January 4, 2018

BIO-INSPIRED QOS AWARE RESOURCES ALLOCATION AND MANAGEMENT AT THE CLOUD DATA CENTER

Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

SHRIDHAR G. DOMANAL

DEPARTMENT OF INFORMATION TECHNOLOGY
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575 025, INDIA
JANUARY, 2018

Nothing is good or bad
it all depends on how you interpret it

DECLARATION

By the Ph.D. Research Scholar

I hereby declare that the Research Thesis entitled **BIO-INSPIRED QOS AWARE RESOURCES ALLOCATION AND MANAGEMENT AT THE CLOUD DATA CENTER** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfillment of the requirements for the award of the Degree of **Doctor of Philosophy** in **Department of Information Technology** is a *bonafide report of the research work carried out by me.* The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

(IT13F03, Shridhar G. Domanal)
Department of Information Technology

Place: NITK, Surathkal.

Date:

CERTIFICATE

This is to *certify* that the Research Thesis entitled **BIO-INSPIRED QOS AWARE RESOURCES ALLOCATION AND MANAGEMENT AT THE CLOUD DATA CENTER** submitted by **SHRIDHAR G. DOMANAL**, (Register Number: IT13F03) as the record of the research work carried out by him, is *accepted as the Research Thesis submission* in partial fulfillment of the requirements for the award of degree of **Doctor of Philosophy**.

Prof. G. Ram Mohana Reddy
Research Guide

Prof. G. Ram Mohana Reddy
Chairman - DRPC

Acknowledgements

First and foremost, I would like to extend my sincere gratitude to my research guide, Prof. G. Ram Mohana Reddy, Head of Information Technology Department, for his dedicated help, advice, inspiration, encouragement, enthusiasm, and continuous support, throughout my research career. This research would not have been possible without his support and timely inputs.

I express my sincere thanks to members of my RPAC committee, Prof. Anathanarayana V. S, and Prof. A. Kandasamy for their valuable feedback and constructive suggestions during my research work. I would like to thank Prof. Rajkumar Buyya, The University of Melbourne, Australia, for his valuable guidance. My sincere thanks to Dr. Gopal Pingali, Vice-President, IBM Cloud Centre of Excellence, Bengaluru for providing me an opportunity to work with his team.

I would like to thank my parents, in-laws and all my family members, who gave constant support and encouragement and I dedicate this work to my wife Veena Shridhar Domanal who gave unconditional cooperation, love and exhaustive support during my research work.

I express my heartfelt thanks to my best friends Manjunath, Ashwin, Gokul and Sanjay, along with my co-researchers who supported me all along and made my Ph.D. journey more enjoyable and fun. Finally, I would like to express my gratitude to all teaching and supporting staff of IT Department for their support.

Place: NITK, Surathkal Shridhar G. Domanal

Date: January, 2018

ABSTRACT

Cloud comprises of many hardware and software resources and managing these resources will play an important role in executing a clients request. Now-a-days clients from different parts of the world are demanding for various services at a rapid rate. In this present situation efficient load balancing algorithms will play an vital role in allocating the clients requests and also ensuring the usage of the resources in an intelligent way so that underutilization of the resources will not occur in the cloud environment. Clients demand for different cloud resources w.r.t Service Level Agreement (SLA) in a seamless manner, therefore resource allocation and management plays an important role in Infrastructure as a Service (IaaS) based cloud environment.

Computing systems in the cloud environment heavily rely on virtualization technology and thus makes the servers feasible for independent applications. Further, virtualization process improves the power efficiency of the data centers (consolidation of physical machines (PMs)) and thereby enabling the assignment of multiple virtual machines (VMs) to a single physical PM. These VM instances can be procured in the form of On-Demand and Spot instances. Consequently, some of the PMs in the cloud data center can be turned off (sleep state) and resulting in low power consumption and thus making cloud data center more efficient.

In this research work, the main focus is towards designing and development of efficient QoS aware load balancing and resources allocation/management algorithms using Bio-Inspired techniques which ensures fault tolerant task execution in heterogeneous cloud environment. Experimental results demonstrate that our proposed

Bio-Inspired Load Balancing and QoS Aware Resources Allocation/Management algorithms outperforms peer research and benchmark algorithms in terms of efficient utilization of the cloud resources, improved reliability and reduced average response time.

Keywords: Scheduling, Load Balancing, Resource Management, Virtual Machines, Instances, Data Center, Bio-Inspired.

Table of Contents

List of Figures

List of Tables

Abbreviations

ACO	Ant Colony Optimization
BoT	Bag of Tasks
BRS	Best Resource Selection
CBA	Combinational Backfill Algorithm
CDC	Count to Dimension Change
CL	Confidence Level
CLDBM	Central Load Balanicng Decision Model
CP	Critical Path
CPU	Central Processing Unit
CRM	Customer Relationship Management
CSO	Cat Swarm Optimization
D-CRAS	Distributed-Cloud Resource Allocation System
DAG	Directed Acyclic Graph
DBaaS	DataBase as a Service
DCBT	Divide and Conquer Based Throttled

DCCP	Deadline Constrained Critical Path
DDFTP	Dual Direction File Transfer Protocol
DL	Deadline
DLB	Dynamic Load Balancing
DRF	Dominant Resource Fairness
EA	Evolution Algorithms
EFT	Earliest Finish Time
ERP	Enterprise Resource Planning
ES	Evolutionary Strategy
FCFS	First Come First Serve
GA	Genetic Algorithm
GB	Giga Byte
GP	Genetic Programming
GWO	Grey Wolf Optimization
HBB-LB	Honey Bee Behaviour Load Balancer
HEFT	Heterogeneous Earliest Finish Time
HHSA	Hyper Heuristic Scheduling Algorithm
HPC	High Performance Computing
HR	Human Resource
IaaS	Infrastructure as a Service

IBA	Improved Backfill Algorithm
ICPCP	IaaS Cloud Partial Critical Path
INS	Index Name Server
IoT	Internet of Things
IT	Information Technology
LBMM	Load Balancing Min Min
LLOOVMA	Load Level Optimization On Virtual Machine Allocation
MCSO	Modified Cat Swarm Optimization
MFLOPS	Million Floating-point Operations Per Second
MIP	Mixed Integer Program
MIPS	Million Instructions Per Second
MOSCOA	Multi Objective Scheduling Cuckoo Optimization Algorithm
MPSO	Modified Particle Swarm Optimization
MS	Makespan
ms	millisecond
NN	Neural Networks
OS	Operating System
OSI	Open Systems Interconnection
PaaS	Platform as a Service
PDC	Proportional Deadline Constrained

PFRO	Pareto based Fly Optimization
PMs	Physical Machines
PSHA	Probabilistic Seismic Hazard Analysis
PSO	Particle Swarm Optimization
PySim	Python based Simulator
QoS	Quality of Service
RAM	Random Access Memory
RH	Request Handler
RR	Round Robin
RTT	Round Trip Time
SaaS	Software as a Service
SCEC	South California Earthquake Center
SCS	Scaling Consolidation Scheduling
SD	Standard Deviation
SDN	Standard Defined Network
SGT	Strain Green Tensor
SLA	Service Layer Agreement
SLPSO	Self Adaptive Learning Particle Swarm Optimization
SMP	Seeking Memory Pool
SPC	Self Position Consideration

SRD	Seeking Range of selected Dimension
VI	Virtual Infrastructure
VMM	Virtual Machine Monitor
VMs	Virtual Machines
XaaS	Anything as a Service

Chapter 1

Introduction

This chapter gives an overview of cloud computing, deployment models, Quality of Service (QoS) parameters, different types of instances offered by the cloud data center and Bio-Inspired computing. Further, this chapter gives the details of scheduling, cloud resource management and virtualization. Further, this chapter highlights the motivations for carrying out this research work.

1.1 Cloud Computing

A cloud is an elastic execution environment of resources involving multiple stakeholders and providing the metered service at multiple granularities for a specified level of quality of service. Figure 1.1 shows the complete details of the cloud computing environment and its components. Cloud computing is a form of distributed computing and it follows a dynamic approach of pay-as-you-go model. The cloud acts as a global hub of resources and has a capacity to provide the metered service at any time. From Figure 1.1, it is clearly observed that components such as stakeholders, deployment models, service types and many more are directly related to the cloud system.

Terms such as cloud and data center are used repeatedly as they are closely related to each other. Cloud is an off-premise form of computing that stores the data on the Internet, whereas a data center refers to on-premise hardware that stores data within

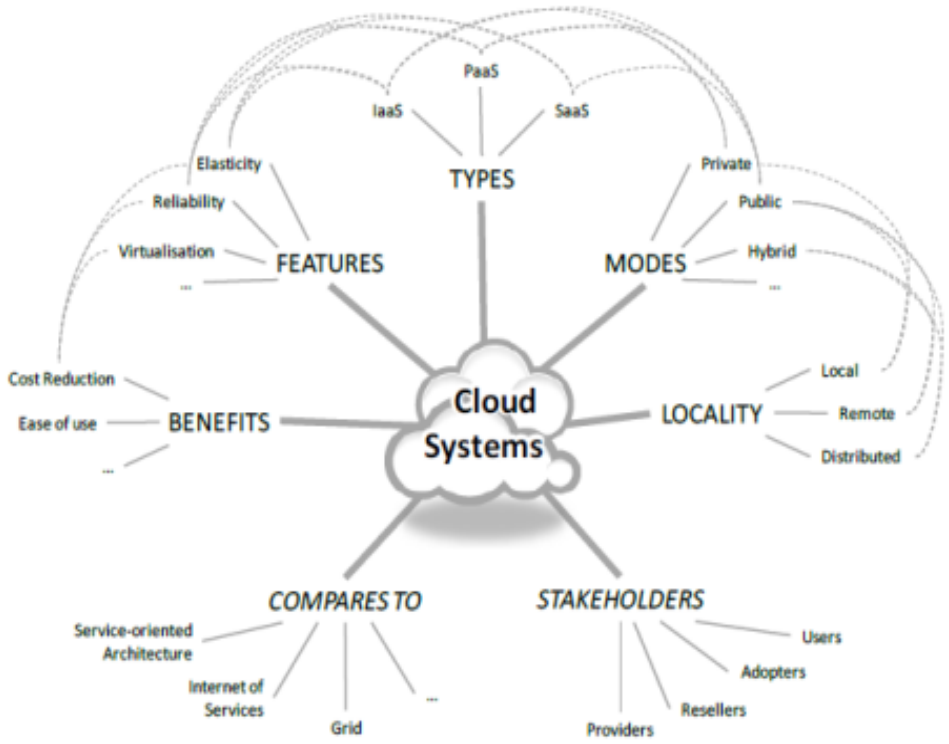


Figure 1.1: Cloud Computing Environment (Hwang et al. (2013))

an organization's local network. Thus, the data center consists of Physical Machines (PMs) which can host several Virtual Machines (VMs) during the computation. Further, it consists of load balancers, diesel generators, cooling units and many more. Data centers are situated at different geographical places and these data centers are connected intelligently by cloud. Users have the choice to use the data center as per their convenience. The data center consists of several components as mentioned above and these are consumed by cloud customers with Service Level Agreements (SLA). A data center can also host different centralized services which are procured by different cloud customers (Hwang et al. (2013)).

The optimum bandwidth network, low cost sharing machines, hardware virtualization and service oriented architecture lead to drastic growth of the cloud computing environment since a few years. Due to the high computing capacity of servers and low cost of services rendered by cloud data centers, the growth rate of cloud is $18.2

billion in 2012 and it is expected to reach $45.6 billion in 2017. Hence, due to this trend, many industries (small and large) are moving to the cloud and because of this the size of the data center is also growing at a rapid rate in order to match the future demand of the cloud users. As mentioned earlier, the data center consists of huge resources and managing these resources is a challenging issue as these resources are consumed by cloud customers. Thus, the emergence of cloud computing has made a tremendous impact on the Information Technology (IT) industry over the past few years, where large companies such as Google, Amazon and Microsoft strive to provide more powerful, reliable and cost-efficient cloud platforms. Accordingly the business enterprises seek to reshape their business models to gain maximum profit from this new paradigm of cloud computing (Liu et al. (2014)). Cloud computing provides several significant features and the details are given below:

- No up-front investment
- Lowering operating cost
- Highly scalable
- Easy access
- Reducing business risks and maintenance expenses.

1.1.1 Characteristics of Cloud

Cloud computing inherits many characteristics from different computing paradigms, but the essential characteristics of the cloud are as follows.

On-demand Services: Each cloud provider has its own portal in which the cloud customer can automatically provide the computing capabilities such as instances, memory and Central Processing Unit (CPU).

Resource Pooling: The data center consists of numerous cloud resources in terms of PMs, VMs etc. and pooled to server multiple cloud customers using multi-tenant model. The resources are dynamically assigned and reassigned according to customers demand. Generally, cloud customers have a higher level of abstraction about the location of the data center.

Broad Network Access: The cloud resources can be easily accessed via the Internet through standard access mechanisms. These mechanisms provide platform independent access through the use of heterogeneous client platforms such as mobiles, laptop, workstation, tablets etc.

Measured Service: Cloud provides its resources to the user based on pay-as-you-go model. The resources consumed by the cloud customers are charged according to its usage. Cloud provides the resources such as Central Processing Unit (CPU) cycles, Random Access Memory (RAM) storage units etc.

Rapid Elasticity: Many industries are moving to the cloud and there is a rapid requirement for cloud resources in a data center. To fulfill the resources demand, the cloud has a model called elasticity in which the resources can be scaled up or scaled down, depending on the cloud customer's requirement.

1.1.2 Computing Paradigms

The technology has evolved from different computing paradigms (Centralized Computing, Parallel Computing, Distributed Computing etc.) and cloud computing environment is evolved from all these computing paradigms.

Centralized Computing: In centralized computing, all computer resources are centralized in one physical system. All resources (processors, memory and storage) are fully shared and tightly coupled within an integrated OS. Many data centers and supercomputers are centralized systems, but they are used in parallel, distributed, and cloud computing applications.

Parallel Computing: In parallel computing, all processors are either tightly coupled with centralized shared memory or loosely coupled with distributed memory and this discipline is referred to as parallel processing. Inter-processor communication is accomplished through shared memory or via message passing.

Distributed Computing: In distributed computing, multiple autonomous computers with their own private memory, communicates through a computer network in peer to peer fashion. Information exchange in a distributed system is accomplished through message passing. It is mainly used in engineering field where the information processing is done at different locations.

Cloud Computing: In cloud computing, the Internet based cloud resources can be either centralized or distributed. The cloud applies parallel or distributed computing, or both paradigms. Clouds can be built with physical or virtualized resources over large data centers that are centralized or distributed. The cloud can be considered as a form of utility computing with different forms of computing paradigms.

1.1.3 Cloud Service Models

In the cloud, the services can be categorized into different models, namely: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) and Anything as a Service (XaaS). Figure 1.2 shows the three important cloud service models and the details of IaaS, PaaS, SaaS and XaaS are as follows.

Infrastructure as a Service (IaaS): This IaaS model puts together infrastructure as demanded by the cloud users, namely: servers, VMs, storage, networks, and the data center fabric. The user can deploy and run on multiple VMs in the guest OSs of specific applications. The user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources (Liu et al. (2014)) and these services are managed by the cloud service provider. Further, the metrics used for billing purpose are based on the type of VMs, number of VMs,

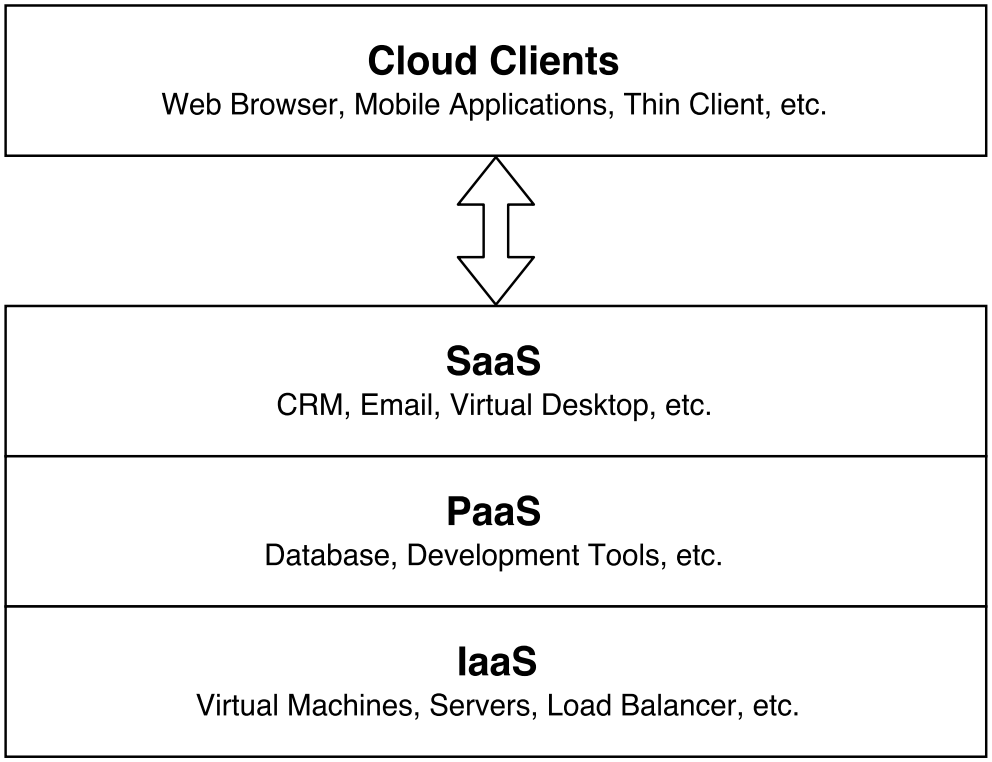


Figure 1.2: Cloud Service Models (Liu et al. (2014))

duration and amount of virtual storage, etc. and these are provisioned based on the pay-as-you-go paradigm.

Platform as a Service (PaaS): This PaaS model enables the user to deploy user-built applications onto a virtualized cloud platform. The PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java. The platform includes both hardware and software integrated with specific programming interfaces. The provider supplies the Application Programming Interface (API) and software tools (e.g., Java, Python, Web 2.0, .NET). The user is freed from managing the cloud infrastructure.

Software as a Service (SaaS): This SaaS model refers to browser-initiated application software over thousands of paid cloud customers. These SaaS applications are platform independent, and these services can be accessed by different client devices such as tablets, mobile, smart phones, etc. The SaaS model applies to business

processes, industry applications, Consumer Relationship Management (CRM), Enterprise Resource Planning (ERP), Human Resources (HR), and collaborative applications. On the customer side, there is no upfront investment in servers or software licensing and clients will not be aware of underlying cloud infrastructure (servers, operating systems, network etc.). On the service provider side, costs are rather low in comparison with conventional hosting of user applications. Further, these applications can be accessed anywhere from the client side.

Anything as a Service (XaaS): Earlier services are categorized into IaaS, PaaS and SaaS but nowadays the cloud is capable of hosting any type of application which can be rendered as XaaS. Cloud can host a different database service, then the services related to these databases are referred as a Data Base as a Service (DBaaS). Similarly, many services including the security can be hosted by cloud providers and these can be listed under XaaS service model.

1.1.4 Cloud Deployment Models

The cloud can be categorized into different deployment models depending on its size, ownership and access methodology. Each model has its own nature and properties of the cloud which is suited to the specific client environment. Figure 1.3 shows the four cloud deployment models such as Private, Public, Community and Hybrid and details of each deployment are as follows.

Private Cloud: The private cloud is deployed in a secure environment using firewalls which are managed by the governance of the corporate department. This type of cloud has its own boundaries and permits only authorized users to access the cloud. As the private cloud is restricted to its users and environment and therefore it is also called Internal cloud. Applications such as dynamic needs, mission critical assignments, etc. are better suited to adapt private cloud environment.

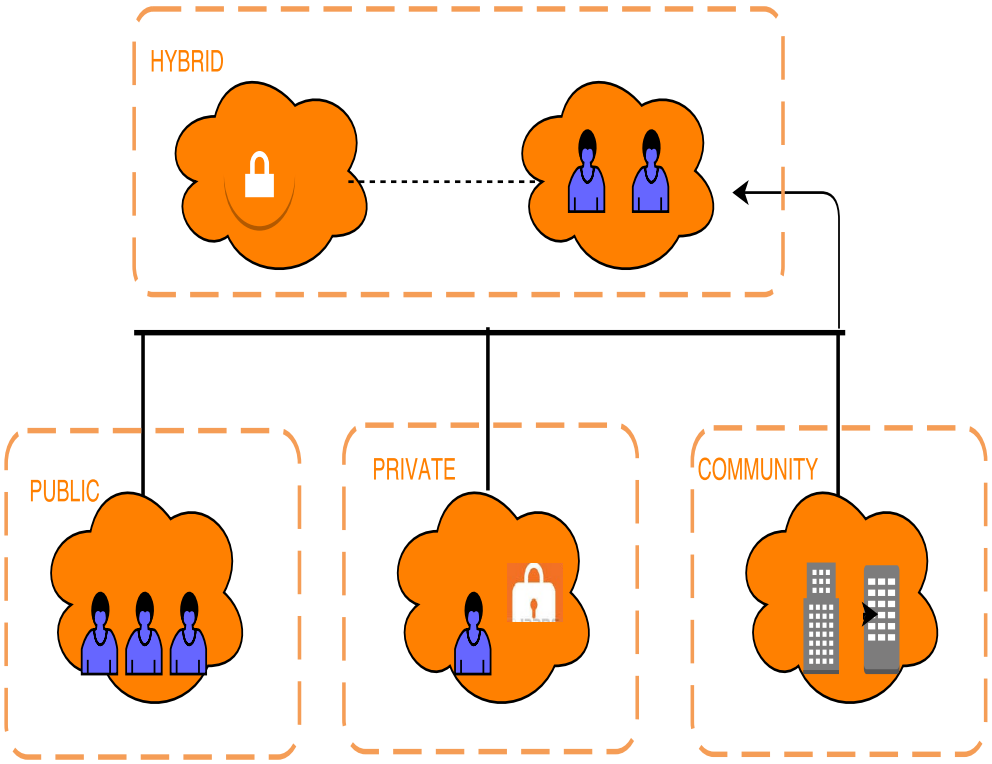


Figure 1.3: Cloud Deployment Models

Public Cloud: The public cloud consists of all the infrastructure (servers, VMs, communication network, etc.) which is hosted by the cloud environment. Typically, these services are consumed by cloud customers based on the pay-as-you-go model. Public cloud provides services in a seamless manner and hence, this type of cloud is best suited for government organizations, academic institutions, research organizations, business enterprises, etc. Due to the decreased capital overheads and operational cost, this model is economical when compared to other deployment models. As an example, Google cloud is referred to as public deployment model.

