, A., & Snasel, V. (2010). A discrete particle swarm optimization approach for grid job scheduling. *International Journal of Innovative Computing, Information and Control*, *6*(9), 1-15.

Izakian, H., Ladani, B. T., Zamanifar, K., & Abraham, A. (2009, March). A novel particle swarm optimization approach for grid job scheduling. In *International Conference on Information Systems, Technology and Management* (pp. 100-109). Springer, Berlin, Heidelberg.

Jayanthi, S. (2014, November). Literature review: Dynamic resource allocation mechanism in cloud computing environment. In *Electronics, Communication and Computational Engineering (ICECCE), 2014 International Conference on* (pp. 279-281). IEEE.

Jena, R. K. (2015). Multi objective task scheduling in cloud environment using nested PSO framework. *Procedia Computer Science*, *57*, 1219-1227.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial intelligence*, *117*(2), 277-296.

Jennings, N. R., & Wooldridge, M. (1995). Applying agent technology. *Applied Artificial Intelligence an International Journal*, *9*(4), 357-369.

Kaur, N., & Chhabra, A. (2016, March). Analytical review of three latest nature inspired algorithms for scheduling in clouds. In *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on* (pp. 3296-3300). IEEE.

Kattan, A., & Fatima, S. (2012, June). Pso as a meta-search for hyper-ga system to evolve optimal agendas for sequential multi-issue negotiation. In *Evolutionary Computation (CEC), 2012 IEEE Congress on* (pp. 1-8). IEEE.Kennedy, J. (2011).

Particle swarm optimization. In *Encyclopedia of machine learning* (pp. 760-766). Springer US.

Khanesar, M. A., Teshnehlab, M., & Shoorehdeli, M. A. (2007, June). A novel binary particle swarm optimization. In *Control & Automation, 2007. MED'07. Mediterranean Conference on* (pp. 1-6). IEEE.

Khanna, G., Beaty, K., Kar, G., & Kochut, A. (2006, April). Application performance management in virtualized server environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP* (pp. 373-381). IEEE.

Koh, Byung-Il, et al. "Parallel asynchronous particle swarm optimization." *International Journal for numerical methods in engineering* 67.4 (2006): 578-595.

Kolomvatsos, K., & Hadjieftymiades, S. (2014). On the use of particle swarm optimization and kernel density estimator in concurrent negotiations. *Information Sciences*, *262*, 99-116.

Kuo, R. J., Wang, M. J., & Huang, T. W. (2011). An application of particle swarm optimization algorithm to clustering analysis. *Soft Computing*, *15*(3), 533-542.

Laguna-Sánchez, G. A., Olguín-Carbajal, M., Cruz-Cortés, N., Barrón-Fernández, R., & Álvarez-Cedillo, J. A. (2009). Comparative study of parallel variants for a particle swarm optimization algorithm implemented on a multithreading GPU. *Journal of applied research and technology*, *7*(3), 292-307.

Lakra, A. V., & Yadav, D. K. (2015). Multi-objective tasks scheduling algorithm for cloud computing throughput optimization. *Procedia Computer Science*, *48*, 107-113.

Lee, G., Tolia, N., Ranganathan, P., & Katz, R. H. (2010, August). Topology-aware resource allocation for data-intensive workloads. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems* (pp. 1-6). ACM.

Lee, Y. C., Wang, C., Zomaya, A. Y., & Zhou, B. B. (2012). Profit-driven scheduling for cloud services with data access awareness. *Journal of Parallel and Distributed Computing*, *72*(4), 591-602.

Li, K., Deb, K., Zhang, Q., & Kwong, S. (2015). An evolutionary many-objective optimization algorithm based on dominance and decomposition. *IEEE Transactions on Evolutionary Computation*, *19*(5), 694-716.

Lin, C. C., Liu, P., & Wu, J. J. (2011, December). Energy-efficient virtual machine provision algorithms for cloud systems. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on* (pp. 81-88). IEEE.

Li-Ping, Z., Huan-Jun, Y., & Shang-Xu, H. (2005). Optimal choice of parameters for particle swarm optimization. *Journal of Zhejiang University-Science A*, *6*(6), 528-534.

Lissy, A., & Mukhopadhyay, D. (2014). Negotiation in Cloud During Service Level Agreement-A Survey. *International Journal of Advance Foundation and Research in Computer*, *1*(12), 49-58.

Liu, J., Luo, X. G., Zhang, X. M., & Zhang, F. (2013). Job scheduling algorithm for cloud computing based on particle swarm optimization. In *Advanced Materials Research* (Vol. 662, pp. 957-960). Trans Tech Publications.

Liu, H., Xu, D., & Miao, H. K. (2011, December). Ant colony optimization based service flow scheduling with various QoS requirements in cloud computing. In *Software and Network Engineering (SSNE), 2011 First ACIS International Symposium on* (pp. 53-58). IEEE.

Lopez-Pires, F., & Barán, B. (2015). Virtual machine placement literature review. *arXiv preprint arXiv:1506.01509*.

Lopez-Pires, F., Barán, B., Amarilla, A., Benítez, L., Ferreira, R., & Zalimben, S. (2016, October). An Experimental Comparison of Algorithms for Virtual Machine Placement Considering Many Objectives. In *Proceedings of the 9th Latin America Networking Conference* (pp. 1-8). ACM.

Low, C., Hsu, C. J., & Su, C. T. (2010). A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance. *Expert Systems with Applications*, *37*(9), 6429-6434.

Luo, L., Wu, W., Di, D., Zhang, F., Yan, Y., & Mao, Y. (2012, June). A resource scheduling algorithm of cloud computing based on energy efficient optimization methods. In *Green Computing Conference (IGCC), 2012 International* (pp. 1-6). IEEE.

Lu, X., & Gu, Z. (2011, September). A load-adapative cloud resource scheduling model based on ant colony algorithm. In *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on* (pp. 296-300). IEEE.

Lu, X., & Zhang, Z. (2015). A Virtual Machine Dynamic Migration Scheduling Model Based on MBFD Algorithm. *International Journal of Computer Theory and Engineering*, *7*(4), 278.

Liu, S., Quan, G., & Ren, S. (2010, July). On-line scheduling of real-time services for cloud computing. In *Services (SERVICES-1), 2010 6th World Congress on* (pp. 459-464). IEEE.

Madhusudhan, B., & Sekaran, K. C. (2013). A genetic algorithm approach for virtual machine placement in cloud. In *Published in the proceeding of International Conference on Emerging Research in Computing, Information, Communication and Applications (ERCICA2013).*

Madni, S. H. H., Latiff, M. S. A., & Coulibaly, Y. (2016). An appraisal of meta-heuristic resource allocation techniques for IaaS cloud. *Indian Journal of Science and Technology*, *9*(4).

Mahendiran, A., Saravanan, N., Subramanian, N. V., & Sairam, N. (2012). Implementation of K-means clustering in cloud computing environment. *Research Journal of Applied Sciences, Engineering and Technology*, *4*(10), 1391-1394.

Maity, S., & Chaudhuri, A. (2014, October). Optimal negotiation of SLA in federated cloud using multiobjective genetic algorithms. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on* (pp. 269-271). IEEE.

Malathy, G., & Somasundaram, R. (2012). Performance enhancement in cloud computing using reservation cluster. *European Journal of Scientific Research, ISSN*, 394-401.Maltese, J., Ombuki-Berman, B. M., & Engelbrecht, A. P. (2016).

A Scalability Study of Many-Objective Optimization Algorithms. *IEEE Transactions on Evolutionary Computation*.

Malhotra, R., & Jain, P. (2013). Study and comparison of various cloud simulators available in the cloud computing. *International Journal*, *3*(9).

Mansour, K., & Kowalczyk, R. (2011). A meta-strategy for coordinating of one-to-many negotiation over multiple issues. *Foundations of Intelligent Systems*, *122*, 343-353.

Mao, Y., Chen, X., & Li, X. (2014). Max–min task scheduling algorithm for load balance in cloud computing. In *Proceedings of International Conference on Computer Science and Information Technology* (pp. 457-465). Springer, New Delhi.