Community Cloud: This type of cloud is mutually shared between many organizations that belong to a particular community, i.e. banks and trading firms. It is multi-tenant setup which is shared among several organizations that belong to a specific group. All the members of the organizations can access the cloud with the same security policies. This type of cloud can be hosted internally or externally and

the budget required to setup this type of cloud is shared by each organization and hence, this model is cheaper and has cost saving capacity.

Hybrid Cloud: It is an integrated deployment cloud model of private, public or community cloud models which are bound together and serve the cloud customers according to their model principles. It cannot be categorized into private or public cloud model as it crosses the isolation and boundary checks given by the organization. It has an ability to increase its resources according to cloud usage. Hence, this type of cloud model is much suitable for e-commerce applications and can be hosted externally or internally. Usually critical, deadline oriented applications are hosted in a private cloud and non critical applications are deployed on public cloud.

1.1.5 Virtualization

In cloud computing, virtualization plays a major role as it emulates the underlying PM and creates many VMs within a given PM. A conventional computer has a single OS image and it offers a rigid architecture that tightly couples the application software to a specific hardware platform. Some software runs on a PM may not be executable on another platform with a different instruction set under a fixed OS (Hwang et al. (2013)). Virtual machines (VMs) offer novel solutions to underutilized resources, application inflexibility, software manageability, and security concerns in existing physical machines. Figure 1.4 (Hwang et al. (2013)) shows the status of a computer machine before and after virtualization.

A traditional computer runs with a host operating system specially tailored for its hardware architecture as shown in Figure 1.4a. After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software referred to as virtualization layer as shown in Figure 1.4b. This virtualization layer is known as a hypervisor or Virtual Machine Monitor (VMM). The VMs are

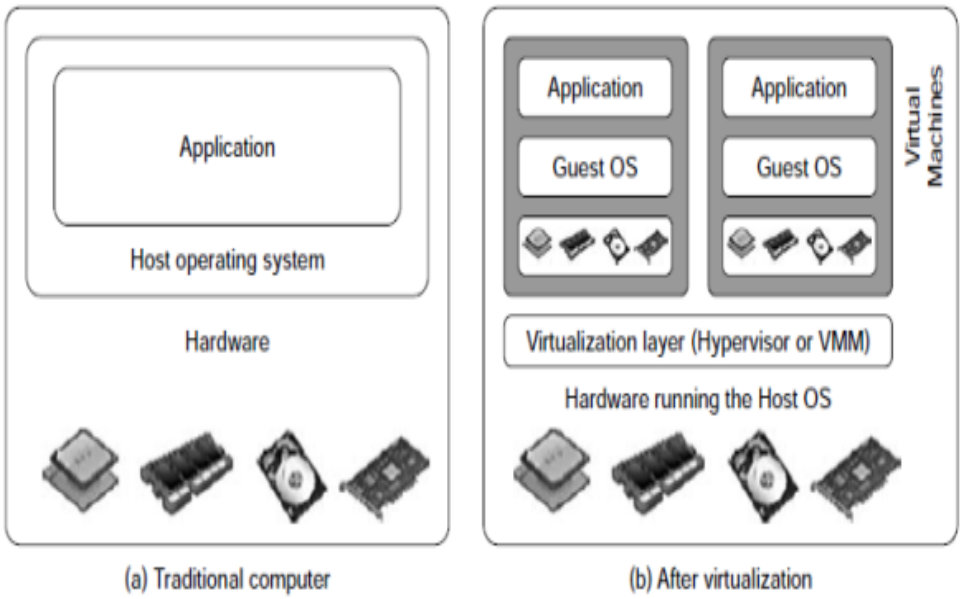


Figure 1.4: Cloud Deployment Models (Hwang et al. (2013))

shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources.

Hypervisor: Hypervisor is a firmware or hardware which acts as an intermediate layer between the host and guest operating systems (i.e. layer between PM and VMs). The hypervisor is responsible for the correct functioning of the VM which is hosted on the PM and hence, hypervisors are also called VMM. The hypervisor is classified into the following two categories (types):

- **Type-1 or bare-metal hypervisors**

In this type, hypervisors run directly on the host's hardware and they are able to manage the guest operating system. These hypervisors do not need a host operating system and hence, these are also called as bare-metal hypervisors.

Example: Xen, Hyper-V and ESX/ESXi.

- **Type-2 or hosted hypervisors**

In this type, hypervisors sit on top of the host operating system. Type-2 hypervisors abstract guest operating systems from the host operating system.

Example: VMware Workstation, QEMU and VirtualBox.

1.1.6 Cloud Instances and QoS Parameters

Typically, data center consists of many cloud resources, including VMs. Usually VMs are referred to as cloud instance, i.e. an instance is nothing but a VM which is offered to the cloud customer on a pay-as-you-go model. Different cloud providers offer many types of instances. Most commonly used cloud instances are discussed below.

On-Demand Instance: In this type of instance, the cloud customer has to pay on an hourly basis. Here, the user has a flexibility to increase or decrease the compute capacity depending on the application scenario. There is no long term commitment for this type of instances and user will follow pay-as-you-go model. This type of instances are suited for applications with short term deadlines or workloads which cannot be interrupted.

Reserved Instance: In this type of instance, the cloud customer can reserve the compute instances for the future scheduling of the applications. The working model will be same as that of On-Demand instances, but the price can be significantly reduced if the customer reserves the instances for long term usage. This type of instances are typically used by the applications in which the need for computing capacity keeps on fluctuating.

Spot Instance: In this type of instance, the cloud customer can get the instance at a lower price compared to On-Demand instances. Here, the cloud customers can bid for spare instances to procure the Spot instances. Many customers can bid for Spot instances, and the Spot instance will be assigned to the user whose bid will be higher. Usually the price of the Spot instances keeps on fluctuating, and it is region

specific. Predicting the bid price will be a major challenge and these are not reliable if the applications are critical. The Spot instance is taken off from the cloud customer if another user bids higher than the current customer. This is the major drawback of using Spot instances. If the Spot instance is terminated by the cloud provider, then the user will not be charged for the Spot instance usage. But, if the Spot instance is deleted by the customer then payment will be on hourly basis. For example: even one minute usage of Spot instance is treated as one hour.

The following QoS parameters play a major role in cloud computing:

- Performance
  - Speed and Capacity.
- Reliability
  - Availability, Consistency and Throughput w.r.t cloud resources.
- Time
  - Response Time and Execution Time.
- Cost
  - Bidding Strategy and Billing.

1.1.7 Load Balancing

In a cloud environment, load balancing technique plays a major role as it distributes the incoming load on the available cloud resources. Each data center has one or many load balancers and maintains the uniform load on the servers within a data center. Cloud has an ability to scale its resources using elasticity property and hence, load balancers are needed to maintain this type of activity.

Cloud customers' workloads are increasing day by day and scheduling of these workloads is also a challenging issue. Before balancing, the incoming workloads have

to be scheduled in an intelligent way so that cloud resources are utilized in an efficient manner. Using load balancing, QoS parameters (Reliability and Time) are significantly optimized and thus leading to efficient utilization of cloud resources with minimum response time. The details of commonly used load balancing algorithms along with scheduling are as follows.

Round Robin: In this algorithm, servers or VMs in the cloud data center are assigned one after the other to serve the incoming users' tasks. It is a very simple load balancing technique and typically suited for the homogeneous cloud environment.

Weighted Round Robin: It is an extension to the Round Robin technique in which the servers are assigned with some static numerical weights. Usually servers which have higher weight-age will receive more user requests.

Least Connections: In this algorithm, the load on each server is considered before scheduling with them. The user requests are assigned to the server with the least active connections.

Adaptive Load Balancing: In this algorithm, there will be an agent assigned to each server for continuous monitoring of the load on each server. The new task will be assigned to the server with less load. The same process continues and maintains the balanced load on the available servers.

There are a few more load balancing algorithms such as Priority Based, Session Based, Chained Failover etc.

1.2 Bio-Inspired Computing

Bio-inspired computing is a popular technique for solving the optimization problem and can be defined as Designing distributed problem-solving devices or algorithms inspired by the behavior of social insects and other animal societies (Bonabeau et al. (1999)). Bio-Inspired computing heavily relies on the fields of Biology, Computer Science and Mathematics. It is a phenomenon of using behavior of living creatures and

modelling them into different frameworks and using these frameworks or approaches in the fields of computer science, engineering and mathematics. These techniques work in a distributed fashion and help to find out the optimal solution in a given environment. The details of different bio-inspired techniques are as follows.

1.2.1 Bio-Inspired Techniques

Bio-Inspired techniques are broadly classified into the following categories (LD and Krishna (2013)).

- Evolution Based
- Ecology Based
- Swarm Intelligence Based.

Evolution Based: Evolution Algorithms (EA) are the most well known, classical and established algorithms among nature inspired algorithms which are based on the biological evolution in nature that are being responsible for the design of all living beings on earth, and for the strategies they use to interact with each other. EA employ this powerful design philosophy to find solutions to hard problems. EA are non-deterministic algorithms or cost based optimization algorithms.

A family of successful EA comprises of Genetic Algorithm (GA), Genetic Programming (GP), Differential Evolution, Evolutionary Strategy (ES) and most recent Paddy Field Algorithm. The members of the EA family share a great number of features in common. They are all population-based stochastic search algorithms performing with survival of the fittest criteria. Each algorithm commences by creating an initial population of feasible solutions, and evolves iteratively from generation to generation towards the best solution. In successive iterations of the algorithm, fitness-based selection takes place within the population of solutions. Better solutions are preferentially selected for survival into the next generation of solutions.

Ecology Based: Natural ecosystems provide a rich source of mechanisms for designing and solving difficult engineering and computer science problems. It comprises the living organisms along with the abiotic environment with which organisms interact with, such as air, soil, water etc. There can be numerous and complex types of interactions among the species of the ecosystem. Also, this can occur as interspecies interaction (between species) or intraspecies interaction (within species). The nature of these interactions can be cooperative or competitive.

Swarm Intelligence Based: Swarm Intelligence (Neumann and Witt (2013)) is a recent and emerging paradigm in Bio-Inspired computing for implementing adaptive systems. Swarm Intelligent encompasses the implementation of the collective intelligence of groups of simple agents that are based on the behavior of real world insect swarms as a problem solving tool. The word swarm comes from the irregular movements of the particles in the problem space (Binitha et al. (2012)).

Swarm Intelligence can be described by five fundamental principles and the details are as follows:

Proximity Principle: The population should be able to carry out simple space and time computations.

Quality Principle: The population should be able to respond to quality factors in the environment.

Diverse Response Principle: The population should not commit its activity along excessively narrow channels.

Stability Principle: The population should not change its mode of behavior every time the environment changes.

Adaptability Principle: The population should be able to change its behavior mode when it is worth the computational price.

These swarm intelligence techniques play a major role in the application which are dynamic in nature and needs optimization, better efficiency, etc. These techniques are much suited to cloud computing environment in which the algorithms related to

cloud data center falls under NP-Hard/NP-Complete complexity classes. Next, we briefly discuss the different Bio-Inspired algorithms.

1.2.2 Different Bio-Inspired Algorithms

Ant Colony Optimization (ACO): ACO is among the most successful swarm intelligence based algorithms proposed by Dorigo Di Caro in 1999. It is a metaheuristic approach based on foraging behavior of ants (Nishant et al. (2012)).

Particle Swarm Optimization (PSO): PSO is a computational intelligence oriented, stochastic, population-based meta-heuristic technique proposed by Kennedy and Eberhart in 1995 (Zuo et al. (2014)).

Grey Wolf Optimizer (GWO): GWO based meta-heuristic Bio-Inspired algorithm mimics the leadership hierarchy and hunting mechanism of grey wolves from the nature (Mirjalili et al. (2014)).

Bat-Termite Algorithm: Hybrid approach by combining the unique features of both social insect termites and mammals bats.

Honey Bee Algorithm: Is is an optimization algorithm based on the intelligent foraging behavior of honey bee swarm, proposed by Karaboga in 2005.

1.3 Motivation

Even though there is much advancement in the areas of cloud computing and Internet of Things (IoT) with respect to responsiveness, reliability and flexibility but, still there is a room for improvement in scheduling, load balancing, optimal resource allocation and management algorithms since these techniques come under NP-Hard/NP-Complete complexity classes. Many researchers developed efficient scheduling and load balancing algorithms, but optimization is still needed in dynamic resource allocation along with VMs migration. Many scheduling algorithms focus only on hardware

utilization of the resources, but fail to address other important QoS parameters (Reliability, Time, Cost and Performance). Once the tasks are assigned by the scheduler then the load balancer plays a vital role in balancing the cloud data center. Hence, the algorithm should be capable enough to schedule and balance the load on the servers in the cloud data center.

Cloud is a global hub of resources which should be managed efficiently and effectively. Many algorithms related to scheduling and resource utilization focus on single objective and works on homogeneous systems. Service Level Agreement (SLA) plays a major role in executing the clients' task within a given deadline and violation of SLA leads to high cost and low performance. Hence, research in this thesis focuses on the design and development of scheduling the clients' task on VMs, resource allocation and management algorithms and thus motivated the following issues:

- Optimizing the Reliability and Time QoS parameters by efficient scheduling and load balancing algorithms.
- Maintaining the balanced load on the servers in a cloud data center.
- Effective and efficient utilization of cloud resources using heuristic algorithms which leads to high performance of the cloud data center.
- Procurement of different types of cloud instances leads to optimize the cost incurred in executing the clients' tasks.
- Providing stability to the cloud data center by optimizing several QoS parameters (Reliability, Time, Throughput, Cost, etc.).

1.4 Outline of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2, existing state-of-the-art algorithms related to scheduling and load balancing are discussed and analyzed. Further, the algorithms related to resource allocation and management are also discussed. The procurement of different cloud instances, workloads are discussed in detail. Based on the outcome of literature review, the problem statement and research objectives are defined.

In Chapter 3, we proposed three scheduling and load balancing algorithms. The proposed algorithms such as Modified Throttled, VM-Assign and Divide and Conquer based Throttled (DCBT) are compared with other state-of-the-art algorithms for performance evaluation.

In Chapter 4, the focus is given on efficient utilization of cloud resources. Here, we proposed Modified Particle Swarm Optimization (MPSO) for scheduling the incoming tasks. Further, we proposed Modified Cat Swarm Optimization (MCSO) and HYBRID (MPSO+MCSO) algorithms for efficient utilization of cloud resources (CPU and Memory). Further, statistical analysis and time complexity of the HYBRID algorithm is discussed.

In Chapter 5, we proposed an application of Grey Wolf Optimizer (GWO) algorithm for efficient scheduling of Bag of Tasks (BoT) and scientific workflows. Further, the performance analysis is carried out in terms of Time and Cost QoS parameters and then compared with existing algorithms including branch and bound based Exact algorithm.

In Chapter 6, the focus is given to the cloud instance procurement and its usage. The details about On-Demand and Spot instances are highlighted. Further, we predicted the prices of future Spot instances using Neural Network (NN) based Back Propagation algorithm which utilizes the past history of Spot instance values. Further, the cost comparison is analyzed using On-Demand and Spot instances.

In Chapter 7, We validated the scheduling algorithm (MPSO) and resource allocation algorithm (HYBRID (MPSO+MCSO)) using the IBM cloud setup. Further, the results of IBM cloud setup are compared with those of PySim cloud simulation setup.

Finally, Chapter 8 summarizes the contributions of the research work and highlights possible future research directions of this thesis.

1.5 Summary

In this chapter, we introduced cloud computing, characteristics, service models, deployment models, cloud instances, QoS parameters and other important cloud technologies. Further, the details of Bio-Inspired computing techniques are elaborated. Finally, we discussed about the motivation for doing this research work. The next chapter deals with background and related work in detail.

Chapter 2

Literature Review

In this Chapter, the complete review of existing state-of-the-art techniques related to scheduling, load balancing and resource allocation & management in the cloud environment is done in detail. There are several existing works on scheduling, load balancing and resource allocation & management in both homogeneous and heterogeneous cloud computing environments. Cloud receives the clients' request at a rapid rate and it has a vast amount of resources which are managed by cloud service providers. Data center consists of Physical Machines (PMs), Virtual Machines (VMs), Network Switches, Routers, Load Balancers, Database Servers, Backup Servers, etc. and managing these resources is a challenging issue.

The cloud can be used in many ways, i.e. cloud offers infrastructure, platform and service to its cloud customers. For any type of request to the cloud, the resources in the cloud should be managed intelligently. Further, the incoming tasks maybe of BoT (independent tasks) or Workflows (dependent tasks) and scheduling of these tasks on the VMs also play an important role. In the cloud, the performance will be degraded if the servers are either underutilized or overutilized and hence there is a need for efficient handling of these servers so that, the load on the servers is maintained in a balanced way. In IaaS based cloud, the managing of the resources (CPU, RAM, PMs and VMs) is based on the pay-as-you-go model and these resources should be used in an optimal manner. Migration of resources (VMs) plays an important role in fast

execution of the tasks given to the cloud environment. Hence, we discuss in detail the state-of-the-the art algorithms and further, we discuss the outcome of literature review followed by problem statement and the objectives of the research work.

2.1 Task Scheduling and Load Balancing

In this Section, the algorithms related to efficient task scheduling and load balancing in both homogeneous and heterogeneous cloud environment are discussed in detail.

Al Nuaimi et al. (2012) presented efficient algorithms for assigning the clients' requests to available nodes in the cloud. Authors investigated different scheduling and load balancing algorithms which are efficient in terms of QoS parameters (Reliability, Time and Cost). They focused on enhancing the overall performance of the cloud data center environment. According to authors, the load balancer should be simple to implement otherwise they will have negative effect on the performance.

Mondal et al. (2012) presented a soft computing based load balancing algorithm for cloud environment. Authors presented a Stochastic Hill climbing optimization algorithm for task allocation to the VMs. Performance of the algorithm is analyzed in the CloudAnalyst simulator and further, they compared with Round Robin (RR) and First Come First Serve (FCFS) algorithms in terms of Reliability QoS parameter.

Wickremasinghe et al. (2010) developed a new simulator called CloudAnalyst to simulate large-scale Cloud applications of scheduling and load balancing. The simulator is very flexible and the users can add their own policy for scheduling approaches along with the customized machine configurations. Further, performance analysis is carried out for different load balancing policies. Mahalle et al. (2013) experimented different load balancing algorithms (Round Robin, Equal Spaced Current Execution Load, Throttled Load Balancing) in terms of execution time QoS parameter.

Zhang and Zhang (2010) proposed an ant colony based load balancing algorithm in cloud computing environment. Authors developed a prototype to compare their

proposed algorithm with SearchMax and SearchMin algorithms for performance evaluation. Pandey et al. (2010) proposed a particle swarm optimization (PSO) based heuristic algorithm to schedule workflows applications in the cloud environment. Authors focused on two QoS parameters (computation cost and data transmission cost) during experimentation. Further, for performance evaluation, this PSO algorithm is compared with Best Resource Selection (BRS) algorithm for cost analysis. Results demonstrate that this PSO approach can achieve 3 times cost savings when compared to BRS algorithm.

Radojević and Žagar (2011) analyzed the issues of load balancing algorithms at different Open System Interconnection (OSI) layers of the network i.e. session switching at the application layer, packet-switching at the network layer. Further, they proposed the new algorithm that incorporates information from virtualized computer environments and end user experience (decision making approach). This algorithm has a capacity to handle critical conditions during load balancing.

Nishant et al. (2012) proposed the modified Ant Colony Optimization (ACO) algorithm for the cloud environment. Authors described a new method to obtain the local minima from which overloaded and underloaded VMs can be found easily. Further, the performance analysis is carried out by finding the shortest path to the VMs. Indukuri et al. (2012) proposed a novel Johnson's Workflows Scheduling algorithm in the cloud computing environment. Further, this algorithm is compared with FCFS algorithm and the results demonstrated that there is significant influence in terms of QoS parameters (average waiting time and total elapsed time).

Kliazovich et al. (2013) proposed a communication fabric scheduling solution, named e-STAB, which extracts communication parameters rather than computational requirements for job allocation. This algorithm improved the communication overhead between the packets and also improved the quality of running the cloud application. The authors experimented the entire setup in the GreenCloud simulator.

Suresh and Vijayakarthick (2011) proposed the Improved Backfill Algorithm (IBA) for job scheduling in the cloud computing environment. Authors tried to improve the combinational back-fill algorithm (CBA) for optimization. By using the IBA algorithm, the QoS parameters (Elasticity and Time) can be optimized using balanced spiral (BS) load balancing method.

Hsiao et al. (2013) presented a fully distributed load re-balancing algorithm to cope with the load imbalance problem for distributed file system execution in a cloud computing environment. Usually in the cloud the VMs are upgraded, added or deleted depending on the load on the servers. The linear increase in the load makes a central load balancer as a bottleneck. Hence, this load re-balancing algorithm outperformed the central distributed algorithm in terms of load imbalance factor, movement cost and algorithmic overhead.

LD and Krishna (2013) proposed Honey Bee Behavior inspired Load Balancing (HBB-LB) algorithm for maintaining the balanced load on VMs. They considered balancing the non preemptive independent tasks during scheduling. This HBB-LB algorithm outperformed other state-of-the-art techniques in terms of (average execution time and average waiting time) QoS parameters.

Domanal and Reddy (2015) proposed a modified throttled algorithm that efficiently schedules the incoming client tasks to the virtual machines. The authors focused on the response time of the tasks. Later, the same authors proposed another VM-Assign load balancing algorithm which focuses not only on the response time, but also on the efficient utilization of VMs present in the cloud. In both algorithms, scheduling of incoming tasks to the VMs were addressed efficiently, but the authors did not consider the resources such as CPU and Memory which are inevitably demanded by clients tasks across the world. Table 2.1 summarizes the existing works on scheduling and load balancing.

Thinakaran et al. (2017) proposed a constraint-aware hybrid scheduler named Pheonix for optimally scheduling the jobs by minimizing the tail latency. Authors

used a novel constraint resource vector which reorders the jobs from the queue for efficient scheduling. But, this algorithm is not application specific and tries to utilize the cloud resources as efficiently as possible. However, this scheduling algorithm improved the average job response time by 1.9x and 5x times when compared with hybrid and Hawk schedulers respectively.

de Souza et al. (2017) formulated the online Virtual Infrastructure (VI) allocation on Software Defined Network (SDN) based cloud data centers as a Mixed Integer Program (MIP). Authors mainly focused on the resources mapping between SDN and virtualised infrastructure from the cloud environment. This algorithm achieved the low latency rate while executing the jobs when compared to other state-of-the-art approaches.

Table 2.1: Summary of Existing Works on Scheduling and Load Balancing

Authors	**Methodology**	**Remarks**
Daochao et al. (2014)	Job Scheduling in distributed cloud using Dominant Resource Fairness (DRF)	Efficient scheduling is done without load balancing of VMs
Zhou et al. (2016)	Dyna: Workflow-as-a-Service based Scheduling	Single objective, i.e. cost optimization
Zhao et al. (2016)	Load Balancing based on Bayes and Clustering (LB-BC) methods	Compared only with Dynamic Load Balancing (DLB) algorithm
Wu et al. (2012)	Index Name Server (INS)	Complex design and takes more time for execution of tasks
Al Nuaimi et al. (2012)	VM Mapping along with task scheduling	Did not consider node configurations
Akbari and Rashidi (2016)	Task Scheduling in Heterogeneous Cloud Environment using Multi-Objective Scheduling Cuckoo Optimization Algorithm (MOSCOA)	Random assignment of tasks to processors during scheduling
Mohamed and Al-Jaroodi (2011)	Dual Direction File Transfer Protocol (DDFTP)	Algorithm is fast but has more replication of data
Tsai et al. (2014)	Hyper Heuristics Scheduling Algorithm for the cloud computing environment	Heterogeneous environment is not considered for scheduling
Radojević and Žagar (2011)	Central Load Balancing Decision Model (CLDBM)	Single point of failure and less reliable
Gunarathne et al. (2010)	Enhanced MapReduce	High processing time and reduce nodes are overloaded

2.2 Resource Allocation and Management

As we know, the cloud customers demand for cloud resources at a rapid rate and hence, efficient utilization of cloud resources is a challenging problem. Many researchers contributed efficient algorithms in managing the cloud resources and the details are as follows.

Tsai et al. (2014) proposed a hyper-heuristic scheduling algorithm (HHSA) for the cloud computing environment. HHSA focused on the diversity and improvement as the two important key operators in finding the local optima. Further, HHSA dynamically determined the suitable heuristic algorithm to find the better candidate solution for the given application.

Gonçalves et al. (2012) proposed the Distributed Cloud Resource Allocation System (D-CRAS) resource allocation model. D-CRAS model ensured the automatic tuning and monitoring of the cloud resources which guarantee the optimal functioning of the tasks without the violation of SLA.

Bittencourt and Madeira (2011) proposed the new Heterogeneous Earliest Finish Time (HEFT) resource allocation algorithm for task execution. This HEFT algorithm intelligently chose the required resources either from both private and public clouds. Further, this HEFT is compared with Greedy approach and Min-Min algorithm for cost comparison.

Zhou et al. (2010) proposed a load balancing scheme based on dynamic resource allocation policy for virtual machine cluster (VMCTune). Here, several VMs are running in the PMs for task execution. Further, the VMCTune algorithm kept track of cloud resources (CPU, RAM) and then migrated the overloaded VM from one PM to another PM. The algorithm ensured that the task execution will not be paused during the migration and thus efficiently balanced the load in the cloud data center.

Zuo et al. (2014) proposed Self-Adaptive Learning Particle Swarm Optimization (SLPSO)-based scheduling approach for executing the tasks. In this approach, if the resources are not available, then the demanded resources by the tasks can be

procured from the external clouds and thus maximizing the task execution without the violation of SLA. Results demonstrated that this SLPSO algorithm outperformed the basic PSO algorithm in terms of computation time.

Papagianni et al. (2013) focused on providing the unified resource allocation framework for network based clouds by indicating objectives related to cost efficiency and average execution time along with the resource mapping procedure while abiding the user requests for QoS-aware virtual resources. Authors assumed that different clouds are interconnected through network and then adopted a heuristic methodology to address resource mapping constraints. Their framework was capable of efficiently utilizing the cloud resources which are deployed in the different regions.

Addis et al. (2013) proposed scalable distributed hierarchical framework based on a mixed-integer nonlinear optimization of resource management. Further, they presented different resource allocation policies for virtualized cloud environment that satisfy the QoS parameters (Performance and Reliability). This framework also minimized the energy factor while executing the users' task.

Wuhib et al. (2012) proposed a Gossip protocol that ensures fair resource allocation among different cloud locations. This Gossip protocol dynamically managed the resources depending on the load on each cloud site and further resources were also scaled up in terms of PMs at different cloud locations. Authors evaluated the performance of Gossip protocol in terms of resources (CPU and Memory) utilization.

Xu and Li (2013) proposed an Anchor, a general resource management architecture that uses the stable matching framework to map virtual machines to physical servers. They presented many resource allocation policies in the Anchor framework using many to one stable matching theory.

Hussain et al. (2013) presented the existing works on resource allocation strategies in High Performance Computing (HPC) systems in the distributed environment. Authors classified HPC system into cluster, grid and cloud system categories and then defined the characteristics and requirements of each class by extracting the common

attributes for choosing suitable resource allocation strategy.

Taneja and Davy (2017) concentrated on efficient utilization of network based infrastructure and accordingly proposed a module mapping algorithm for deploying application modules in Fog-Cloud Infrastructure for Internet-of-Things (IoT) based applications. Authors focused on the computing modules at the fog layer as a microservices and thus led to quick responses to IoT based applications with no delay. However, consideration of other cloud resources will result in more optimized mapping between fog and cloud layers.

Hans et al. (2016) proposed a heuristic based Best-of-Breed approach to allocate the cloud resources for multimedia applications. Authors mainly focused on the resource provisioning from the cloud for live video streaming. Further, they compared their proposed algorithm with standard tabu search method for selecting the appropriate resources for large cloud resource pool.

HoseinyFarahabady et al. (2016) focused on the data center resources utilization and its overall efficiency by considering the correlation between shared and isolated resource usage patterns. Further, they proposed contention aware resource allocation policy which improves the data center resources utilization by 32 % without significant impact on the QoS enforcement level. Thus, this resource allocation policy reduced the overall energy consumption by 35 %.

Pahlevan et al. (2017) proposed a hyper heuristic algorithm which integrates merits of both heuristic and machine learning based policies. Authors mainly focused on the framing the resource reservation frameworks depending on the VMs characteristics. Further, authors focused on energy efficient data center management by an intelligent machine learning algorithm which is embedded in their proposed hyper heuristic algorithm. Experimental results demonstrate that, their proposed algorithm improved the resources utilization of a data center by 24 % when compared to other conventional algorithms.

Leivadeas et al. (2013) presented hierarchical Efficient Resource Mapping Framework in network based cloud environment. In the first phase, a novel request partitioning approach that facilitates the cost-efficient splitting of user requests among eligible cloud service providers. Further, in the second phase actual mapping of requested virtual to physical resource is performed through intracloud. Authors claimed that their framework is efficient in mapping cloud resources when compared to other state-of-the-art techniques. Table 2.2 summarizes the existing works related to resources allocation and management.

2.3 Allocation of VMs Instances

There are different types of VMs instances (On-demand, Reserved and Spot) which are offered by different cloud providers. Scheduling and managing these instances will play an important role in minimizing total cost in executing BoT or scientific Workflows. There are several approaches in scheduling different VMs instances.

Tang et al. (2014) used Amazon EC2 Spot instances and proposed an optimal randomized bidding strategy through linear programming. Further, for performance analysis, they used Markov process and then compared several check-pointing mechanisms to optimize the job completion time and cost.

Weinman (2015) and Karunakaran and Sundarraj (2015) presented different static and dynamic bidding strategies to fetch more Spot instance from Amazon cloud. Authors focused on different QoS parameters (Reliability, Scalibility and Cost) in executing Workflows and deadline driven business applications.

Poola et al. (2016) proposed a dynamic bidding strategy to minimize the average cost of job completion when there is a deadline constraint. They compared the dynamic bidding strategy with both average and random bidding strategies. Further, Poola et al. (2014) proposed bidding strategy based on multifaceted resource

Table 2.2: Summary of Existing Works on Resource Allocation/Management

Authors	**Methodology**	**Remarks**
Zheng and Wang (2016)	Resource Allocation and Task Scheduling using Pareto based fruit Fly Optimization algorithm (PFOA)	Authors did not consider other heuristic approaches for performance evaluation
Rasti-Barzoki and Hejazi (2015)	Distributed Scheduling and Resource Allocation using Pseudo-Polynomial Dynamic Programming Algorithm	Authors considered supply chain management application. However, no prioritized requests are considered.
Pillai and Rao (2016)	Resource Allocation mechanism for machines in cloud based on the uncertainty principle of game theory	No comparison with other state-of-the-art algorithms
Wang et al. (2010)	Energy-aware Resource Allocation method for workflows executions	Authors did not consider Resource-aware scheduling
Wang et al. (2010)	Load Balancing Min-Min (LBMM) algorithm for the cloud environment	Task's execution time is not considered and leads to bottleneck for task scheduling
Cheng et al. (2017)	Self-adaptive task tuning approach, that automatically searches the optimal configurations for individual tasks running on different nodes.	Performance analysis is done on job completion time
Chunlin et al. (2017)	Optimization policy in resources allocation in both Cloud and Mobile platforms	Only cost QoS parameter is considered for performance analysis
Xu and Li (2013)	Stable matching framework based Resource management for mapping virtual machines to physical servers	Authors did not consider dynamic resources demands and job scheduling

provisioning to estimate the future Spot prices. They discussed fault tolerance techniques namely, migration and job duplication so that there will not be any violation of deadlines.

Yi et al. (2012) discussed how check-pointing and migration mechanisms helps to minimize the cost and volatility of resource provisioning in heterogeneous cloud environment. They also proposed a migration policy in case of instance failure so that monetary cost can be reduced during job completion.

Ribas et al. (2015) presented a Petri net-based Multi Criteria Decision Making (MCDM) framework to assess the cloud services in heterogeneous cloud environment. Their framework also consists of different policies to adapt different cloud resources. Further, they proposed that the business applications (independent jobs) should be executed on Spot instances to minimize the total cost.

Further, we discuss the state-of-the-art approaches related to cloud survey, VMs migration, Workflows, power management, etc. in both homogeneous and heterogeneous cloud environment.

Aceto et al. (2013) discussed the issues related to monitoring of the cloud environment in terms of utilization of cloud resources (PMs, VMs, CPU, RAM, etc.). They also described both commercial and open source cloud platforms and their services. Finally, they discussed the open issues, main research challenges and future directions in the field of cloud monitoring.

Bittencourt et al. (2012)focused on scheduling the Workflows applications in hybrid clouds. They analyzed communication channels and bandwidth for performance evaluation. They concluded that Workflows scheduling in hybrid clouds is easy and efficient when compared to private clouds.

Zheng et al. (2013) listed the issues in live VMs migration in cloud computing environment. Further, they proposed a model with in which VMs can be migrated from one PM to another PM. The model has different options to optimize the migration parameters and users can also customize the migration policies.

Gutierrez-Garcia and Ramirez-Nafarrate (2013) proposed policy-based agents for VMs migration in the cloud data center. Here, there are mainly two agents to monitor the cloud resource utilization (Memory and CPU). Authors also monitored the heterogeneity of the VM and PM during migration along with inputs given by agents.

Baker et al. (2015) presented the lowest energy consumption model between the cloud user and the data center. The authors focused on transferring of big data between the two nodes. As the data size is increased, then proportionate bandwidth requirement is also increased. Hence, they proposed GreeDi, a network-based routing algorithm to find the most energy efficient path to the cloud data centre for processing and storing the big data. Baker et al. (2013) proposed high-end autonomic meta-director framework route to the green data center. They focused on finding the shortest path between source and the data center. Performance analysis is carried out on energy parameter traversing with different paths between the source and data center. Table 2.3 gives the summary of existing works on VMs instances and cost analysis. Further, Table 2.4 summarizes the recent works on multimedia and analytics at the cloud data center.

Table 2.3: Existing Works on VMs Instances and Cost Analysis

Authors	**Methodology**	**Remarks**
Xu et al. (2016b)	On-Demand and Spot instances for in-memory storage workloads	No performance evaluation with state-of-the-art approaches
Díaz et al. (2017)	LLOOVIA (Load Level based Optimization for VIrtual machine Allocation) technique for VMs allocation	Authors did not consider other heuristic approaches for performance evaluation
Arabnejad et al. (2017)	Proportional Deadline Constrained (PDC) and Deadline Constrained Critical Path (DCCP) Workflows scheduling	Task prioritization is considered while scheduling with single QoS parameter (cost).

Table 2.4: Summary of Recent Works on Multimedia and Analytics

Authors	**Methodology**	**Remarks**
Hong et al. (2017)	FairGV: Fair and Fast GPU Virtualization for computing mixed workloads	GPU enhanced the overall performance but in some cases there is a performance degradation due to VMs migration
Ahmad et al. (2017)	Hybrid controller for processing both interactive and batch applications	Resources utilization is efficient but managing non-pre-emptive and prioritized tasks is not increased
Wang et al. (2017)	Framework designed for managing big volumes of data in Multimedia Sensing as a Service (MSaaS) domain	Energy efficient data processing at cloud and edge nodes however management of resources configuration is not dealt
Kryftis et al. (2017)	Network architecture which exploits resource prediction engine for optimal selection of multimedia content provision methods from cloud data center	Prediction of future network traffic demands are effective when compared to real time scenario but other QoS parameters are not considered
Wei et al. (2017)	Cloud based Online Video Transcoding (COVT) system which aims to offer economical and QoS guaranteed solution for online large-volume video transcoding	The system reserves a minimum number of resources needed for the video transcoding and thus efficiently manages the cloud resources. However, SLA constraints are not focused much
Pace et al. (2017)	Scheduling of analytical applications using heuristic approach	The framework designed for scheduling analytical applications for efficient utilization of the resources with better response time. However, other QoS parameters are not considered for cost optimization

2.4 Outcome of Literature Review

After extensive literature review on the existing scheduling, load balancing, resource allocation & management techniques, we identified the following open issues and research gaps for further optimization in terms of different QoS parameters (Reliability, Throughput, Time and Cost) at the cloud data center.

- In most of the existing algorithms on efficient scheduling and load balancing at the cloud data center are implemented in the homogeneous environment. Hence, there is a need for efficient scheduling and load balancing techniques for both homogeneous and heterogeneous cloud environments.

- In most of the existing works on resources allocation and management, authors focused on single parameter optimization (CPU). Hence, we need multi-objective (CPU, RAM, Storage, etc) and dynamic resources allocation and management algorithms at the cloud data center.

- Many authors proposed scheduling and resources allocation & management algorithms using Bio-Inspired approaches. However, these algorithms fall under NP-Hard/NP-Complete complex class and hence, there is a need for hybrid Bio-Inspired algorithms which can optimize the computation time (BoT and Workflows) and thereby efficiently utilizing the cloud resources (CPU and Memory) at the cloud data center.

- Many existing works on VMs instances scheduling are applied only on limited business application with short deadlines. Hence, there is a need for techniques which will optimize several QoS parameters (Reliability, Time and Cost) and thus utilizing the different types of VMs instances at the cloud data center. Further, efficient algorithms in predicting the Spot instances are also needed to efficiently utilize the cloud resources.

- Recently, many authors proposed novel and innovative frameworks for scheduling and managing multimedia applications using machine learning algorithms. Further, they focused more on the allocation of cloud resources to the live streaming of videos using hybrid cloud infrastructure.

- Most of the existing algorithms are simulated in the cloud environment. Hence, there is a need for validating the proposed algorithms in real time cloud setup so that the effective utilization of the algorithms can be monitored.

2.5 Problem Statement

The goal of this research work is to design and develop efficient Bio-Inspired QoS aware algorithms for scheduling, Resources allocation and management at the cloud data center. Accordingly, the research problem is stated as follows.

"To design and develop optimized QoS aware scheduling and load balancing algorithms for efficient resources allocation and management along with the virtual machine migration using Bio-Inspired techniques which ensures the task execution using both On-Demand and Spot instances in cloud data center environment".

2.6 Research Objectives

The objectives of the research work are as follows:

- To design and develop Bio-Inspired QoS aware load balanced scheduling algorithms for cloud data center environment.

- To design and develop novel Bio-Inspired algorithms for resources allocation and management in cloud data center environment.

- To design and develop a cost optimized Bio-Inspired scheduling algorithm for Bag of Tasks (BoT) and Workflows in the cloud data center environment.
- To design and develop a fault tolerant system with VM migration and cost optimization method in a cloud data center environment to guarantee task completion without the violation of SLA using on demand and spot instances.

In order to accomplish the aformentioned research objectives, we designed and developed load balanced scheduling algorithms (Modified Throttled, VM-Assign, Divide and Conquer based Throttled and Modified Particle Swarm Optimization (MPSO)) in a cloud data center. Further, we designed and developed Bio-Inpsired algorithms (MPSO, Modified Cat Swarm Optimization (MCSO) and HYBRID (MPSO+MCSO)) for efficient resource allocation and management. Scheduling of BoT and Workflows are proposed using Grey Wolf Optimizer (GWO) alogrithm on VMs instances (On-Demand and Spot). The proposed algorithms in this research are experimented in both homogeneous and heterogeneous cloud computing environments using Cloud-Analyst and Python based simulators. Finally, the MPSO and HYBRID algorithms are validated using real cloud setup.

2.7 Summary

In this chapter, we presented the existing state-of-the-art techniques related to scheduling, load balancing, resource allocation & management algorithms. Further, we discussed the impact of optimizing the cloud QoS parameters in the efficient utilization of the cloud resources. We clearly listed the challenging issues based on the outcome of literature review along with problem definition and research objectives.

In the following chapters, we discuss the issues and the suitable solutions for optimizing the multi objective QoS parameters in both homogeneous and heterogeneous cloud computing environments.

Chapter 3

Scheduling and Load Balancing at Cloud Data Center

Scheduling with load balancing is one of the critical components for efficient operations in the IaaS based cloud computing environment. In recent years, many clients from all over the world are demanding the various services at a rapid rate. The load balancing algorithms should be very efficient in allocating the requests and also ensuring the usage of the resources in an intelligent way so that underutilization of the resources will not occur in the cloud environment. Cloud consists of enormous resources in terms of VMs and Servers, these resources are dynamically configured either physically or virtually. In this chapter, we proposed three different scheduling algorithms for allocating the clients' requests and thus balanced the load on the available VMs. The research contributions towards efficient Scheduling and Load Balancing in a cloud data center are as follows.

- To design and develop an efficient scheduling and load balancing technique using a Modified Throttled algorithm.
- To design and develop an efficient scheduling and load balancing technique using VM-Assign algorithm.
- To design and develop an efficient scheduling and load balancing technique using Hybrid (Divide and Conquer based Throttled) algorithm.

The proposed algorithms follow a systematic approach to schedule the incoming tasks and efficiently distribute the workload on the available resources of the cloud data center. The details of the proposed algorithms are given in the following sections.

3.1 Proposed Modified Throttled Algorithm

Many scheduling and load balancing algorithms based on optimizing different QoS parameters are proposed by different researchers (Radojević and Žagar (2011)). This research contribution mainly focuses on the optimum allocation of tasks on available cloud resources (VMs) using the CloudAnalyst tool and our proposed Modified Throttled algorithm achieves less average response time when compared to Throttled and Round Robin algorithms.

3.1.1 Basics of Throttled Algorithm

In Throttled algorithm, the load balancer maintains an index, table of virtual machines as well as their states (Available or Busy). The client/server first makes a request to the data center to find a suitable virtual machine (VM) to perform the recommended job. The data center queries the load balancer for allocation of the VM. The load balancer scans the index table from the top until the first available VM is found or the index table is scanned fully. If the VM is found, then the VM id is sent to the data center. The data center communicates the request to the VM identified by the id. Further, the data center acknowledges the load balancer of the new allocation and then the data center revises the index table accordingly. While processing the request of the client, if appropriate VM is not found, then the load balancer returns =1 to the data center (Wickremasinghe et al. (2010)). When the VM completes the due task, then a request is acknowledged by the data center.

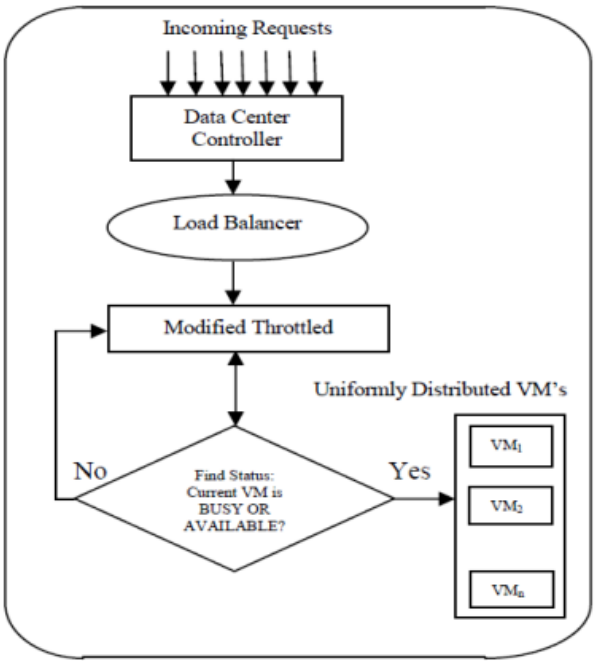


Figure 3.1: Flow of the Proposed Modified Throttled Algorithm

3.1.2 Proposed Methodology

The proposed modified throttled algorithm as shown in Figure 3.1 mainly focuses on the scheduling the incoming tasks on the available VMs in the cloud data center environment

Here, data center receives the clients' tasks and the interconnected load balancer based on our proposed algorithm schedules the tasks on the available cloud resources. Our proposed algorithm uniformly distributes the load on the available VMs and thus maintains the stable load on the VMs. Algorithm 3.1 shows the working principle of the proposed Modified Throttled Load Balancer.