Mark, C. C. T., Niyato, D., & Chen-Khong, T. (2011, March). Evolutionary optimal virtual machine placement and demand forecaster for cloud computing. In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on* (pp. 348-355). IEEE.

Marler, R. T., & Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, *26*(6), 369-395.

Masdari, M., Salehi, F., Jalali, M., & Bidaki, M. (2016). A Survey of PSO-based scheduling algorithms in cloud computing. *Journal of Network and Systems Management*, *25*(1), 122-158.

Mastelic, T., Oleksiak, A., Claussen, H., Brandic, I., Pierson, J. M., & Vasilakos, A. V. (2015). Cloud computing: Survey on energy efficiency. *Acm computing surveys (csur)*, *47*(2), 33.

Mathew, T., Sekaran, K. C., & Jose, J. (2014, September). Study and analysis of various task scheduling algorithms in the cloud computing environment. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on* (pp. 658-664). IEEE.

Mehdi, N. A., Mamat, A., Ibrahim, H., & Subramaniam, S. K. (2011). On the fly negotiation for urgent service level agreement on intercloud environment. *Journal of Computer Science*, *7*(10), 1596.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.

Messina, F., Pappalardo, G., Santoro, C., Rosaci, D., & Sarné, G. M. (2014, June). An agent based negotiation protocol for cloud service level agreements. In *WETICE Conference (WETICE), 2014 IEEE 23rd International* (pp. 161-166). IEEE.

Milani, F. S., & Navin, A. H. (2015). Multi-objective task scheduling in the cloud computing based on the patrice swarm optimization. *International Journal of Information Technology and Computer Science (IJITCS)*, *7*(5), 61.

Miranda, V., & Fonseca, N. (2002, October). EPSO-evolutionary particle swarm optimization, a new algorithm with applications in power systems. In *Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES* (Vol. 2, pp. 745-750). IEEE.

Mirzayi, S., & Rafe, V. (2013). A survey on heuristic task scheduling on distributed systems. *AWERProcedia Information Technology & Computer Science*, *1*, 1498-1501.

Moorthy, R. S., Somasundaram, T. S., & Govindarajan, K. (2014, September). Failure-aware resource provisioning mechanism in cloud infrastructure. In *Global Humanitarian Technology Conference-South Asia Satellite (GHTC-SAS), 2014 IEEE* (pp. 255-260). IEEE.

Munir, E. U., Li, J. Z., Shi, S. F., & Rasool, Q. (2007, August). Performance analysis of task scheduling heuristics in grid. In *Machine Learning and Cybernetics, 2007 International Conference on* (Vol. 6, pp. 3093-3098). IEEE.

Naik, B., Swetanisha, S., Behera, D. K., Mahapatra, S., & Padhi, B. K. (2012, November). Cooperative swarm based clustering algorithm based on PSO and k-means to find optimal cluster centroids. In *Computing and Communication Systems (NCCCS), 2012 National Conference on* (pp. 1-5). IEEE.

Netjinda, N., Sirinaovakul, B., & Achalakul, T. (2012, May). Cost optimization in cloud provisioning using particle swarm optimization. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2012 9th International Conference on* (pp. 1-4). IEEE.

Neshat, M., Yazdi, S. F., Yazdani, D., & Sargolzaei, M. (2012). A new cooperative algorithm based on PSO and k-means for data clustering.

Nguyen, T. D., & Jennings, N. R. (2003, September). Concurrent bilateral negotiation in agent systems. In *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on* (pp. 844-849). IEEE.

Nguyen, Q. T., Quang-Hung, N., Tuong, N. H., & Thoai, N. (2013, January). Virtual machine allocation in cloud computing for minimizing total execution time on each machine. In *Computing, Management and Telecommunications (ComManTel), 2013 International Conference on* (pp. 241-245). IEEE.

Nurika, O., Paputungan, I. V., & Hassan, M. F. (2014). Performance Oriented Genetic Algorithm Framework of Concurrent SLA Negotiations in Cloud. *World Applied Sciences Journal*, *30*(30), 280-287.