Algorithm 3.1 Modified Throttled Load Balancer

0: *Input*: Number of incoming jobs: x_1, x_2, x_3,..., x_n
Available VMs: vm_0, vm_1, vm_2,..., vm_k
1: Output: All incoming jobs x_1, x_2, x_3,..., x_n are allocated one by one to the available vm_0, vm_1, vm_2,..., vm_k
2: Load balancer maintains an index table of VMs and its state (BUSY/AVAILABLE). Initially, all VMs are available.
3: Data Center Controller receives a new request.
4: Data Center Controller queries the Modified Throttled Load Balancer for the next task allocation.
5: Load Balancer starts with the VM at first index, then checks for the availability of the VM.
6: **Case 1: If VM is in "AVAILABLE" state**
7: The load balancer returns the VM id to the Data Center Controller
8: The data center controller sends the request to the VM.
9: The data center controller notifies the load balancer for the new allocation.
10: Then the load balancer updates the allocation table accordingly.
11: **Case 2: If VM is in "BUSY" state**
12: The Modified Throttled Load Balancer returns -1.
13: When the VM completes the execution, then the data center controller notifies the load balancer stating that this VM is AVAILABLE.
14: If there are more requests, Data Center Controller repeats step 4 with next index and the process is repeated until the size of the index table is reached. After reaching the size of index table, parsing starts with the first index.
15: Goto Step 2.

Our proposed Modified Throttled algorithm maintains an index, table of virtual machines and also the state of VMs similar to the Throttled algorithm. We have made an attempt to improve the response time as well to achieve efficient usage of available virtual machines. The proposed algorithm employs a method of selecting a VM for processing the clients' request where a VM at first index is initially selected depending upon the state of the VM. If the VM is available, then it is assigned to the request and id of the VM is returned to data center, else -1 is returned. When the next request arrives, then the VM at index next to previously assigned VM is chosen depending on the status of the VM. But, in the Throttled algorithm, the

index table is parsed from the first index every time when the data center receives the request from the client. Algorithm 3.1 describes the complete working principle of Modified Throttled load balancer. The proposed algorithm marginally improves the cloud resource utilization when compared to Throttled algorithm. The details of the second proposed algorithm are as follows.

3.2 Proposed VM-Assign Algorithm

As we know, the cloud is comprised of many resources in terms of PMs, VMs etc. and efficient utilization of these resources needs an intelligent algorithm. In this research contribution, we mainly deal with resource utilization and the focus is on efficient utilization of VMs which are allocated to perform the clients' task. Our proposed VM-Assign algorithm improves the resource utilization when compared to Active VM load balancer algorithm. The details of the proposed algorithm are as follows.

3.2.1 Basics of Active VM Algorithm

Mahalle et al. (2013) developed Active monitoring load balancer algorithm which maintains information about each VMs and the number of tasks currently allocated to each of the VMs. When a new task comes, then it identifies the least loaded VM from the available cluster. If there are more than one, then the first identified will be selected for the task allocation. Active VM load balancer returns the VM id to the data center controller, which allocates the new task for execution. Data center controller notifies the Active VM load balancer of the new task allocation.

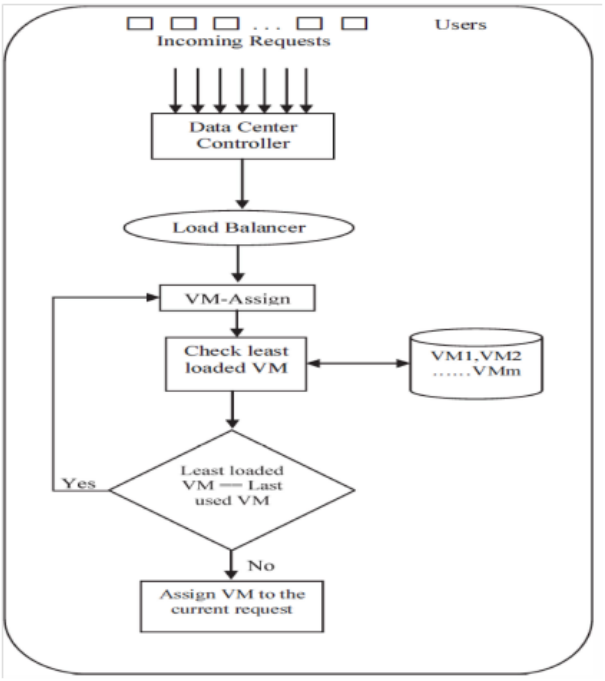


Figure 3.2: Flow of the Proposed VM-Assign Algorithm

3.2.2 Proposed Methodology

The proposed VM-Assign algorithm mainly focuses on finding out the least loaded virtual machine and allocating the incoming tasks to the VMs in an efficient manner. This algorithm also ensures the assigning of the incoming tasks in a balanced way. Figure 3.2 shows the methodology of our proposed VM-Assign algorithm.

The proposed algorithm employs a method for selecting a VM for processing clients' task where, a VM at first index is initially selected depending upon the state of the VM. If the VM is in "AVAILABLE" state, then this VM is assigned to the task and its id is returned to the data center, else -1 is returned. When the next task arrives, then the VM at index next to previously assigned VM is chosen depending on the state of the VM and follows the above step. Further, for the upcoming tasks, the VM is chosen depending on its load. Here, the algorithm mainly focuses on the least loaded VM and further ensures that the assigned VM in the current iteration has not participated in the last iteration. Algorithm 3.2 shows the proposed VM-Assign Load

Balancing algorithm.

Algorithm 3.2 VM-Assign Load Balancer

0: *Input*: Number of incoming jobs: x_1, x_2, x_3,..., x_n
Available VMs: vm_0, vm_1, vm_2,..., vm_k
1: Output: All incoming jobs x_1, x_2, x_3,..., x_n are allocated one by one to the available vm_0, vm_1, vm_2,..., vm_k
2: Initially, all the VMs have 0 allocations.
3: VM-assign load balancer maintains the index / assign table of VMs which has number of requests currently allocated to each VM.
4: When requests arrive at the data center then it passes to the load balancer.
5: Index table is parsed and least loaded VM is selected for execution.

6: **Case 1: If VM is Found**
7: Checks whether the chosen least loaded VM is used in the last iteration.
8: **If YES**
9: Go to Step 4 to find the next least VM.
10: **If NO**
11: Least loaded VM is chosen.
12: VM-assign load balancer returns the VM id to the data center.
13: Task is assigned to the VM for execution. Data center controller acknowledges the load balancer.
14: VM-assign load balancer updates the tasks held by each VM.
15: When the VM finishes the processing the task, data center receives the response.

16: Data center controller notifies the VM-assign load balancer for the VM de-allocation and accordingly updates the table.
17: Repeat the process from Step 2 for the execution of next task.

The proposed algorithm achieves better VM utilization and maintains a balanced load when compared to Active VM load balancer algorithm. The details of the third proposed algorithm are as follows.

3.3 Proposed Hybrid (Divide and Conquer Based Throttled) Algorithm

In this research contribution, we applied the hybrid scheduling approach in IaaS based cloud for efficient usage of resources. Here, we combined the Divide and Conquer and Throttled approaches and came up with a new hybrid scheduling algorithm referred to as Divide and Conquer Based Throttled (DCBT) algorithm. The hybrid DCBT algorithm efficiently utilizes the cloud resources and it also has a better execution time when compared to Throttled algorithm.

3.3.1 Basics of Divide and Conquer Algorithm

Divide and Conquer approach is an algorithm design paradigm based on multi branch recursion procedure. It follows top down approach to make a task simpler and then bottom up approach to get a final solution of a task. In divide and conquer approach, the given task is divided into smaller sub-tasks and then each task is solved independently. When we keep on dividing the sub tasks into even smaller sub-tasks, we may eventually reach a stage where no more division of task is possible and then each small sub task is solved. Further, the solutions of all sub tasks are merged together to get a final solution of the original task from which it is initiated.

3.3.2 Proposed Methodology

The proposed hybrid scheduling and load balancing algorithm combines the methodology of Divide and Conquer and Throttled algorithms referred to as DCBT. The DCBT algorithm plays an important role in distributing the incoming load in an efficient manner so that it maximizes the resource utilization in a cloud environment. The main aim of the proposed DCBT is to reduce the total execution time of the tasks and thereby maximizing the resource utilization. Further, the proposed DCBT algorithm is analyzed using CloudSim simulator and also in a customized distributed

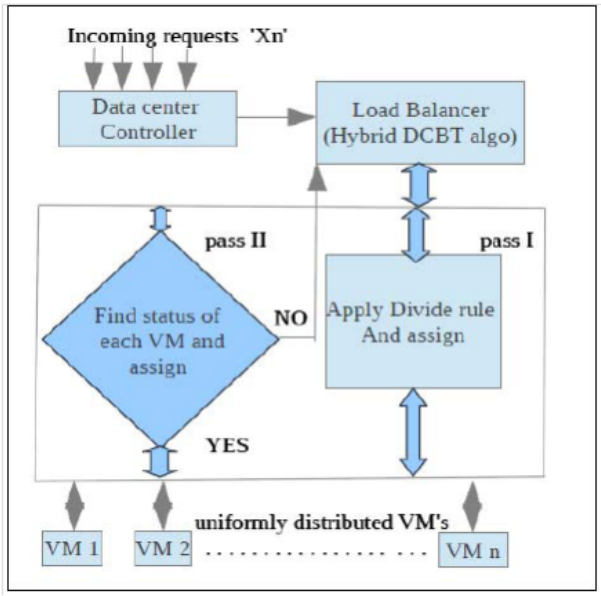


Figure 3.3: Flow of the Proposed DCBT Algorithm

environment using Python. Figure 3.3 shows the flow diagram of our proposed DCBT algorithm.

The proposed DCBT algorithm is implemented in two scenarios. (i) In the CloudSim simulation environment and (ii) in the customized environment written in Python which resembles the cloud simulation environment. Results of both platforms are same and encouraging. The details of the platforms and algorithm design are as follows.

Figure 3.3 shows the complete flow of the proposed DCBT algorithm based on the combined approaches of different phases namely Pass I and Pass II. Here, Pass I implements the Divide and Conquer approach and Pass II implements the Throttled algorithm approach. Further, our proposed DCBT algorithm takes care of priority tasks, i.e. if the data center comes across any prioritized task then it intimates our DCBT algorithm for handling it. Further, our proposed DCBT algorithm ensures that the prioritized task will be executed at the earliest so that it will not lead to starvation of prioritized tasks. It means that while executing the tasks, if the DCBT algorithm finds any prioritized task, then it puts that task in the next available

slot for execution. Thus, the behavior of proposed DCBT algorithm will remain the same for both the platforms. But, in CloudSim environment tasks are assigned to VMs whereas in a customized environment tasks are assigned to Request Handlers (RH). Hence, the usage of VMs and RH are used interchangeably in this research contribution. Algorithm 3.3 shows the proposed DCBT scheduling algorithm with two phases, namely: Pass I and Pass II and the details are as follows.

Steps for Pass I Phase of DCBT Algorithm:

Step 1: Find the number of tasks in queue for a period 't'.

Step 2: Find the number of available RH in a real time distributed system.

Step 3: Tasks are divided continuously by the number of available RH in a real time distributed system.

Step 4: After the division, check the remainder == number of RH then, assign tasks to the RH for execution

Step 5: Steps 1 to 4 are iterated continuously in coordination with pass II until it finishes the execution.

Steps for Pass II Phase of DCBT Algorithm:

Step 1: Initially Load Balancer assigns the tasks to the available Request Handlers or Virtual Machines.

Step 2: For the next assignment, Load Balancer checks for the availability of RH.

Step 3: Next task is allocated to available RH or VM, iff

1. RH or VM should be free.
2. Assigned RH or VM should not be used in the previous assignment.
3. If any prioritized requests, then put them at first location in the queue.

Step 4: Steps 1 to 3 are repeated in coordination with Pass I until all the tasks are executed completely.

Algorithm 3.3 Hybrid (DCBT) Task Scheduler for Pass I and II Phases

```
1: for all total tasks
2:         for all available RH from servers
3:                 divide tasks by RH
4:                 assign tasks to available RH
5:         end for
6: end for
7: return RH for executing the task

   Pass II
8: for all total tasks
9:         find the available RH from servers
10:        if (RH (i )or VM ≤ RH (i+1)) and so on
11:                assign RH i
12:        else
13:                assign RH i+1
14: end for
15:        if(RH or VM is used in previous iteration)
16:                Search next RH or VM
17:        else
18:                Assign current RH or VM
19: return RH for executing the task
```

The proposed algorithm achieves better execution time in both platforms when compared to a Modified Throttled algorithm and also it efficiently utilizes the cloud resources i.e. it maintains the balanced load on the VMs. Next, we will discuss about the performance analysis of our proposed algorithms.

3.4 Performance Evaluation

In this chapter, we proposed three load balancing algorithms (Modified Throttled, VM-Assign, and Divide and Conquer Based Throttled). The proposed algorithms are compared with other state-of-the-art algorithms and analysed with different QoS parameters. Next, we will discuss the experimental setup, results and analysis.

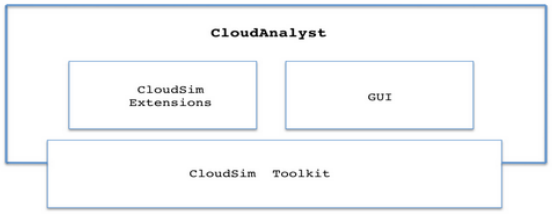


Figure 3.4: Architecture of CloudAnalyst simulator

3.4.1 Experimental Setup

The proposed algorithms are experimented in CloudAnalyst simulation tool which inherits the properties of CloudSim tool. The tool completely simulates the IaaS based cloud computing environment and it is designed by University of Melbourne, Australia (Wickremasinghe et al. (2010)). The CloudAnalyst simulator gives the real time scenario with six different geographical locations, i.e. a number of users from particular locations can be identified by depending on the specific application, e.g. Facebook users from Asia, Africa, etc. The simulator is very flexible and it provides data centers, virtual machines, bandwidth and many more QoS parameters for experimentation. Figure 3.4 shows the snapshot of the CloudAnalyst architecture.

Hypothetical applications like Facebook users, Twitter users, Internet users are considered for experimentation. Six different geographical locations (six different continents of the world) are considered (Mondal et al. (2012)). A single time zone is considered for all user locations. For simplicity, one hundredth of the total users from each continent is considered and it is assumed that only 5% of total users are online during peak hours and during off-peak hours, users are one tenth of the peak hours. For experimentation, internet users from six different continents are considered, i.e. six user bases and peak and non-peak users are given in Table 3.1.

We considered internet users from different continents from the month of June 2012 (Users (url)). The same data is experimented with three different scheduling algorithms and response time of each algorithm is considered for performance analysis. Each data center has a capacity to host a number of virtual machines which are needed

Table 3.1: Workload Setup for Scheduling

User Base	**Region**	**Online Users during peak hours**	**Online Users during non-peak hours**
North America	0	135000	13500
South America	1	125000	12500
Europe	2	255000	25500
Asia	3	535000	53500
Africa	4	30000	3000
Oceania	5	10000	1000

for a particular application. Machines have 100 GB of storage space, 4 GB of RAM, each machine has 4 CPU and a power of 10k MIPS.

For customized setup, we used four nodes having different configurations. The four nodes are connected together in which one node acts as a load balancer, another node act as a dynamic task generator and remaining two nodes act as servers which have many VMs. Here, the task consists of Million Instructions Per Second (MIPS) which is generated by the client generator.

3.4.2 Results and Analysis

The proposed three algorithms (Modified Throttled, VM-Assign, and Divide and Conquer Based Throttled) are compared with state-of-the-art algorithms and evaluated for QoS parameters (Reliability and Time). The proposed algorithms outperform in efficient utilization of cloud resources, mainly VMs by maintaining the balanced load in the data center. The proposed algorithms are efficient in achieving better average response time and execution time. The detailed analysis of the proposed algorithms is as follows.

Table 3.2: Utilization of VMs

Sl. No.	**Throttled**	**Round Robin**	**Modified Throttled**
VM1	1182	254	254
VM2	76	254	254
VM3	8	253	254
VM4	2	253	253
VM5	0	254	253

Results of Modified Throttled Algorithm:

The proposed Modified Throttled algorithm mainly focuses on both QoS parameters (Reliability and Time), i.e. efficient utilization of VMs and average response time. The details are as follows.

Efficient Utilization of VMs

The proposed Modified Throttled algorithm will not parse the index table from the beginning every time. But this is not the case with Throttled algorithm in which the algorithm always parses the index table from the beginning irrespective of a number of tasks. In Throttled algorithm, a few VMs are overloaded and remaining VMs are underutilized and thus resulting in load imbalance on the available VMs. But, the proposed Modified Throttled algorithm overcomes the problem with the parsing the index table and hence the VMs are utilized efficiently. Both the algorithms are compared using five VMs and Table 3.2 gives the information about how many times each VM is efficiently utilized.

From Table 3.2, it is observed that Round Robin and Modified Throttled algorithms efficiently utilize the available VMs compared to Throttled algorithm. However, the results of Round Robin and Modified Throttled algorithms seem to be same, but in the Round Robin, the state of the VM (BUSY/AVAILABLE) is not considered

Table 3.3: Average Response Time Analysis

Algorithms	**Average Response Time**
Throttled	364.76ms
Round Robin	363.52ms
Modified Throtled	362.67ms

and hence leads to the queuing of the incoming tasks on the server. The modified Throttled algorithm gives better results by checking the state of the VM and thus avoiding the queuing at the server. Even though Round Robin and Modified Throttled algorithms give the same result in terms of VMs utilization but the proposed Modified Throttled algorithm has better average response time when compared to other benchmark algorithms.

Average Response Time Analysis

The proposed Modified Throttled algorithm along with the other two benchmark algorithms are experimented for average response time analysis. The experiment is repeated several times and average response time of Modified Throttled algorithm is slightly better when compared to Round Robin and Throttled algorithms. Table 3.3 gives the information about the average response time of all three algorithms.

Results of VM-Assign Algorithm:

The proposed VM-Assign algorithm mainly focuses on Reliability QoS parameter, i.e. efficient utilization cloud resources (VMs).

Efficient Utilization of VMs

The proposed VM-Assign algorithm will not allow the same VM to be assigned to the new task if it was allocated in its previous step. But this is not the same case with Active-VM load balancing algorithm in which it assigns the least loaded VM

Table 3.4: VMs Usage with 5 VMs

Sl. No.	**Active-VM**	**VM-Assign**
VM1	1178	258
VM2	78	252
VM3	10	253
VM4	4	251
VM5	2	254

depending on the current load. Active-VM load balancer algorithm will not bother about VM's usage. Hence, in Active-VM load balancer algorithm, a few VMs are overloaded with many tasks to be processed and remaining VMs will handle only a few tasks. Thus, resulting in underutilization and overutilization of the cloud resources and this will lead to load imbalance in the cloud data center. But our proposed VM-Assign load balance algorithm utilizes all available VMs in an efficient manner. The proposed algorithm proves that there is no underutilization or overutilization of the VMs in the cloud data center. The algorithm is initially tested with five VMs and later 25 VMs, the comparative analysis is carried out for proposed VM-Assign and Active-VM load balancer algorithms. In both cases, our proposed VM-Assign algorithm balances the load on all available VMs in an efficient way. Hence we can say that our proposed VM-Assign algorithm will overcome the under- or over-utilization of resources in the cloud data center. Table 3.4 gives the information about how many times each VM is efficiently used.

From the Table 3.4, we can observe that VM-Assign load balancer algorithm distributes the incoming tasks to all VMs in an intelligent way as compared to Active-VM load balancer algorithm. In the proposed VM-Assign algorithm, utilization of the resources is neither underutilized nor overutilized, whereas in Active-VM load balancer, VM0 is overutilized and VM4 is used only twice and other VMs are underutilized. The experiment is repeated for different number of VMs like 25, 50, and 100. Next

we will analyze the tasks held by 25 VMs using proposed VM-Assign and Active-VM load balancer algorithms. Table 3.5 shows the usage of 25 VMs.

From Tables 3.4 and 3.5, it is clearly observed that Active-VM load balancing algorithm is overutilizing the initial VMs and underutilizing the later ones. But our proposed VM-Assign algorithm distributes the incoming tasks to all VMs in an intelligent way and hence all the resources are efficiently used. Response time analysis of both algorithms is also experimented but there is no significant difference in the response time of both algorithms and hence we considered only VMs usage. Further, the results for 50 and 100 virtual machines follow the same pattern of allocation as shown in Tables 3.4 and 3.5 respectively.

Results of DCBT Algorithm:

The proposed DCBT algorithm mainly focuses on both QoS parameters (Reliability and Time), i.e. efficient utilization of VMs and execution time. The details of efficient utilization of VMs and execution time are as follows.

Efficient Utilization of VMs

The proposed DCBT algorithm combines the two different approaches and then schedules the incoming tasks on the available VMs in IaaS based cloud environment. Initially, the tasks from the queue are taken and divided into smaller groups which follows divide and conquer based approach and then assigned to the VMs. Further, it follows the Modified Throttled approach to choose the suitable VM for executing the upcoming task. Initially DCBT assigns the tasks to the available VMs and make them busy. Later, the tasks are executed on VMs and load on any VM is neither overloaded nor underloaded. Hence, our proposed DCBT algorithm efficiently utilizes the available VMs in the given cloud environment. Table 3.6. shows the number of tasks handled by different configured VMs.

From Table 3.6, it is clearly observed that the proposed DCBT algorithm intelligently distributes the incoming tasks on the available VMs and maintains the balanced

Table 3.5: VMs Usage with 25 VMs

Sl. No.	**Active-VM**	**VM-Assign**
VM1	18609	3693
VM2	20078	4376
VM3	6569	4330
VM4	5845	4312
VM5	4973	4350
VM6	4568	4318
VM7	4164	3818
VM8	4907	4588
VM9	4337	4565
VM10	3886	4610
VM11	3267	4615
VM12	3321	4582
VM13	2966	4065
VM14	3869	4890
VM15	2988	4863
VM16	2948	4828
VM17	2332	4852
VM18	2425	4971
VM19	2047	5053
VM20	2467	4440
VM21	2249	5098
VM22	2564	5278
VM23	1883	5309
VM24	1984	5472
VM25	1648	5618

Table 3.6: Number of Tasks Executed by VMs

Sl. No.	**Throttled Algorithm**	**DCBT Algorithm**
VM1	1058	253
VM2	158	253
VM3	34	253
VM4	21	253
VM5	0	253

load. But the Throttled algorithm overutilizes the initial VMs and underutilizes the later VMs. From Table 3.6, we can clearly observe that VM5 is never assigned any task by Throttled algorithm where as tasks are assigned to VM4 by proposed DCBT algorithm. Similarly, VM1 is used many times by Throttled algorithm, but that is not the same case with our proposed DCBT algorithm. The number of tasks coming at a batch of ten are processed depending on the VMs availability. The experiment is repeated for different number of incoming tasks and VMs. Thus, our proposed DCBT algorithm distributes the load equally and outperforms the Throttled algorithm in all cases.

Execution Time Analysis

The proposed DCBT algorithm is experimented for execution time analysis and compared with Throttled algorithm. The experiment is repeated for different sets of VMs and in all cases, our proposed DCBT algorithm has better execution time when compared to Throttled algorithm. Table 3.7 shows the execution time analysis of both Throttled and DCBT algorithms in seconds.

For the experimentation we considered the both non-prioritized and prioritized requests. The proposed DCBT algorithm will execute the prioritized requests within its deadline and in parallel it also manages the non-prioritized tasks in an efficient

Table 3.7: Execution Time Analysis (in seconds)

Number of Tasks	**Throttled Algorithm**	**DCBT Algorithm**
20	131	118
100	645	589
1000	6333	57696

manner. In our experimentation, we also considered different BoT with varying inter arrival time between them. It is observed from Table 3.7 that our proposed DCBT algorithm improves the efficiency of execution time by 9.972%.

3.5 Summary

In this chapter, we proposed three different scheduling algorithms in the IaaS based cloud environment. The proposed algorithms outperform benchmark algorithms in Reliability and Time QoS parameters such as average response time, total execution time and efficient utilization of cloud resources (VMs). For the experimentation we used different sets of inputs of VMs and Bag of Tasks (BoT) and in this chapter we used only independent tasks.

In the proposed DCBT algorithm we used hybrid approach and we considered both prioritized and non-prioritized tasks. Here, we concentrated on VMs usage as resource utilization but other resources such as Memory and CPU are not considered in this chapter. But hot plugging technique of assigning needed resources, motivated us to consider these resources during scheduling in the cloud environment. Hence, in the next chapter we discuss in detail about resource allocation management strategies using Bio-Inspired techniques.

Chapter 4

Resource Allocation and Management at Cloud Data Center

Resource allocation and management in the cloud environment plays a major role in the IaaS based cloud environment. Since the clients' requests are received by the cloud data centers at a rapid rate, hence the enormous amount of data is bundled together at the cloud data centers. In the previous chapter, we considered only VMs as a cloud resource, but many clients' demand for cloud resources in terms of Memory and Central Processing Unit (CPU). Intelligent handling and assignment of these resources need a better scheduling and resource allocation approaches. Even though many researchers contributed towards scheduling and resource allocation approaches, but still there is a need for optimization as these approaches come under NP-Hard/NP-Complete complexity classes. Hence, to solve these problems in this chapter, we proposed Bio-Inspired algorithms for scheduling and resource allocation as these approaches are best suited for solving the NP-Hard/NP-Complete complex problems. The research contributions towards efficient Bio-Inspired scheduling, resource allocation, and management are as follows.

- To design and develop an efficient Bio-Inspired algorithm for scheduling the tasks using Modified Particle Swarm Optimization (MPSO) technique.

- To design and develop an efficient dynamic resource allocation and management policy using both MPSO and Modified Cat Swarm Optimization (MCSO) techniques.
- To design and develop an efficient HYBRID (MPSO+MCSO) algorithm for managing the cloud resources.
- To evaluate the performance of proposed HYBRID approach with benchmark exact algorithm along with statistical hypothesis analysis.

The proposed Bio-Inspired algorithms follow the standard approaches of cloud computing for scheduling, resource allocation and management. The proposed algorithms are compared with benchmark algorithms for performance evaluation. Further, the results of the proposed algorithms are compared with statistical hypothesis analysis and time complexity of HYBRID (MPSO+MCSO) algorithm is also analyzed. Our approach follows the process of hot-plugging in which the resource units such as Memory and CPU will be added and removed from the VMs without interrupting the current state of the VMs. The details of the proposed Bio-Inspired algorithms are given in the following sections.

4.1 Basics of MPSO Algorithm

Particle Swarm Optimization (PSO) was developed by (Eberhart and Kennedy (1995)) 1995 and it is widely used stochastic optimization technique based on the behavior of animals and birds. In MPSO, the particle is represented by its position and velocity; these particles keep track of local best (LB) and global best (GB) values; fitness function determines the LB and GB values. In the scheduling approach, particles refer to VMs; LB refers to underloaded VM from each cluster and GB refers to the minimum value among all LB values. The algorithm iterates continuously to get the new LB and GB values. In MPSO, GB does not remain the same in each iteration

when compared to PSO. The position and velocity of a particle are updated based on the following Equations 4.1.1 and 4.1.2 respectively (Liu and Wang (2012)).

$$x(t+1) = x(t) + v(t) \tag{4.1.1}$$

$$x(t+1) = v(t) + c_1 r_1(LB - x(t)) + c_2 r_2(GB - x(t)) \tag{4.1.2}$$

where,

x(t) is current position of particle/Current load of a VM.

LB is least under loaded VM from cluster.

GB is least under loaded VM from all LB values.

c_1 and c_2 acceleration coefficients, usually $c_1 = c_2 = 2$.

r_1 and r_2 are random numbers between (0,1).

4.2 Basics of MCSO Algorithm

The basic Cat Swarm Optimization (CSO) technique (Bilgaiyan et al. (2014)) deals with two important phases like seeking mode and tracing mode. In the proposed Modified Cat Swarm Optimization (MCSO), we mainly concentrate on seeking mode rather than on tracing mode as it is similar to the MPSO approach. MCSO is used for resource allocation management based on seeking mode only. However, MCSO cannot be efficiently used for scheduling if it uses both seeking and tracing modes. MCSO deals with four different types of memories like Seeking Memory Pool (SMP), Seeking Range of selected Dimension (SRD), Counts to Dimension Change (CDC) and Self Position Consideration (SPC). These memory pools play an important role in assigning the cloud resources to VMs in the current work. The usage details of these memory pools are given in the following Sections along with the proposed MCSO algorithm. The main difference between MPSO and MCSO is that the seeking mode

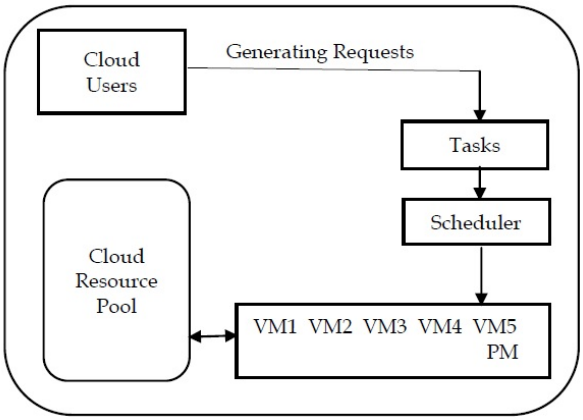


Figure 4.1: Model for Explaining Our Proposed Work

of MCSO overcomes the limitations of the MPSO by comparing the future resource demands with excess resources present in the VMs.

4.3 Model for Explaining our Proposed Algorithms

Figure 4.1 shows an example model where the tasks are scheduled on the VMs and resources such as CPU and Memory are utilized efficiently by our proposed MPSO, MCSO and HYBRID (MPSO+MCSO) algorithms. Here, we used different types of VMs on a PM which can communicate with the scheduler for the efficient functioning of proposed algorithms. The tasks are coming at a batch of ten and their inter arrival time remains constant. Initially, tasks are scheduled efficiently on VMs and then the allocation of required resources will take place.

The incoming task demands n number of cloud resources where n = *[0,9]* and allocation of n resources are provided by either cloud resource pool or by the VMs. Each cluster contains different VMs with the best two n resource values represented as *excess_res1* and *excess_res2*, respectively and the remaining resources available with VMs are stored in *excess_res3[]*. During resource allocation, the on demand resources are compared with *excess_res1*, *excess_res2* and *excess_res3[]*, respectively by our proposed MPSO, MCSO and HYBRID (MPSO+MCSO) algorithms. The

excess_res1 and *excess_res2* mappings are used by the MPSO algorithm where as *excess_res3[]* mapping is used by the MCSO algorithm. On the other hand, HYBRID (MPSO+MCSO) algorithm uses all the three aforementioned resource mappings. The details of the proposed MPSO based scheduling algorithm are as follows.

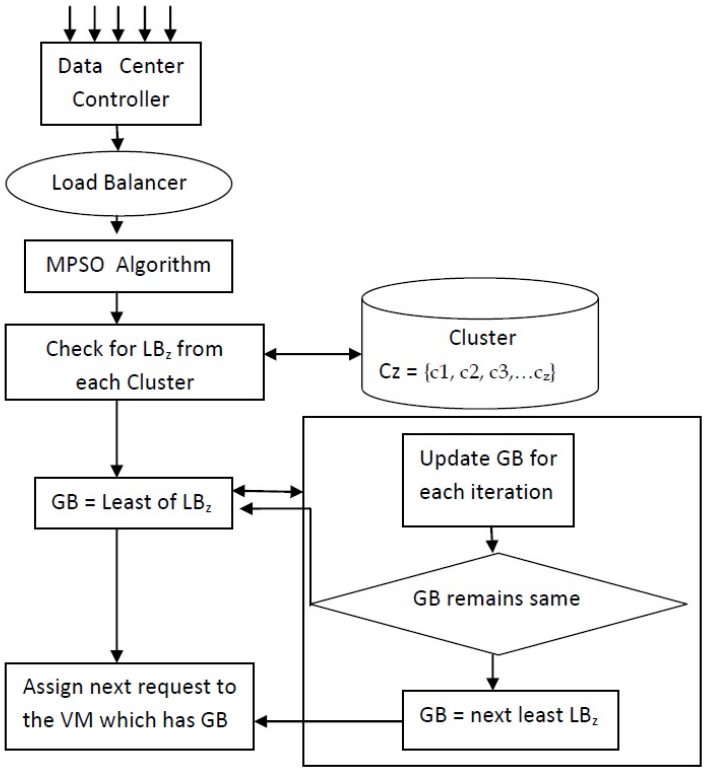


Figure 4.2: Flow Diagram of MPSO based Scheduler

4.4 Proposed MPSO Algorithm for Scheduling

Since the cloud receives tasks at a rapid rate from the outside world, hence assigning and executing these tasks is a challenging issue. In this thesis contribution, we propose a MPSO technique for scheduling the incoming tasks against available VMs. Figure 4.2 shows the flow diagram of the proposed MPSO based scheduling algorithm.

Number of incoming tasks i.e. $\{x_1, x_2, x_3, ..., x_n\}$ are to be scheduled on VMs i.e. $\{vm_0, vm_1, vm_2, ..., vm_k\}$. Here, our proposed MPSO algorithm is deployed for scheduling the tasks in a balanced way. The MPSO algorithm plays an important role in assigning the incoming tasks to the VMs as efficiently as possible. We experimented this work for a private cloud which receives the tasks in a batch of ten (can be extended) and these tasks are assigned to the VMs. Clustering depends on the number of VMs taken for experimentation. Algorithm 4.1 gives the complete details of task scheduling by using MPSO technique.

In every iteration, each cluster will identify the least loaded VMs referred to as the local best (LB_z) and the smallest among these VMs referred to as global best (GB). The next task is allocated to the VM that is associated with GB. If the GB remains the same in the subsequent iteration, then GB is updated with second least LB_z from the cluster list C_z. The same process is continued until all the tasks are executed. The time complexity of MPSO algorithm is *0(n.z)*. Since z is a constant hence the time complexity of MPSO algorithm will be *0(n)* in polynomial time.

Algorithm 4.1 Task Scheduling using MPSO Technique

0: *Initialization*: $\{vm_0, vm_1, vm_2, ..., vm_k\}$ count = 0
Local Best (LB_z) = 0
Global Best (GB) = 0
VMs = $\{vm_0, vm_1, vm_2, ..., vm_k\}$
Clusters, $C_z = \{c_1, c_2, c_3, ..., c_z\}$
Cluster size = k/C_z
1: *for* all incoming requests $\{x_1, x_2, x_3, ..., x_n\}$
2: each cluster C_z = least loaded VM
3: Assign each one of them as LB_z from C_z
4: *end for*
5: Assign GB = least LB_z
6: Next task allocated to VM which contains GB
7: *if* (Next allocation == last used GB) then skip
8: goto Step 2 for next least LB_z
9: *else*
10: goto Step 6

4.5 Proposed MPSO Algorithm for Resource Allocation and Management

In this thesis contribution, the MPSO algorithm is designed for allocating and managing the clients' resource demands. Here, the tasks demand for dynamic resources at a rapid rate in the cloud environment and satisfying these demands is a challenging task. To execute the clients' demand, the VMs must have enough resources and providing these resources to VMs is managed by the proposed MPSO. For more clarification, let us assume that there is a cloud resource pool (Res_pool) which provides the resources demanded by the tasks and Res_pool acts like a resource repository. Each of the VMs has at least two minimum resources (CPU and Memory) for executing a task. In the first iteration, for all VMs, resources are given by cloud resource pool and from the second iteration on-wards it depends on the resources as demanded by the tasks. Each of the VMs will not use all the resources for execution of tasks and these unused excess resources which remain with the VMs can be utilized for the future task demands. Our proposed MPSO algorithm plays an important role in assigning these unused resources either from the VMs or from the cloud resource pool.

For resource allocation and management strategy, the tasks are coming from the clients side with the same inter arrival time. We considered a queue size of ten tasks and each task demands cloud resources in random fashion. Allocation and management of these resources is carried out by a load balancer based on our proposed algorithms. In this work, we assumed CPU and Memory as mandatory resources for task's execution and each of the VMs needs at least two resources (i.e. CPU and Memory) for execution of a task. In the first iteration, resources are taken from the cloud resource pool which acts as the global hub of resources.

To assign the demanded resources to VMs with better efficiency, let us form the clusters $\{c_1, c_2, c_3,..., c_z\}$ from the VMs $\{vm_0, vm_1, vm_2,..., vm_k\}$ which operate in both sequential and parallel modes. For parallel mode execution, we need at least two

clusters. For subsequent assignment of resources to the VMs, let us take the best two excess resources, i.e. *excess_res1* and *excess_res2* (VMs with unused extra resources) from each cluster and check for a match with the next resource demands. If the next resource demand is matching with the best values (unused resources), then VMs start executing the task immediately, or else the resources are taken from the resource pool for the task's execution.

After each iteration, if there are any unmatched/unused resources, then these resources can be released to the resource pool. Once VMs complete the execution of a task, then the minimum resources available with VMs can also be released to the cloud resource pool. The main focus is on utilizing the resources from the VMs rather than lending the resources from the resource pool. In doing so, we dynamically exploit the resources and thus communication overhead between the cloud resource pool and VM is reduced significantly. Algorithm 4.2 describes the resource allocation and management using the proposed MPSO technique.

Usually in MPSO algorithm, the local and global best may result in minimum or maximum values but these are application specific. In the Algorithm 4.1, we took only one best (minimum: load on the VM) value from each cluster; but in the Algorithm 4.2, our main objective is to choose the best (maximum) two values and thus resulting in more optimization if it matches the subsequent resource demand for the tasks. Thus the MPSO algorithm is designed accordingly to suit the requirements of scheduling and resource allocation and management approaches.

Even though our proposed MPSO algorithm improves the allocation of resources intelligently, but it has the following limitations.

- *excess_res1 and excess_res2 may not match exactly with the subsequent future resource demand for task's execution.*
- *More communication overhead between VMs and cloud resource pool.*

Algorithm 4.2 Resource Allocation and Management Using MPSO Technique

0: *Initialization*: Res_pool, Needed_res, min_res=2
sum_res= 0
VMs = {vm_0, vm_1, vm_2,...,vm_k}
Clusters, C_z = {c_1, c_2, c_3,..., c_z}
Cluster size = k/C_z
1: *for* all incoming requests {x_1, x_2, x_3,..., x_n}
2: Res_demand ← sum of resources from requests
3: Needed_res ← resource demand for each request
4: *for* all available VMs
5: sum_res ← sum_res + Res_demand
6: *end for*
7: *if* (iteration 1)
8: Res_pool ← Res_pool - sum_res
9: *end if*
10: *else*
11: *for* all Cluster size, C_z = {c_1, c_2, c_3,..., c_z}
12: excess_res1 ← first best of c_z
13: excess_res2 ← second best of c_z
14: *end for*
15: *for* all available VMs {vm_0, vm_1, vm_2,..., vm_k}
16: *for* all Cluster size, C_z = {c_1, c_2, c_3,..., c_z}
17: *if* (excess_res1 == Needed_res[])
18: VM executes using excess_res1
19: *else*
20: Res_pool ← Res_pool - Needed_res[]
21: *if* (excess_res2 == Needed_res[])
22: VM executes using excess_res2
23: *else*
24: Res_pool ← Res_pool - Needed_res[]
25: *end for*
26: *end for*
27: Res_pool ← Res_pool + remaining Needed_res[]
28: *end for*

Next, we will propose the resource allocation and management algorithm using MCSO and thus overcoming the limitations of the proposed MPSO algorithm.

4.6 Proposed MCSO Algorithm for Resource Allocation and Management

In this thesis contribution, the proposed MCSO algorithm overcomes the limitations of the MPSO approach since MCSO algorithm achieves higher number of resource matching when compared to that of MPSO algorithm. The details of proposed MCSO algorithm are as follows.

Cats always remain calm and move slowly and this behavior of cats is referred to as seeking mode. When the presence of prey (resource match happens) is sensed, then cats chase it with high speed, and this behavior is referred to as the tracing mode. The tracing mode acts similar to that of MPSO algorithm, but the seeking mode awaits the opportunity to capture a prey. In the proposed MCSO algorithm, we mainly concentrate on the seeking mode rather than the tracing mode. In MPSO algorithm, it matches only with the *excess_res1* and *excess_res2* values from each cluster, but the remaining excessive resources from each cluster are not considered for the assignment and this issue is addressed in the proposed MCSO algorithm as shown in Algorithm 4.3.

As we know, the CSO algorithm passes through different types of memory and in the proposed MCSO these different types of memory play an important role in the decision making of resource assignment. Thus the seeking mode of MCSO has four different types of memory. The excess resources available with VMs other than *excess_res1* and *excess_res2* are stored in the seeking memory pool (SMP). Using SMP, the cats await the opportunity in order to find the exact match with the future resource demand for the tasks. If there is a match, then status is stored in the seeking range of selected dimension (SRD). If SRD has a new update, then VMs start executing the task and this status is referred to as counts to dimension change (CDC).

Algorithm 4.3 Resource Allocation and Management Using MCSO Technique

0: *Initialization*: Res_pool, Needed_res, min_res=2
sum_res= 0
VMs = {vm_0, vm_1, vm_2,..., vm_k}
Clusters, C_z = {c_1, c_2, c_3,..., c_z}
Cluster size = k/C_z
1: *for* all incoming requests {x_1, x_2, x_3,..., x_n}
2: Res_demand ← sum of resources from requests
3: Needed_res[] ← resource demand for each request
4: *for* all available VMs
5: sum_res ← sum_res + Res_demand
6: *end for*
7: *if* (iteration 1)
8: Res_pool ← Res_pool - sum_res
9: *end if*
10: *else*
11: *for* all Cluster size, C_z = {c_1, c_2, c_3,..., c_z}
12: excess_res3[] ← rest of first and second best from
13: each cluster
14: *end for*
15: *for* all available VMs {vm_0, vm_1, vm_2,..., vm_k}
16: *for* all Cluster size, C_z = {c_1, c_2, c_3,..., c_z}
17: *while* (size of(excess_res3[])
18: *if* (excess_res3[] == Needed_res[])
19: VM executes using excess_res3[]
20: *else*
21: Res_pool ← Res_pool - Needed_res[]
22: *end while*
23: *end for*
24: *end for*
25: Res_pool ← Res_pool + remaining Needed_res[]
26: *end for*

The cat's position is changing in every update of seeking mode of MCSO and it is stored in self position consideration (SPC). The tracing mode operation is applied to the outcome of seeking mode and the procedure is same as that of MPSO algorithm. There might be a situation in which all remaining excessive resources (other than *excess_res1* and *excess_res2*) may match with future resource demand; hence, time to lend the resources from the resource pool may be reduced considerably.

The seeking mode matching ratio of MCSO is greater than the best two i.e. *excess_res1* and *excess_res2* matching policies of the MPSO and thus MCSO will improve the dynamic allocation and management of cloud resources. However, the MCSO algorithm has the following limitations.

- *The values of excess_res3[] may not (rare case) match with the subsequent resource demand for task's execution.*
- *If the above condition is true, then the algorithm will be slower and communication overhead between VM and cloud resource pool will be increased.*

The proposed MPSO and MCSO approaches behave intelligently in assigning the cloud resources as efficiently as possible, but still there is a scope for further improvement in allocating the cloud resources. Thus we proposed the hybrid technique by combining both MPSO and MCSO approaches and hereafter it is referred to as HYBRID (MPSO+MCSO) algorithm. The details of HYBRID (MPSO+MCSO) algorithm are as follows.

4.7 Proposed HYBRID (MPSO+MCSO) Algorithm for Resource Allocation and Management

The limitations of the MPSO and MCSO algorithms can be overcome by our proposed HYBRID (MPSO+MCSO) Bio-Inspired algorithm which combines the merits of both MPSO and MCSO techniques. Thus, HYBRID approach provides better performance in terms of allocation of resources with reduced total execution time.

Further, communication overhead between VMs and resource pool is marginally decreased.

In MPSO and MCSO, we compare only exact matches of *excess_res1*, *excess_res2* with the *Needed_res[]* from future resource demand. In the worst-case, HYBRID approach may not work efficiently and degrades the performance of the resource allocation and thus the system will respond with high delay. To overcome this situation, we consider the *excess_res3[]* which contains the remaining resources other than *excess_res1* and *excess_res2*. The complete flow of HYBRID (MPSO+MCSO) technique is given in Algorithm 4.4.