Omara, F. A., Khattab, S. M., & Sahal, R. (2014). Optimum Resource Allocation of Database in Cloud Computing. *Egyptian Informatics Journal*, *15*(1), 1-12.

Omezzine, A., Saoud, N. B. B., Tazi, S., & Cooperman, G. (2016, August). Negotiation based scheduling for an efficient saas provisioning in the cloud. In *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on* (pp. 33-40). IEEE.

Pacini, E., Mateos, C., & García Garino, C. (2014). Dynamic scheduling based on particle swarm optimization for cloud-based scientific experiments. *CLEI Electronic Journal*, *17*(1), 3-3.

Panagidi, K., Kolomvatsos, K., & Hadjiefthymiades, S. (2014). An intelligent scheme for concurrent multi-issue negotiations. *Int. J. Artif. Intell.*, *12*(1), 129-149.

Panchal, B., & Kapoor, R. K. (2013). Dynamic VM allocation algorithm using clustering in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, *3*(9), 143-150.

Pandey, S., Wu, L., Guru, S. M., & Buyya, R. (2010, April). A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced information networking and applications (AINA), 2010 24th IEEE international conference on* (pp. 400-407). IEEE.

Patel, K. S., & Sarje, A. K. (2012, October). VM provisioning method to improve the profit and SLA violation of cloud service providers. In *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on* (pp. 1-5). IEEE.

Pietri, I., & Sakellariou, R. (2016). Mapping virtual machines onto physical machines in cloud computing: A survey. *ACM Computing Surveys (CSUR), 49*(3), 49.

Pittl, B., Mach, W., & Schikuta, E. (2015, December). A negotiation-based resource allocation model in iaas-markets. In *Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on* (pp. 55-64). IEEE.

Pongchairerks, P. (2009). Particle swarm optimization algorithm applied to scheduling problems. *ScienceAsia, 35*(1), 89-94.

Poola, D., Garg, S. K., Buyya, R., Yang, Y., & Ramamohanarao, K. (2014, May). Robust scheduling of scientific workflows with deadline and budget constraints in clouds. In *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on* (pp. 858-865). IEEE.Premalatha, K., & Natarajan, A. M. (2009). Discrete PSO with GA operators for document clustering. *International Journal of Recent Trends in Engineering*, *1*(1), 20-24.

Quang-Hung, N., Thoai, N., & Son, N. T. (2012). Performance constraint and power-aware allocation for user requests in virtual computing lab. *arXiv preprint arXiv:1210.1026*.

Rahwan, I., Kowalczyk, R., & Pham, H. H. (2002, January). Intelligent agents for automated one-to-many e-commerce negotiation. In *Australian Computer Science Communications* (Vol. 24, No. 1, pp. 197-204). Australian Computer Society, Inc.

Rajavel, R., & Thangarathinam, M. (2015). Optimizing Negotiation conflict in the cloud service negotiation framework using probabilistic decision making model. *The Scientific World Journal*, *2015*.

Rana, S., Jasola, S., & Kumar, R. (2011). A review on particle swarm optimization algorithms and their applications to data clustering. *Artificial Intelligence Review*, *35*(3), 211-222.

Ramezani, F., Lu, J., & Hussain, F. (2013, December). Task scheduling optimization in cloud computing applying multi-objective particle swarm optimization. In *International Conference on Service-oriented computing* (pp. 237-251). Springer, Berlin, Heidelberg.

Reyes-Sierra, M., & Coello, C. C. (2006). Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, *2*(3), 287-308.

Sahal, R., Khattab, S. M., & Omara, F. A. (2013, May). GPSO: An improved search algorithm for resource allocation in cloud databases. In *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on* (pp. 1-8). IEEE.

Saini, G., & Kaur, H. (2014). A novel approach towards K-mean clustering algorithm with PSO. *International Journal of Computer Science & Information Technologies*,*5*.

Salman, A., Ahmad, I., & Al-Madani, S. (2002). Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, *26*(8), 363-371.

Sakellari, G., & Loukas, G. (2013). A survey of mathematical models, simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, *39*, 92-103.