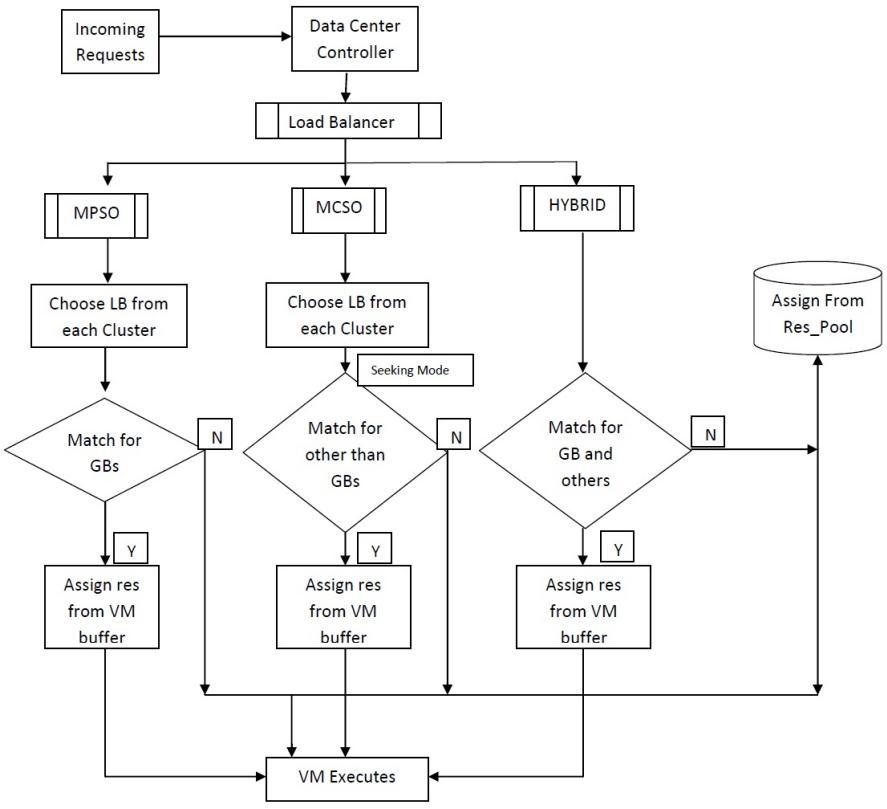


Figure 4.3: Overall Flow Diagram of Proposed Algorithms

In the proposed HYBRID approach, we modify the condition by checking the upper bounds of *excess_res1*, *excess_res2* and *excess_res3[]* of Algorithms 4.2 and 4.3. In doing so, we can decide how many resources from *excess_res1*, *excess_res2* and

Algorithm 4.4 Resource Allocation and Management Using HYBRID (MPSO+MCSO) Approach

0: *Initialization*: Res_pool, Needed_res, min_res=2
sum_res= 0
VMs = {vm_0, vm_1, vm_2,..., vm_k}
Clusters, C_z = {c_1, c_2, c_3,..., c_z}
Cluster size = k/C_z
1: *for* all incoming requests {x_1, x_2, x_3,..., x_n}
2: Res_demand ← sum of resources from requests
3: Needed_res ← resource demand for each request
4: *for* all available VMs
5: sum_res ← sum_res + Res_demand
6: *end for*
7: *if* (iteration 1)
8: Res_pool ← Res_pool - sum_res
9: *end if*
10: *else*
11: *for* all Cluster size, C_z = {c_1, c_2, c_3,..., c_z}
12: excess_res1 ← first best of c_z
13: excess_res2 ← second best of c_z
14: excess_res3[] ← rest of first and second best of c_z
15: *end for*
16: *for* all available VMs {vm_0, vm_1, vm_2,..., vm_k}
17: *for* all Cluster size, C_z = {c_1, c_2, c_3,..., c_z}
18: *if* (excess_res1 ≥ Needed_res[])
19: VM executes using excess_res1
20: *else*
21: Res_pool ← Res_pool - Needed_res[]
22: *if* (excess_res2 ≥ Needed_res[])
23: VM executes using excess_res2
24: *else*
25: Res_pool ← Res_pool - Needed_res[]
26: *while* (size 0f(excess_res3[]))
27: *if* (excess_res3[] ≥ Needed_res[])
28: VM executes using excess_res3[]
29: *else*
30: Res_pool ← Res_pool - Needed_res[]
31: *end while*
32: *end for*
33: *end for*
34: Res_pool ← Res_pool + remaining Needed_res[]
35: *end for*

excess_res3[] will be allocated to the forthcoming task and the remaining (unmatched) extra resources can be returned to the cloud Res pool. Thus, our proposed HYBRID algorithm outperforms MPSO and MCSO algorithms when considered individually. Algorithm 4.4 behaves intelligently from line numbers 12 to 30 as it combines the features of both Algorithms 4.2 and 4.3 with optimum modifications. We need to apply both approaches simultaneously, so that the lines numbered 18-25 and 26-29 will be executed in parallel using Python threads during the execution (applying both MPSO and MCSO). Hence, the matching possibility of *excess_res3[]* with future resource demand is more than that of MPSO and MCSO algorithms when taken separately. The performance evaluation of all proposed algorithms is discussed in results and analysis section of this chapter. Figure 4.3 gives the overall flow diagram of MPSO, MCSO and HYBRID (MPSO+MCSO) algorithms.

4.8 Performance Evaluation

In this thesis contribution, we proposed a scheduling and three resource allocation and management algorithms using Bio-Inspired techniques. The proposed algorithms are compared with other state-of-the-art techniques, Branch-and-Bound based Exact algorithm and analyzed with different QoS parameters. Further, we will discuss the experimental setup, results and analysis in the following subsections.

4.8.1 Experimental Setup

The proposed work is experimented on a simulator which gives the real-time scenario of cloud environment. The complete simulation scenario is written in Python language and hereafter it is referred to as Python Simulator, in short, PySim. Figure 4.4 shows the block diagram of PySim, which is used for experimenting the proposed Bio-Inspired algorithms.

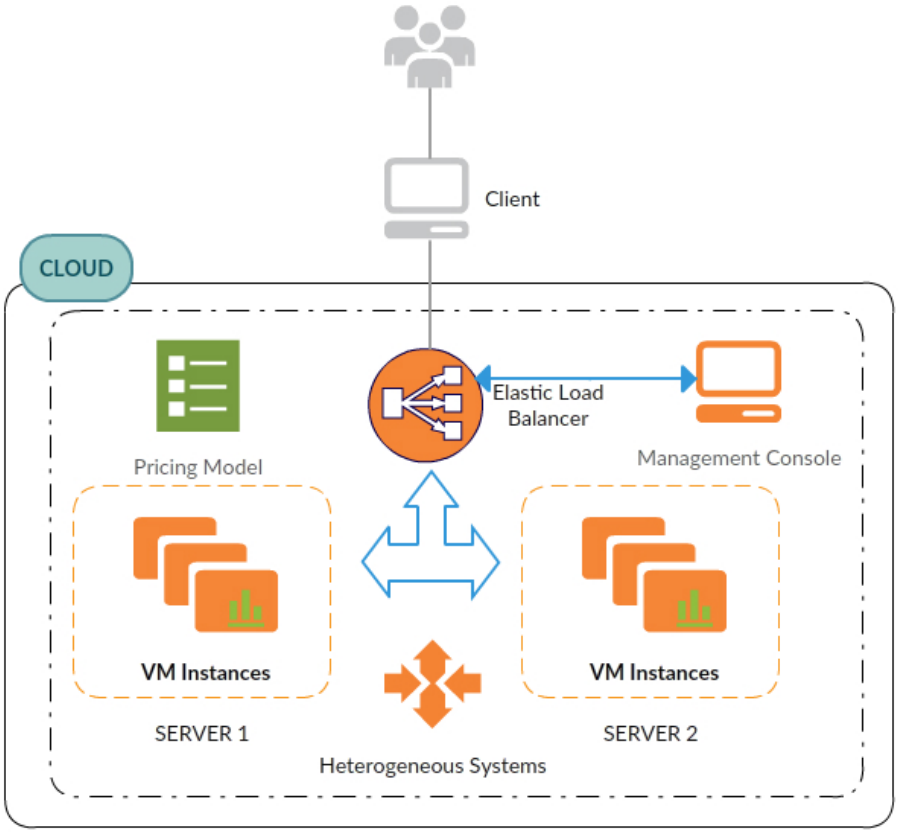


Figure 4.4: Block Diagram of PySim

The experimental setup consists of four physical machines which are interconnected with different system configurations. The configuration details of customized simulation setup are given in Table 4.1 and it consists of four different machines in which one machine acts as a task sender (Client Machine), one machine acts as a load balancer and other two machines act as servers with several VMs which execute the incoming tasks.

Here, the task refers to the execution of a job which demands several resources. We considered CPU and Memory as two types of resources which are needed for executing a task. Resources are given by the cloud resource pool which has a huge number of resources. The systems used in the customized setup are with different system configurations such as the load balancer with i7 processor of 3.40 GHz clock speed and 8 GB RAM; the server machines with i7 processor of 3.40 GHz clock speed

Table 4.1: Configuration Details of PySim

Machine	**GHz**	**RAM(GB)**	**Storage(GB)**
Client Machine	2.80	4	100
Load Balancer	3.40	8	100
Server 1	3.40	16	200
Server 2	3.40	16	200

Table 4.2: Configurations of VMs

VM Type	**MIPS**	**RAM(GB)**	**Storage(GB)**
Small	500	0.5	20
Medium	1000	1	30
Large 1	1500	2	40
X. Large	2000	3	50
Extra Large	2500	4	50

and 16 GB RAM and other client machine which generates tasks with a dual core processor of 2.80 GHz clock speed and 4 GB of RAM.

Experimentation is carried out with different sets of tasks and VMs. The performance evaluation is carried out in terms of efficient utilization of available virtual machines, average response time and optimum usage of cloud resources. If the resources are not in use, then these unused resources are given back to the cloud resource pool. We consider Round Trip Time (RTT) of 1 second when the VMs take the resources from cloud resource pool. Clusters are made depending on the number of VMs and each of the VMs has its own buffer to store the excess resources. Table 4.2 shows the VMs configuration used in the experiment.

4.8.2 Results and Analysis

The results of our proposed Bio-Inspired algorithms are analyzed with respect to QoS parameters (Reliability and Time). The proposed algorithms achieve better performance in terms of efficient utilization of cloud resources (VMs, Memory and CPU) and better average response time. Further, the proposed algorithms are compared with Branch-and-Bound based Exact Algorithm. We also carried out statistical analysis for null hypothesis using T-Test. Further, the time complexity of the proposed HYBRID (MPSO+MCSO) algorithm is also analyzed. Next, we will discuss about the usage of Branch-and-Bound based Exact algorithm with which we evaluate the performance of our proposed algorithms.

Branch-and-Bound Based Exact Approach

To evaluate the performance, proposed algorithms are compared with bench mark solution based on Branch-and-Bound based Exact algorithm (Mingozzi et al. (1998)). Since the exact algorithm gives the global optimum solution for efficient utilization of cloud resources, hence we compared our proposed algorithms with Exact algorithm, and the details of the Exact algorithm are as follows. Let *'N'* be the node at level *'l'* of the search tree along with *'lb'* as lower bound and *'ub'* as upper bound. The root node N0 corresponds to an empty solution and each node at level $l \geq 1$ corresponds to the partial solution. Initially at root level, all VMs are not allocated and the child node is created by comparing the MIPS of each VMs along with resource demand. Further, the child node N^+ is created only if the following conditions are met.

- MIPS(VMs) $\leq$ MIPS(task)
- Resources(VMs) $\leq$ Resource Demand(task)
- Resources(VMs) $\neq$ Resource Demand(task)

Hence, if the aforementioned conditions are met, then *'lb'* is computed. Later, if it reaches *'ub'*, then the child node is pruned and the best fit VMs are chosen for the scheduling, resource allocation and thus execution of tasks is carried out.

Evaluation Criteria

In the proposed work, incoming tasks, i.e. $\{x_1, x_2, x_3, ..., x_n\}$ are to be scheduled on the virtual machines, i.e. $\{vm_0, vm_1, vm_2, ..., vm_k\}$. Further, the cloud resources demanded by these requests are intelligently handled by our proposed algorithms. Hence, the objectives for evaluating the performance of proposed algorithms are as follows.

Objective 1: Efficient utilization of VMs.

$$VM \;\; Type(i) = Number\ of\ tasks\ handled\ by\ VMs. \tag{4.8.1}$$

VM Type(i) refers to type of VM used in the experiment. We have used five different types of VMs (Small, Medium, Large, X-Large, Extra Large) and more details are given in Table 4.2 of Section 4.8.1.

Objective 2: Minimization of Average Response Time.

Average Response Time (AR_Time) is the total amount of time taken for responding to a task and AR_Time is calculated by the following Equation 4.8.2.

$$AR\_Time = t_2[\sum_{x=1}^{n} VM \;\; Type(i)] - t_1[\sum_{x=1}^{n} VM \;\; Type(i)] \tag{4.8.2}$$

$$FV = \frac{(R)\sum_{vm=0}^{k} VM \;\; Type(i)}{\sum_{x=1}^{n}(RD)} \tag{4.8.3}$$

For resource allocation and management, the fitness value (FV) is given by the Equation 4.8.3. Here, at each iteration, we find the FV and then choose the suitable

VMs for assigning the task for the execution. If FV value is feasible then it takes unused resources from the VMs otherwise it assigns the resources from the cloud resource pool as explained in the example. Table 4.3 gives the notations and definitions used in the proposed methodology.

Table 4.3: Notations and Definitions

Notations	**Definitions**
$VMType_{(i)}$	Type of VM on which the task is being executed
RD	Number of resources demanded by tasks
R	Resources (CPU and Memory)
t_1	Time at which task enters the VM
t_2	Time at which task completes the execution
excess_res1	First highest resource from the cluster
excess_res2	Second highest resource from the cluster
Needed_res[]	Number of resource demand from the request
excess_res3[]	Remaining resources other than the best excess_res1 & excess_res2

Load Balancing of VMs by MPSO

In this thesis contribution, MPSO algorithm is used for efficient scheduling of tasks on VMs and this MPSO algorithm will ensure that the allocated VMs are utilized in a more efficient manner.

From Table 4.4, we can observe that both Round Robin and proposed MPSO algorithms efficiently schedule the tasks as compared to Throttled algorithm. But, if we compare Round Robin and proposed MPSO algorithms, the results are same. But in Round Robin, we do not check the state of the VM (BUSY/AVAILABLE) and

Table 4.4: Utilization of VMs

Sl. No.	**Throttled**	**Round Robin**	**ACO**	**Exact Algorithm**	**Proposed MPSO**
VM1	1182	254	356	253	254
VM2	76	254	289	254	254
VM3	8	253	389	254	254
VM4	2	253	180	254	253
VM5	0	254	54	253	253

Table 4.5: Average Response Time Analysis

Algorithms	**Average Response Time**
Throttled	365.52ms
Round Robin	364.85ms
ACO	362.67ms
Exact Algorithm	365.87ms
Proposed MPSO	360.11ms

thus leads to the queuing of the incoming task on the server. The proposed MPSO algorithm gives better results by checking the state of the VM and it has cluster based comparison of allocating tasks to a VM and thus avoids the server queuing. In Ant Colony Optimization (ACO), the tasks can be assigned to VMs which have less pheromone content. In Exact algorithm, the scheduling of the tasks is similar to that of proposed MPSO but the Exact algorithm needs more average response time. The same steps are repeated with different number of clusters when there are same number of VMs. For the aforementioned experiment, we used two clusters with equal number of VMs on each cluster and the average response time in milliseconds (ms) for the same setup is given in Table 4.5.

It is clearly observed from Tables 4.4 and 4.5 that the proposed MPSO algorithm is not only efficient in balancing the load on the virtual machines but also achieves

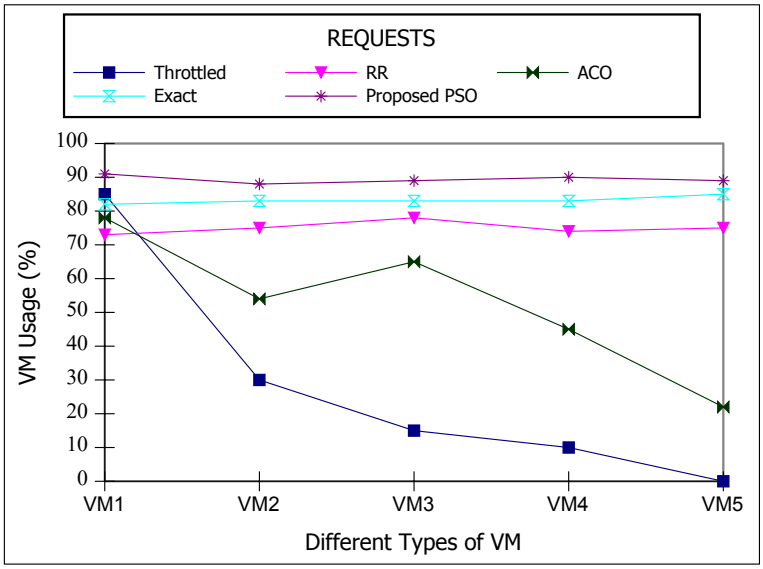


Figure 4.5: Average Utilization of VMs

better average response time. The experiment is repeated for different sets of virtual machines, tasks and the results obtained are consistent.

Till now performance analysis is carried out using small load (few tasks) on different VMs according to the standards used in Amazon cloud offerings. Next, we will analyze the utilization of VMs with different combinations of 1000, 2000 and 3000 incoming tasks. Figure 4.5 shows the utilization of VMs using different algorithms used in the experiment. It is observed from Figure 4.5 that Throttled and ACO algorithms are not consistent in utilizing the VMs when compared to other algorithms. Even though RR and Exact algorithms utilize the VMs efficiently, but our proposed MPSO algorithm is more efficient in utilizing the VMs with less Average Response time. As MCSO algorithm uses two modes (seeking and tracing mode) then during scheduling seeking mode takes more time when compared to that of the tracing mode. Hence, we did not consider proposed MCSO for scheduling approach. Similarly, our proposed HYBRID (MPSO+MCSO) is also not considered for scheduling since it combines both MPSO and MCSO.

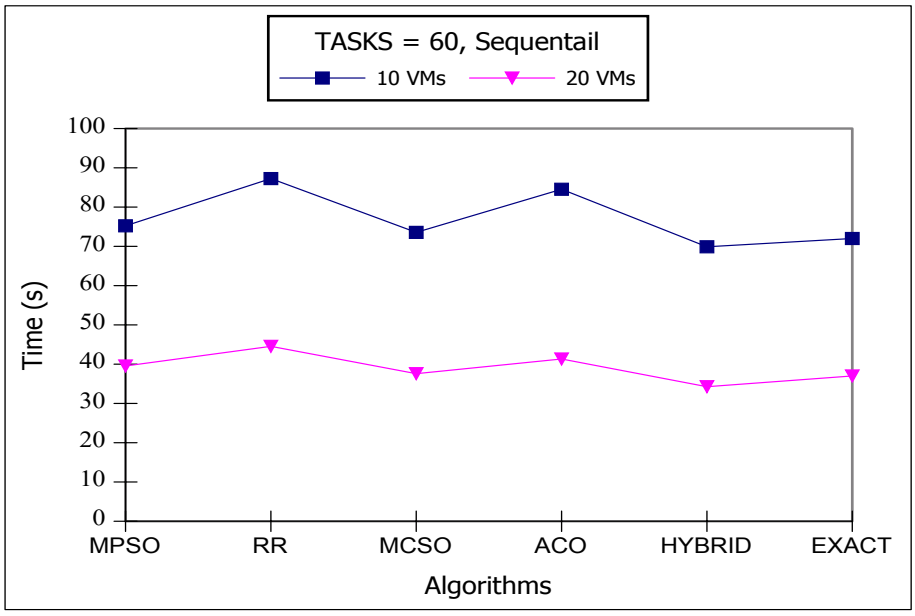
Resource Utilization by MPSO, MCSO and HYBRID (MPSO+MCSO)

The incoming tasks enter the cloud with different resource requests and VMs facilitate the resources as demanded by the tasks which are managed by the proposed techniques of this chapter. Proposed algorithms are experimented with different number of VMs and tasks. In the first iteration, VMs take resources from cloud resource pool and from the second iteration onwards the proposed algorithms such as MPSO, MCSO and HYBRID (MPSO+MCSO) allocate the resources either from cloud resource pool or from the unused resources of VMs. We experimented the proposed algorithms with two clusters which have equal number of VMs. Here, the inter arrival time between the tasks remains same and the tasks are served in a batch of ten. Once the tasks are served by the VMs, then the excess resources are stored in the individual buffers of VMs. Further, these resources are used for serving the upcoming task demand. As discussed earlier, initially VMs start executing the tasks by lending the resources from the cloud resource pool. Experiments are conducted for evaluating the Average Response Time in both sequential and parallel modes; and the corresponding results are shown in Figures 4.6 and 4.7, respectively.