Sakr, S., & Liu, A. (2012, June). Sla-based and consumer-centric dynamic provisioning for cloud databases. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on* (pp. 360-367). IEEE.

Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., & George, A. D. (2004). Parallel global optimization with the particle swarm algorithm. *International journal for numerical methods in engineering*, *61*(13), 2296-2315.

Schwickerath, U., Jones, R., Shiers, J., Brook, N., Grandi, C., Eck, C., ... & Gibbard, B. (2005). *LHC computing Grid: Technical Design Report* (No. CERN-LHCC-2005-024). CERN.

Shabeera, T. P., Kumar, S. M., Salam, S. M., & Krishnan, K. M. (2017). Optimizing VM allocation and data placement for data-intensive applications in cloud using ACO metaheuristic algorithm. *Engineering Science and Technology, an International Journal*, *20*(2), 616-628.

Shah, M. M. D., Kariyani, M. A. A., & Agrawal, M. D. L. (2013). Allocation of virtual machines in cloud computing using load balancing algorithm. *International Journal of Computer Science and Information Technology & Security (IJCSITS)*, *3*(1), 2249-9555.

Shankar, A., & Bellur, U. (2010). Virtual Machine Placement in Computing Clouds. *CoRR, vol. abs/1011.5064*.

Shaw, S. B., & Singh, A. K. (2014, September). A survey on scheduling and load balancing techniques in cloud computing environment. In *Computer and Communication Technology (ICCCT), 2014 International Conference on* (pp. 87-95). IEEE.

Shidik, G. F., Sulistyowati, N. S., & Tirta, M. B. (2016, August). Evaluation of cluster K-Means as VM selection in dynamic VM consolidation. In *Communications (APCC), 2016 22nd Asia-Pacific Conference on* (pp. 124-128). IEEE.

Shen, W., Li, Y., Ghenniwa, H., & Wang, C. (2002). Adaptive negotiation for agent-based grid computing. *Journal of the American Statistical Association*, *97*(457).

Shin, S., Kim, Y., & Lee, S. (2015, January). Deadline-guaranteed scheduling algorithm with improved resource utilization for cloud computing. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE* (pp. 814-819). IEEE.

Shindler, M., Wong, A., & Meyerson, A. W. (2011). Fast and accurate k-means for large datasets. In *Advances in neural information processing systems* (pp. 2375-2383).

Shishira, S. R., Kandasamy, A., & Chandrasekaran, K. (2016, September). Survey on meta heuristic optimization techniques in cloud computing. In *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on* (pp. 1434-1440). IEEE.

Silaghi, G. C., ŞErban, L. D., & Litan, C. M. (2012). A time-constrained SLA negotiation strategy in competitive computational grids. *Future Generation Computer Systems*, *28*(8), 1303-1315.

Simon, D. (2013). *Evolutionary optimization algorithms*. John Wiley & Sons.

Sim, K. M. (2013). Complex and concurrent negotiations for multiple interrelated e-markets. *IEEE transactions on cybernetics*, *43*(1), 230-245.

Singh, S., & Chana, I. (2016). A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of grid computing*, *14*(2), 217-264.

Son, S., & Jun, S. C. (2013, May). Negotiation-based flexible SLA establishment with SLA-driven resource allocation in cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on* (pp. 168-171). IEEE.

Srinivasan, D., & Seow, T. H. (2003, December). Particle swarm inspired evolutionary algorithm (ps-ea) for multiobjective optimization problems. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on* (Vol. 4, pp. 2292-2297). IEEE.

Suresh, M., & Karthik, S. (2014, March). A Load Balancing Model in Public Cloud Using ANFIS and GSO. In *Intelligent Computing Applications (ICICA), 2014 International Conference on* (pp. 85-89). IEEE.

Taranti, P. G., de Lucena, C. J. P., & Choren, R. (2011, April). A quantitative study about tardiness in java-based multi-agent systems. In *Agent Systems, their Environment and Applications (WESAAC), 2011 Workshop and School of* (pp. 37-44). IEEE.

Tawfeek, M. A., El-Sisi, A., Keshk, A. E., & Torkey, F. A. (2013, November). Cloud task scheduling based on ant colony optimization. In *Computer Engineering & Systems (ICCES), 2013 8th International Conference on* (pp. 64-69). IEEE.

Thio, N., & Karunasekera, S. (2005, March). Automatic measurement of a qos metric for web service recommendation. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian* (pp. 202-211). IEEE.

Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information processing letters*, *85*(6), 317-325.

Tripathi, P. K., Bandyopadhyay, S., & Pal, S. K. (2007). Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients. *Information Sciences*, *177*(22), 5033-5049.

Tsai, C. W., & Rodrigues, J. J. (2014). Metaheuristic scheduling for cloud: A survey. *IEEE Systems Journal*, *8*(1), 279-291.

Tu, K. Y., & Liang, Z. C. (2011). Parallel computation models of particle swarm optimization implemented by multiple threads. *Expert Systems with Applications*, *38*(5), 5858-5866.

Veeramallu, G. K. S. B. (2014). Dynamically Allocating the Resources Using Virtual Machines. *International Journal of Computer Science and Information Technologies*, *5*(3), 4646-4648.

Verma, J. K., Katti, C. P., & Saxena, P. C. (2014). MADLVF: An energy efficient resource utilization approach for cloud computing. *International Journal of Information Technology and Computer Science (IJITCS)*, *6*(7), 56.

Verma, A., & Kaushal, S. (2014, March). Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud. In *Engineering and Computational Sciences (RAECS), 2014 Recent Advances in* (pp. 1-6). IEEE.

Venter, G., & Sobieszczanski-Sobieski, J. (2006). Parallel Particle Swarm Optimization Algorithm Accelerated by Asynchronous Evaluations. *JACIC*, *3*(3), 123-137.

Wan, L. Y., & Li, W. (2008, July). An improved particle swarm optimization algorithm with rank-based selection. In *Machine Learning and Cybernetics, 2008 International Conference on* (Vol. 7, pp. 4090-4095). IEEE.

Wang, S., Liu, Z., Zheng, Z., Sun, Q., & Yang, F. (2013, December). Particle swarm optimization for energy-aware virtual machine placement optimization in virtualized data centers. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on* (pp. 102-109). IEEE.

Wang, Q., & Zheng, H. C. (2011, April). Optimization of task allocation and knowledge workers scheduling based-on particle swarm optimization. In *Electric Information and Control Engineering (ICEICE), 2011 International Conference on* (pp. 574-578). IEEE.

Wang, Z., Zhang, J., & Si, J. (2014, July). Application of particle swarm optimization with stochastic inertia weight strategy to resources scheduling and assignment problem in cloud manufacturing environment. In *Control Conference (CCC), 2014 33rd Chinese* (pp. 7567-7572). IEEE.

Warneke, D., & Kao, O. (2011). Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. *IEEE transactions on parallel and distributed systems*, *22*(6), 985-997.Wen, X., Huang, M., & Shi, J. (2012, October). Study on resources scheduling based on ACO algorithm and PSO algorithm in cloud computing. In *Distributed Computing and Applications to Business, Engineering & Science (DCABES), 2012 11th International Symposium on* (pp. 219-222). IEEE.

Whitley, D. (2014). An executable model of a simple genetic algorithm. *Foundations of genetic algorithms*, *2*(1519), 45-62.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.

Wu, Z., Ni, Z., Gu, L., & Liu, X. (2010, December). A revised discrete particle swarm optimization for cloud workflow scheduling. In *Computational Intelligence and Security (CIS), 2010 International Conference on* (pp. 184-188). IEEE.

Xia, Q., Lan, Y., & Xiao, L. (2015, August). A Heuristic Adaptive Threshold Algorithm on IaaS Clouds. In *Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), 2015 IEEE 12th Intl Conf on* (pp. 399-406). IEEE.

Xiong, A. P., & Xu, C. X. (2014). Energy efficient multiresource allocation of virtual machine based on PSO in cloud data center. *Mathematical Problems in Engineering*, *2014*.

Xu, H., & Li, B. (2011, April). Egalitarian stable matching for VM migration in cloud computing. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on* (pp. 631-636). IEEE.