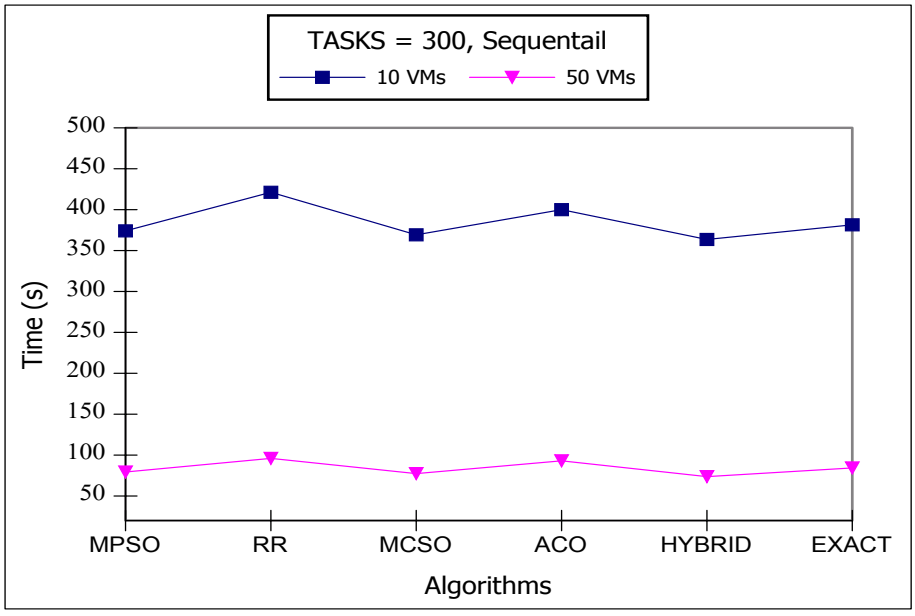
In Figure 4.6a, we used 10 and 20 VMs in two clusters with 60 incoming tasks. In Figure 4.6b, we used 10 and 50 VMs in two clusters with 300 incoming tasks. In MPSO, for the best match (GB) case approach, the two best VMs from each cluster are taken for resources matching with the upcoming demands. For example, excess res1 and excess res2 from each cluster are compared with resource demand. Further, the remaining VMs will follow the MCSO approach. For example, excess res3[] from each cluster is compared with the upcoming demand. Our HYBRID (MPSO+MCSO) approach combines the merits of both MPSO and MCSO and thus the HYBRID approach achieves superior performance for the best match case scenario.

In Figure 4.7a, we used 10 and 20 VMs in two clusters with 60 incoming tasks and in Figure 4.7b, we used 10 and 50 VMs in two clusters with 300 incoming tasks. It is clearly observed from Figure 4.7 that the parallel mode execution will reduce the

average response time as compared to sequential mode.

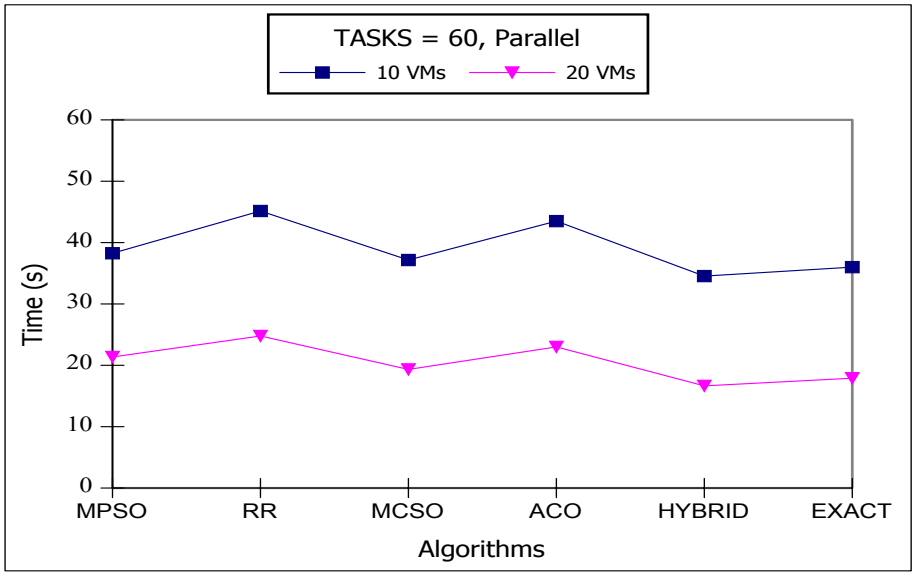


(a) Sequential Analysis with 60 Tasks

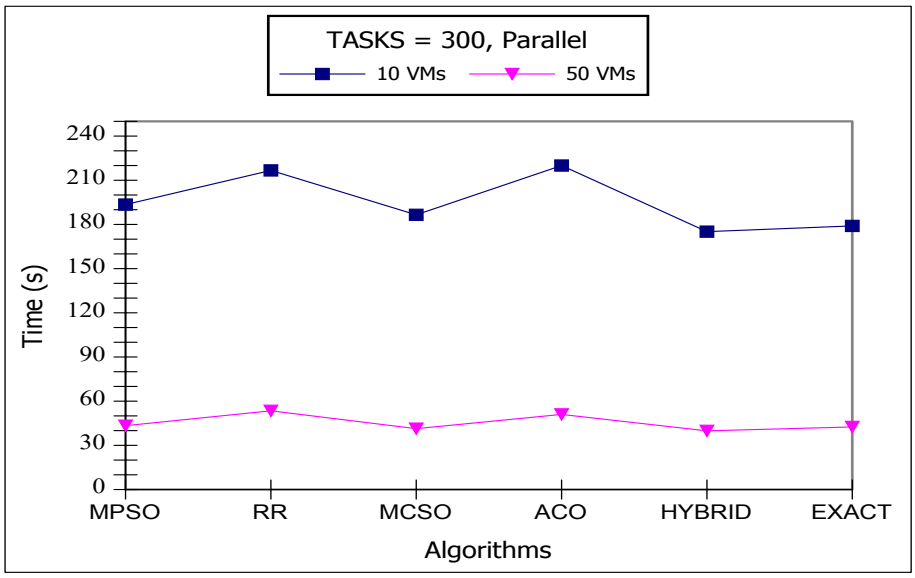


(b) Sequential Analysis with 300 Tasks

Figure 4.6: Sequential Analysis



(a) Parallel Analysis with 60 Tasks



(b) Parallel Analysis with 300 Tasks

Figure 4.7: Parallel Analysis

Figures 4.6 and 4.7 illustrate that MPSO, MCSO and HYBRID (MPSO+MCSO) algorithms give feasible solutions with respect to average response time when compared to Branch-and-Bound based Exact algorithm and further our HYBRID algorithm takes less time when compared to other state-of-the-art the algorithms.

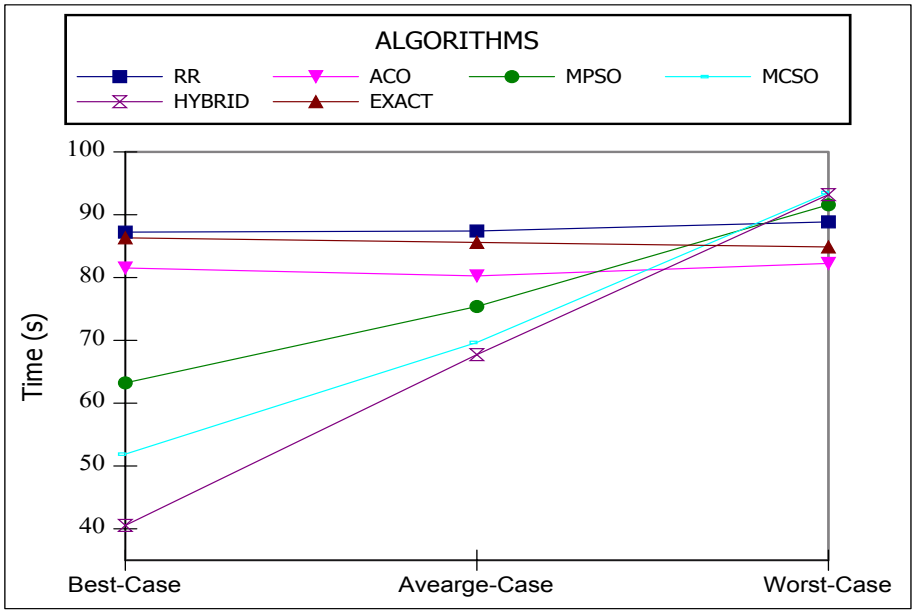


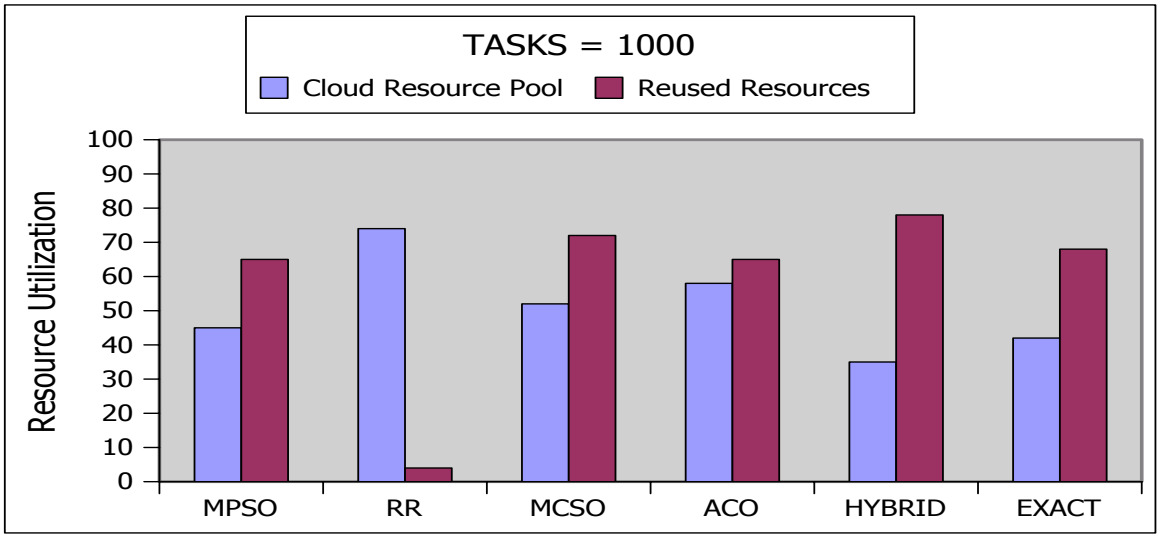
Figure 4.8: Execution Time Analysis of Proposed Algorithms

Further, we will analyze the execution time analysis for different cases (Best, Average and Worst) of our proposed algorithms along with benchmark algorithms. We experimented with different combinations of tasks and varied numbers of VMs using the proposed and state-of-the-art benchmark algorithms. It is observed from Figure 4.8 that, the worst-case execution time for all the proposed algorithms is a rare case in which the matching never happens with excess resources present in the buffer with the future resource demand. Since RR takes the resources from the cloud resource pool for most of the time, hence the execution time of the RR will remain the same for all the cases. In ACO, the resources match with the VMs which have the highest pheromone content. Since there will not be any perfect resource match for most of the time due to high pheromone evaporation rate and hence ACO takes same execution time in all cases. The Exact algorithm tries to match with all possible combinations and then allocates resources with possible solutions. Hence the Exact algorithm also takes same execution time in all cases.

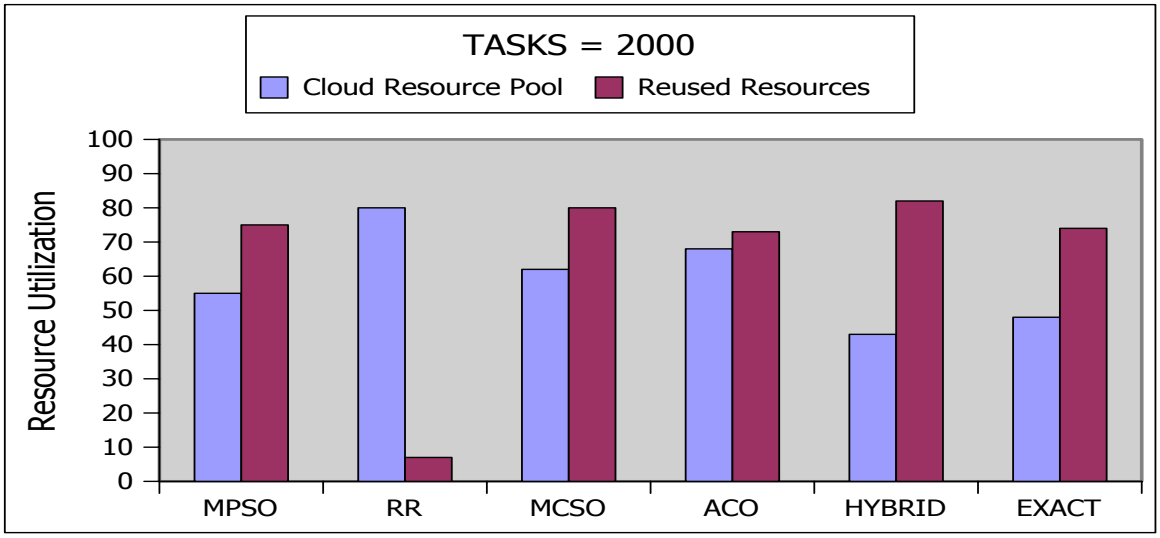
The worst-case execution time of MPSO, MCSO and HYBRID (MPSO+MCSO) approaches are continuously increasing because of the delay in the mismatch of resources from the VMs buffer to the future resources demand. In the MPSO, the best-case is considered when the best two values from each cluster match with the upcoming resource demand. In the average-case, resources match is varied so that MPSO can work efficiently when compared to RR and ACO. In MCSO, the best-case is considered when the rest of the values (other than *excess_res1* and *excess_res2*) are matched with excess resources from the VMs buffer to the future resource demand. Thus, in MCSO, more resources got matched when compared to MPSO and hence, MCSO outperforms MPSO, ACO and RR algorithms.

Further, HYBRID approach takes less execution time in both best and average-cases. And if all the resource mappings are true in the HYBRID approach, then HYBRID algorithm outperforms both MPSO and MCSO when considered separately. Hence, HYBRID (MPSO+MCSO) approach is more efficient in terms of resource allocation and management in the cloud environment.

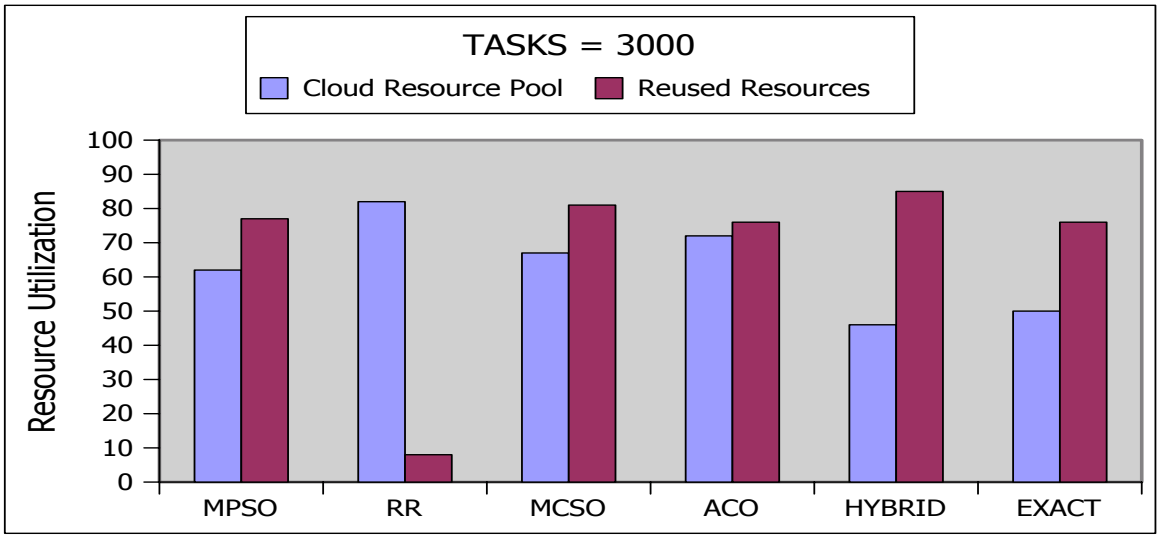
As mentioned earlier, resources are either taken from the cloud resource pool or unused resources of the respective VMs from the clusters. Accordingly, the proposed algorithms are analyzed with respect to resource utilization factor. Figures 4.9a, 4.9b and 4.9c show the resource utilization with 1000, 2000 and 3000 tasks respectively. In all three cases, RR takes maximum resources from the cloud resource pool. ACO takes an almost equal amount of resources from both cloud resource pool and unused resources from VMs. Our proposed MCSO provides better resource utilization as compared to proposed MPSO. On the other hand, the proposed HYBRID (MPSO+MCSO) algorithm is more efficient in utilizing the unused resources from clusters rather than taking it from the cloud resource pool.



(a) Resource Utilization with 1000 Tasks



(b) Resource Utilization with 2000 Tasks



(c) Resource Utilization with 3000 Tasks

Figure 4.9: Average Resource Utilization Analysis

All the proposed algorithms are compared with the benchmark Exact algorithm. It is clearly observed from Figure 4.9 that our proposed HYBRID (MPSO+MCSO) algorithm achieves high resource utilization efficiency when compared to all other state-of-the-art methods considered for performance evaluation. The performance analysis is carried out on the proposed Bio-Inspired algorithms on two different QoS parameters, i.e. efficient utilization of VMs and complete utilization of cloud resources. Next, we will analyze the time complexity of the HYBRID algorithm and followed by the statistical hypothesis on proposed algorithms.

Statistical Analysis on Workloads

In the cloud environment workloads will vary depending on the applications. The real time applications need resources with no delay and cognitive applications need more number of VMs with dedicated cores in them. So as the workload varies, the algorithm should behave consistently without affecting the performance during the execution. Hence, Table 4.6 gives the complete statistical analysis on the workloads used in the performance analysis of the proposed algorithms. For statistical analysis, Table 4.6 gives the Variance, the Standard Deviation (SD), the Confidence Level (CL) and the Accuracy of the Workloads (1K, 2K and 3K) taken for the experimentation. Our results show that, irrespective of changes in the Variance, the Standard Deviation and the Confidence Level, the Accuracy results will not deviate and thus achieving the consistent performance.

Workloads are categorized as Small scale (MIPS varies from 500 to 1000), Medium scale (MIPS varies from 1000 to 5000) and Large scale (MIPS varies from 5000 to 10000). All the proposed algorithms are evaluated with these categories for 1000, 2000 and 3000 tasks respectively. Every time, the experiments are repeated in each scale and the average results are taken for performance analysis.

Table 4.6: Statistical Analysis on Workloads

Workloads	**Range**	**Variance**	**SD**	**CL**	**Accuracy**
Small Scale (1K)	500 to 1000	20991.14	144.88	713.66 to 799.87	95.42%
Medium Scale (1K)	1000 to 5000	1208899.86	1099.49	2854.34 to 3472.04	96.72%
Large Scale (1K)	5000 to 10000	2007141.43	1416.73	7150.48 to 7718.67	96.05%
Small Scale (2K)	500 to 1000	18062.27	134.39	702.36 to 785.68	96.84%
Medium Scale (2K)	1000 to 5000	1333937.04	1154.96	2965.45 to 3499.58	96.47%
Large Scale (2K)	5000 to 10000	2095914.77	1447.72	7111.58 to 7789.36	95.98%
Small Scale (3K)	500 to 1000	22252.45	149.17	725.56 to 820.48	96.40%
Medium Scale (3K)	1000 to 5000	1422554.74	1192.70	2992.58 to 3548.22	96.12%
Large Scale (3K)	5000 to 10000	2141372.20	1463.34	7211.25 to 7799.87	96.57%

Time Complexity Analysis

Our proposed HYBRID algorithm is based on MPSO and MCSO techniques. In this experiment, n number of tasks which are scheduled on heterogeneous VMs hosted on C_z clusters (fixed number of clusters). Therefore, the time complexity of our proposed HYBRID (MPSO+MCSO) algorithm is [O(number of tasks) * O(number of clusters) *O(number of m iterations)] * [O(MPSO) + O(MCSO)] i.e. [$O(n)$ * $O(C_z)$ * $O(m)$] * [$O(vm_k,C_z)$ + $O(vm_k,C_z)$]. It is reduced to $O(n * C_z * m * vm_k)$, since C_z and m are constants. Finally, the time complexity of proposed HYBRID algorithm is $O(n * vm_k)$, which is $O(n) < O(n * vm_k) < O(n^2)$.

Table 4.7: Results of T-TEST Analysis

Algorithms	**P-values**
RR-HYBRID	0.005
ACO-HYBRID	0.0035
MPSO-HYBRID	0.0015
MCSO-HYBRID	0.001
EXACT-HYBRID	0.0045

Statistical Hypothesis Analysis

In the statistical hypothesis, two entities are evaluated about a given workload or population so that we can determine the best supported entity for different cases. Hence, we made statistical analysis by evaluating the proposed HYBRID (MPSO+MCSO) algorithm with state-of-the-art benchmark algorithms. For statistical hypothesis, we used T-Test analysis for resource allocation and management. The results of T-Test analysis are shown in Table 4.7. For the null hypothesis analysis, Threshold value of P or Significance Level of α are considered. In this experiment, we considered $\alpha = 0.05$ which is the standard cutoff for null hypothesis. The P-value of HYBRID (MPSO+MCSO) algorithm in comparison with other algorithms is less than $\alpha = 0.05$ and thus it rejects the null-hypothesis. Hence, our proposed HYBRID (MPSO+MCSO) algorithm is more efficient when compared to all other algorithms considered for Resource Allocation and Management.

4.9 Summary

In this chapter, we proposed a Bio-Inspired (MPSO) scheduling algorithm and three (MPSO, MCSO and HYBRID) algorithms for resource allocation and management of the IaaS based cloud environment. The proposed Bio-Inspired algorithms outperform the state-of-the-art and benchmark algorithms in terms of QoS parameters

such as average response time, total execution time and utilization of cloud resources mainly memory and CPU. The proposed algorithms provide the statistical hypothesis analysis and has better time complexity when compared to other state-of-the-art algorithms. Both MPSO and MCSO algorithms are efficient in utilizing the cloud resources from VMs, but HYBRID approach outperforms both MPSO and MCSO algorithms in terms of better execution time. Further, statistical analysis on workloads shows that, irrespective of changes in the Variance, the Standard Deviation and the Confidence Level, the Accuracy results will not deviate and thus achieving the consistent performance.

In the next chapter, we propose another Bio-Inspired approach for optimizing the QoS parameters (Execution Time and Cost) and how it is useful in the cloud environment. Further, we will also analyze how this Bio-Inspired approach handles both dependent and independent workloads.

Chapter 5

Bag-of-Tasks and Workflows Scheduling at Cloud Data Center

There are several existing works on the Bag-of-Tasks and Workflows scheduling but still, there is a scope for further improvement of resources allocation while minimizing the execution cost since these algorithms come under NP-Hard/NP-Complete complex class. Furthermore, the existing works fail to meet users' expectation with respect to Service Level Agreement (SLA) in heterogeneous cloud computing environment.