Xue, S., Li, M., Xu, X., Chen, J., & Xue, S. (2014). An ACO-LB algorithm for task scheduling in the cloud environment. *Journal of Software*, *9*(2), 466-473.

Yang, X. S. (2010). *Nature-inspired metaheuristic algorithms*. Luniver press.

Yang, X. S., & Deb, S. (2009). Cuckoo search via Lévy flights [C]//Nature & Biologically Inspired Computing. In *2009. NaBIC 2009. World Congress on. IEEE* (pp. 210-214).

Yang, W. A., Guo, Y., & Liao, W. H. (2011). Optimization of multi-pass face milling using a fuzzy particle swarm optimization algorithm. *The International Journal of Advanced Manufacturing Technology*, *54*(1), 45-57.

Ye, K., Huang, D., Jiang, X., Chen, H., & Wu, S. (2010, December). Virtual machine based energy-efficient data center architecture for cloud computing: a performance perspective. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing* (pp. 171-178). IEEE Computer Society.

Ye, X., Liu, S., Yin, Y., & Jin, Y. (2017). User-oriented many-objective cloud workflow scheduling based on an improved knee point driven evolutionary algorithm. *Knowledge-Based Systems*.

Ying, Y., Shou, Y. Y., & Li, M. (2009). Hybrid genetic algorithm for resource constrained multi-project scheduling problem. *Journal of Zhejiang University (Engineering Science)*, *1*, 006.

Zhan, S., & Huo, H. (2012). Improved PSO-based task scheduling algorithm in cloud computing. *Journal of Information & Computational Science*, *9*(13), 3821-3829.

Zhan, Z. H., Liu, X. F., Gong, Y. J., Zhang, J., Chung, H. S. H., & Li, Y. (2015). Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Computing Surveys (CSUR)*, *47*(4), 63.

Zhang, Q., Cheng, L., & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, *1*(1), 7-18.

Zhang, L., & Liu, Q. (2016). An automated multi-issue negotiation mechanism based on intelligent agents in e-commerce. *Journal of Advanced Management Science Vol*, *4*(2).

Zhao, S., Lu, X., & Li, X. (2015). Quality of service-based particle swarm optimization scheduling in cloud computing. In *Proceedings of the 4th International Conference on Computer Engineering and Networks* (pp. 235-242). Springer, Cham.

Zhao, J., Yang, K., Wei, X., Ding, Y., Hu, L., & Xu, G. (2016). A heuristic clustering-based task deployment approach for load balancing using bayes theorem in cloud environment. *IEEE Transactions on Parallel and Distributed Systems*, *27*(2), 305-316.

Zhao, C., Zhang, S., Liu, Q., Xie, J., & Hu, J. (2009, September). Independent tasks scheduling based on genetic algorithm in cloud computing. In *Wireless Communications, Networking and Mobile Computing, 2009. WiCom'09. 5th International Conference on* (pp. 1-4). IEEE.

Zheng, X. (2014). *Qos representation, negotiation and assurance in cloud services*. Queen's University (Canada).

Zheng, X., & Jia, Y. (2011, December). A study on educational data clustering approach based on improved particle swarm optimizer. In *IT in Medicine and Education (ITME), 2011 International Symposium on* (Vol. 2, pp. 442-445). IEEE.

Zheng, X., Martin, P., Brohman, K., & Zhang, M. (2014). Cloud service negotiation: a research report. *International Journal of Business Process Integration and Management 10, 7*(2), 103-113.

Zhou, Y., & Tan, Y. (2009, May). GPU-based parallel particle swarm optimization. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on* (pp. 1493-1500). IEEE.

Zhong, H., Tao, K., & Zhang, X. (2010, July). An approach to optimized resource scheduling algorithm for open-source cloud systems. In *ChinaGrid Conference (ChinaGrid), 2010 Fifth Annual* (pp. 124-129). IEEE.

Zhu, W., Zhuang, Y., & Zhang, L. (2017). A three-dimensional virtual resource scheduling method for energy saving in cloud computing. *Future Generation Computer Systems*, *69*, 66-74.