In the previous chapters (Chapters 3 and 4), we proposed a few scheduling algorithms with better performance in terms of QoS parameters (Reliability and Time) when compared to other state-of-the-art approaches. In this thesis contribution, we mainly focus on developing a novel Bio-Inspired scheduling algorithm with cost optimization. Hence, we propose an application of Grey Wolf Optimization (GWO) technique which aims to minimize the Workflows execution cost and thereby maximizing the BoT execution speed while efficiently utilizing the cloud resources (VMs) in the Infrastructure as a Service (IaaS) based cloud computing environment. The key contributions of this chapter are as follows.

- To design and develop an efficient scheduling of BoT with minimal execution time using GWO based Bio-Inspired approach.

- To design and develop an efficient scheduling of workflow tasks on the critical path with minimal execution cost using the proposed GWO technique.
- Statistical analysis of proposed algorithms using T-Test for null-hypothesis and the performance evaluation of proposed algorithms in terms of QoS parameters (Cost, Reliability and Time).

Our proposed scheduling algorithms are evaluated using various scientific workflows and experimental results demonstrate that our proposed heuristic-based scheduling algorithms outperform peer research and benchmark algorithms in terms of efficient utilization of resources (VMs), minimal execution time and reduced cost. The details of proposed GWO based scheduling algorithms are as follows.

5.1 Proposed GWO based Scheduling Algorithms

In this contribution, we consider BoT and workflow as an input to the cloud environment and then proposed GWO based Bio-Inspired algorithms are used for efficient scheduling. The details of BoT and Workflows are as follows.

5.1.1 Basics of BoT

A BoT application can be modeled as B = (T), where T =$\{t_1, t_2, ..., t_n\}$ is the set of tasks which are independent of each other and these tasks should be scheduled in an efficient manner. Since the tasks in the BoT are independent, hence these tasks can be scheduled as and when they arrive in the job queue. Figure 5.1 shows the sample BoT in which each task has some job to be executed on the assigned VM from the scheduler.

Figure 5.1: Sample BoT with Independent Tasks

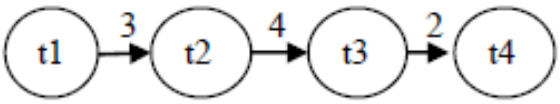


Figure 5.2: Sample Workflows Application with Dependent Nodes

5.1.2 Basics of Workflows

Application workflows can be modeled by a Directed Acyclic Graph (DAG) which is defined as G = (T, E) where G is the graph of Workflows, T =$\{t_1, t_2, ..., t_n\}$ is the set of tasks or nodes in the Workflows and E is the set of direct edges between the tasks. A data dependency exists if there is an edge between two nodes behaving like a parent and child nodes. The child node cannot be executed unless the parent node completes the execution. Figure 5.2 shows the sample workflows application in which each task has a deadline associated with it and it has to execute within a given time limit in the workflows.

5.1.3 Example for Explaining Our Proposed Work

In this proposed work, tasks are scheduled on the suitable VMs by our proposed GWO algorithm. BoT and Workflows are used as a workload during the experimentation of proposed GWO. Figure 5.3 shows an example for explaining our proposed work. The incoming tasks may be of two kinds i.e. BoT or Workflows. In BoT, the GWO based scheduler must choose the suitable VM from different clusters. For Workflows, GWO based scheduler checks for the critical path, then chooses the suitable VM from the clusters. The details about choosing the VM from clusters and finding the critical path are given in the following sections.

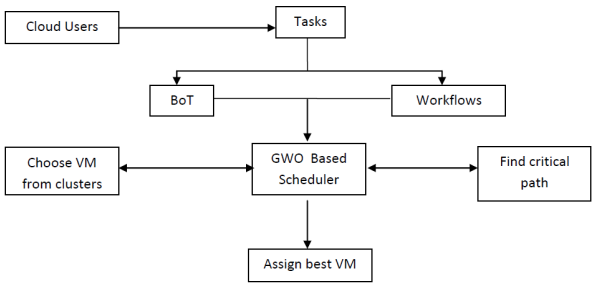


Figure 5.3: Example for Explaining Our Proposed Work

5.1.4 Proposed Methodology

In Chapter 4, we proposed MPSO, MCSO and HYBRID heuristic approaches for different objectives in the IaaS based cloud environment. In this work, we proposed a GWO based meta heuristic Bio-Inspired algorithm. The details of Grey Wolf Optimization technique are as follows.

Grey wolf belongs to the Canidae family and these are referred to as apex predators, indicating that they are considered to be on top of the food chain in a biological system and always roam in a group with varying sizes (Mirjalili et al. (2014)). GWO mimics the leadership hierarchy and the hunting mechanism of grey wolves from nature. As GWO is a Bio-Inspired algorithm, this can be applied to solve NP-Hard/NP-Complete complex problems. Thus, the proposed GWO based scheduling algorithm mimics the leadership hierarchy and the hunting mechanism of the grey wolves.

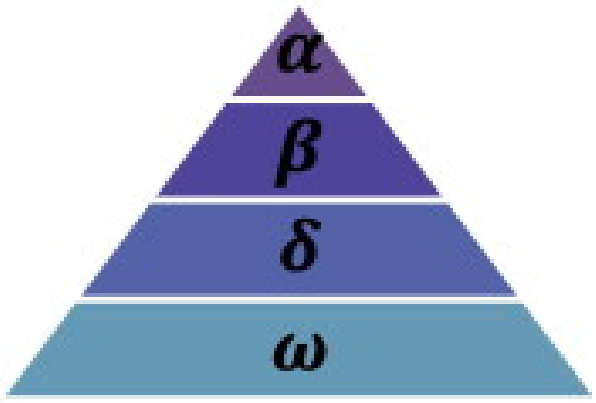


Figure 5.4: Hierarchy of Grey Wolves (Mirjalili et al. (2014))

Here, the hunting refers to the choosing the right VM to execute a given task from the BoT or from the Workflows. There are four types of grey wolves (alpha, beta, delta and omega) and these helps in three main steps of hunting like, searching the prey, encircling the prey and catching the prey. Figure 5.4 shows the hierarchy of Grey wolves. The first level of hierarchy is alpha wolves which are the strongest, decision makers and dominate the entire pack.

The second level of hierarchy is beta wolves which are subordinates to the alpha wolves and dominate the lower levels in the pack. In case if alpha passes away, then, beta will take place on alpha and rules the entire pack. The third level of hierarchy is delta wolves which are obeying both the alpha and beta wolves, but the delta wolves dominate the omega wolves (the lowest level of hierarchy). These omega wolves are just followers of the pack and helps in finishing the food. They are large in number and help the pack to move forward towards catching a prey, i.e. choosing the suitable VM from the clusters.

In this proposed work, we deploy the GWO based scheduler for efficient utilization of resources (VMs) of IaaS based heterogeneous cloud platform. We concentrate mainly on the scheduling of the BoT and Workflows to minimize the overall execution time and cost respectively. The mathematical model for the scheduling is as follows.

$$D = [C * Xp(t)X(t)] \tag{5.1.1}$$

$$X(t+1) = Xp(t)A * D \tag{5.1.2}$$

$$A = 2a * r1a \tag{5.1.3}$$

$$C = 2 * r2 \tag{5.1.4}$$

Where,

X is the current position of VM

Xp is the VM given by the hierarchy / pack

t is the current iteration

r1 and *r2* are random numbers between 0 and 1

a is load of VM and is decreased from higher to lower

In this work, the hierarchy of grey wolves is represented by α, β, δ and ω respectively. In addition to this, we added super α as a supreme wolf who takes a final decision regarding which VM is selected for the current task scheduling. The analysis is carried out in two different aspects, i.e. scheduling of BoT for fast execution so that the total execution time should be as minimum as possible along with efficient utilization of VMs. Further, by considering the Workflows dependency and deadlines, we can minimize the total cost incurred for the execution by applying proposed GWO scheduling technique. Next, we discuss the scheduling of BoT using the GWO technique and the details are as follows.

Algorithm 5.1 Scheduling of BoT on VMs Using GWO Technique

0: ***Initialization***: $\{vm_0, vm_1, vm_2,..., vm_k\}$ α, β, δ, ω and super α
VMs = $\{vm_0, vm_1, vm_2,..., vm_k\}$
earch search space S_z = {s1,s2,s3}
search space size, k/S_z
1: *for* all incoming BoT requests $\{x_1, x_2, x_3,..., x_n\}$
2: α = Suitable VM from s1
3: β = Suitable VM from s2
4: δ = Suitable VM from s3
5: ω = Listing busy VMs
6: *end for*
7: Assign super α = VMs from α, β, δ search spaces
8: Next task allocated to VM which chosen from Super α
9: *if* (Next allocations == last used GB) then skip
10: goto Step 1 for suitable VM
11: *else*
12: goto Step 8
13: Makespan = (Finish time of current task - Start time of current task) on a VM

The Algorithm 5.1 shows the scheduling of BoT on IaaS based heterogeneous cloud platform. Incoming tasks in the BoT $\{x_1, x_2, x_3,..., x_n\}$ are to be scheduled on the available VMs $\{vm_0, vm_1, vm_2,..., vm_k\}$ efficiently to balance the load on each VM. Here, the tasks are independent and scheduling of these tasks takes place depending on their arrival time in a job queue. The entire search space of the platform is divided into three regions and it is managed by α, β, δ respectively.

Each region contains heterogeneous VMs with varying configurations. Each search space gives a suitable VM, depending on its load, status and usage. Then the super α decides the suitable VM for the current allocation of the task which depends on the machine status, configuration and also it takes care of the parameter which is to be optimized. In this BoT scheduling, we concentrate on Time QoS parameter i.e. to get minimal execution time for the execution of tasks. Next, we discuss how the proposed GWO algorithm helps to optimize the cost in scheduling the scientific workflows and the details are shown in Algorithm 5.2.

Algorithm 5.2 focuses on optimizing the cost while efficiently scheduling the workflows without missing the deadlines. As workflows contain the combinations of many dependent and independent tasks and therefore the execution of workflows has a certain pattern with critical paths. Each of workflows has a certain structure and jobs are usually depending on one or more tasks at different hierarchy. Super α plays a vital role in deciding which tasks are to be executed in the current execution and finding the critical path for the dependent tasks. Further, super α decides the optimum VM for the current allocation by considering its cost, status, speed etc. Performance analysis of proposed algorithms are as follows.

Algorithm 5.2 Scheduling of Scientific Workflows on VMs Using GWO Technique

0: ***Initialization***: $\{vm_0, vm_1, vm_2,..., vm_k\}$ α, β, δ, ω and super α
VMs = $\{vm_0, vm_1, vm_2,..., vm_k\}$
earch search space S_z = {s1,s2,s3}
search space size, k/S_z
1: *for* all incoming workflow requests $\{x_1, x_2, x_3,..., x_n\}$
2: Set 1 = Find independent tasks
3: Set 2 = Find dependent tasks and critical path
4: *end for*
5: *for* all Set 1 and Set 2
6: α = Suitable VM from s1
7: β = Suitable VM from s2
8: δ = Suitable VM from s3
9: ω = Listing busy VMs
10: *end for*
11: Assign super α = VMs from α, β, δ search spaces
12: Super α = Choose optimum VM w.r.t. cost
13: Task allocated = VM chosen from super α
14: *if* (Next allocations $\neq$ Optimum VM) then skip
15: goto Step 1 for next suitable VM
16: *else*
17: goto Step 12

5.2 Performance Evaluation

In this Chapter, we proposed GWO based algorithms to schedule the BoT and Workflows. Next, we will discuss the experimental setup and objective functions which influence our proposed algorithms to perform better than the state-of-the-art techniques. Further, we will discuss how the QoS parameters of BoT and Workflows are optimized using our proposed GWO based scheduling algorithms.

Table 5.1: Amazon EC2 Units

Name	**EC2 Units**	**Processing Capacity (MFLOPS)**
m1.small	1	4400
m1.medium	2	8800
m1.large	3	17600
m1.xLarge	8	35200
m1.xLarge	13	57200
m3.doubleXLarge	26	114400

5.2.1 Experimental Setup

The research contributions in this Chapter are experimented on the customized simulation environment implemented in Python language. The complete details of the experimental setup are given in Section 4.8 of Chapter 4. Further, we will discuss the types of Bot, Workflows and VMs used in the experiment.

In this work, we considered the Amazon EC2 instances as VMs for the performance evaluation. These Amazon EC2 instances are heterogeneous in nature and our proposed GWO algorithms play a vital role in choosing the suitable instance from the clusters. In our work, the VMs are distributed in different clusters with different configurations. Table 5.1 shows the Amazon EC2 instances (Rodriguez and Buyya (2014)) which are used by both BoT and Workflows for different case studies during the experiment. The results of these case studies are as follows.

The proposed algorithms mainly concentrate on QoS parameters (Reliability, Time and Cost), i.e. fast execution of BoT and cost optimization of Workflows respectively. Algorithm 5.1 is experimented for scheduling the BoT and analyzed with fast execution along with the maintaining the balanced load on the system. Algorithm 5.2 is experimented for scheduling the Workflows and analyzed with cost optimization by choosing the suitable VM from the IaaS based cloud environment.

As explained above, for the experimentation we used BoT and Workflows as input for the proposed Algorithms 5.1 and 5.2 respectively. BoT is generated using the random process and different ranges of tasks are considered for the experimentation. We considered different categories of BoT as small, medium and large, further each of the categories contains million floating point instructions per second (MFLOPS) with different ranges generated by the random process. The same set of BoT is experimented with state-of-the-art algorithms for time analysis.

Algorithm 5.2 is experimented for Workflows scheduling and later compared with state-of-the-art algorithms for cost optimization. In the experimentation, we considered two types of workflows i.e. Cybershake and Montage with a different number of tasks or nodes at different levels. The details of Cybershake and Montage workflows are as follows.

The CyberShake Workflows (Yu et al. (2005)) is used by the Southern California Earthquake Center (SCEC) to characterize the earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) technique. Figure 5.5 shows the basic flow diagram of Cybershake workflows. Strain Green Tensor (SGT) data generated from finite simulations is maintained in the form of large master SGT files for x and y dimensions. In addition to SGT data, there is a collection of estimated future fault ruptures with variations. These datasets are combined to estimate the seismic hazard at the particular site. The ExtractSGT jobs in the workflow extract SGTs pertaining to a given rupture from the master SGT files for the specific site and these are considered as data partitioning jobs. With this example, we constructed a workflow of 29 data intensive tasks or nodes which require large memory and CPU (Maechling et al. (2007)). The same workflow is experimented using Amazon EC2 instances in a heterogeneous cloud environment.

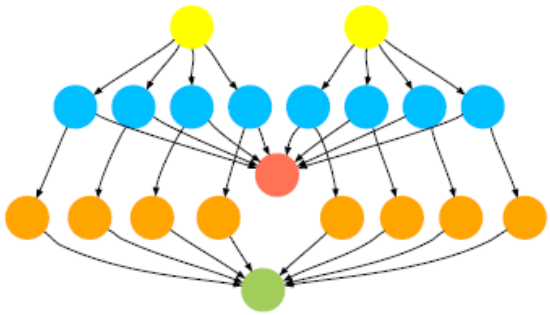


Figure 5.5: Workflows in Cybershake Application (Mirjalili et al. (2014))

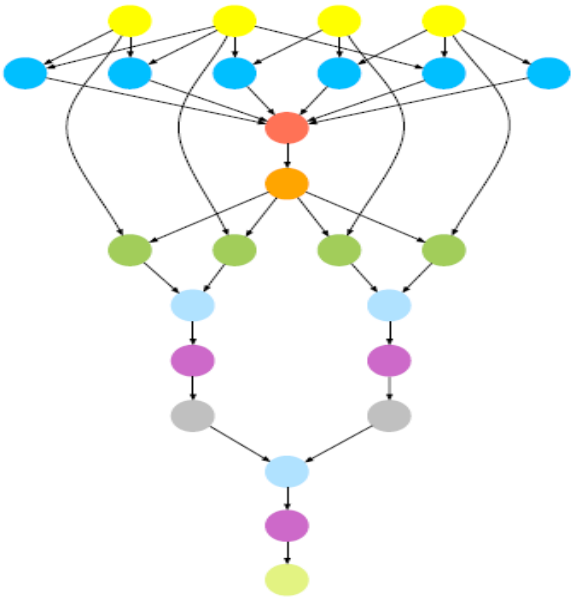


Figure 5.6: Workflows in Montage Application (Mirjalili et al. (2014))

The Montage workflow is an astronomical application used to construct the custom mosaics of the sky based on the number of high definition input images. Figure 5.6 shows the flow diagram of Montage workflows application (Yu et al. (2005)). These workflow tasks are CPU intensive and need more time to process. We experimented Montage workflow with minimum 25 tasks or nodes. Both Montage and Cybershake workflows are scheduled on the IaaS heterogeneous cloud platform. The proposed Algorithm 5.2 is experimented on these two workflows for better utilization of EC2

instances along with the reduced overall cost. The results of BoT and Workflows scheduling are as follows.

5.2.2 Results and Analysis

The proposed work is evaluated on BoT and Workflows with respect to two parameters such as execution time in terms of makespan and execution cost in terms of dollars. For the performance evaluation, the proposed algorithms are compared with Branch-and-Bound based Exact algorithm along with other state-of-the-art techniques. We also carried out statistical analysis for null hypothesis using T-Test. Further, the time complexity of the proposed GWO algorithm is also analyzed. Further, we discuss the evaluation criteria which influences our proposed GWO algorithm in optimizing the different QoS parameters.

Evaluation Criteria

For evaluation, we need to find out the makespan and critical path. Makespan gives the total time spent by the BoT during the experimentation and critical path gives the priority to execute the nodes from the Workflows. The details about makespan and critical path are given below.

Makespan: Makespan (MS) is referred to as total elapsed time required to execute the entire BoT or Workflows. Deadline (DL) is considered as a constraint measured relative to the Workflows or BoT Submission Time (S_t). Further, the makespan should not be more than deadline, i.e. (MS $\leq$ DL). Thus, makespan of the BoT or Workflows is computed by the Equation 5.2.1.

$$MP = F_t - S_t \tag{5.2.1}$$

Critical Path: Critical Path (CP) is referred to as the longest path from the start node to the exit node of the workflow. Critical path execution is used only in the Workflows execution as tasks are dependent in nature. The critical paths are separated depending on the deadline of the respective tasks in the Workflows and then execution of CP is carried out. In this work, we are not merging all CPs into a single CP instead these are evaluated separately. Thus, CP is calculated by the Equations 5.2.2 and 5.2.3 respectively.

$$DL = Nc/TaskLengthofeachTask(x) \tag{5.2.2}$$

$$EFTofCP = \sum_{X=1}^{n}[DL\sum_{X=1}^{n}Task(x)] \tag{5.2.3}$$

Table 5.2: Notations and Definitions

Notations	**Definitions**
$VMType_{(i)}$	Type of VM on which the task is being executed
Nc	Node / VM capacity
F_t	Execution Finish Time of Task
S_t	Execution Start Time of Task
Task (i)	Current Task being scheduled
CP	Critical Path
MS	Makespan
EFT	Earliest Finish Time
DL	Deadline

In the proposed work, incoming tasks, i.e. $\{x_1, x_2, x_3,..., x_n\}$ are to be scheduled on the virtual machines such as $\{vm_0, vm_1, vm_2,..., vm_k\}$. Further, cloud resources (VMs) are intelligently handled by our proposed algorithm. Table 5.2 gives the notations and its definitions used in this research contribution. The objectives of this

research work are as follows.

Objective 1: To Reduce the makespan of BoT.

$$Total\ Makespan = \sum_{X=1}^{n} F_t - \sum_{X=1}^{n} S_t \tag{5.2.4}$$

$$VM\ Type(i) = \sum_{X=1}^{n} \sum_{vm=0}^{k} [Nc_{(}vm)/Task\ Length(x)] \tag{5.2.5}$$

Total makespan is calculated by using Equation 5.2.4 and the suitable VM instance type for assigning the task is given by the Equation 5.2.5. This objective mainly concentrates on BoT scheduling and the motto is to optimize the execution time.

Objective 2: To Reduce the execution cost of Workflows.

$$Total\ Cost = \sum_{X=1}^{n} Cycles\ of\ Task(x) * cost\ of\ Nc \tag{5.2.6}$$

Total cost spent for the execution of Workflows is calculated by the Equation 5.2.6. This objective mainly concentrates on Workflows cost optimization along with balanced load on the VMs. Next, we will study the performance of the proposed algorithms on different QoS parameters.

Execution Time Analysis

The proposed Algorithm 5.1 efficiently schedules the BoT by our proposed GWO algorithm for different cases in which the task length varies from small, medium and large categories. Tasks are generated in terms of MFLOPS and will be scheduled on the VMs of heterogeneous cloud in an efficient manner. We analyzed the total execution time of state-of-the-art algorithms in comparison with our proposed GWO approach. Figures 5.7, 5.8 and 5.9 show the total execution time of state-of-the-art algorithms in comparison with the proposed GWO based heuristic algorithm.

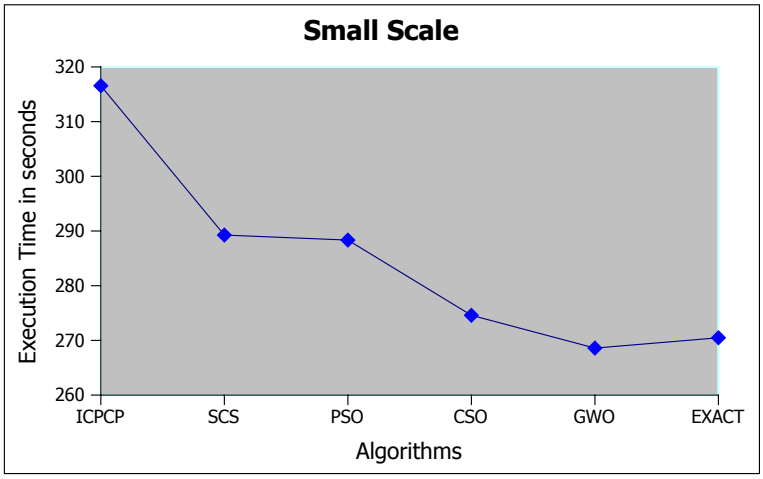


Figure 5.7: Total Execution Time with Small Scale

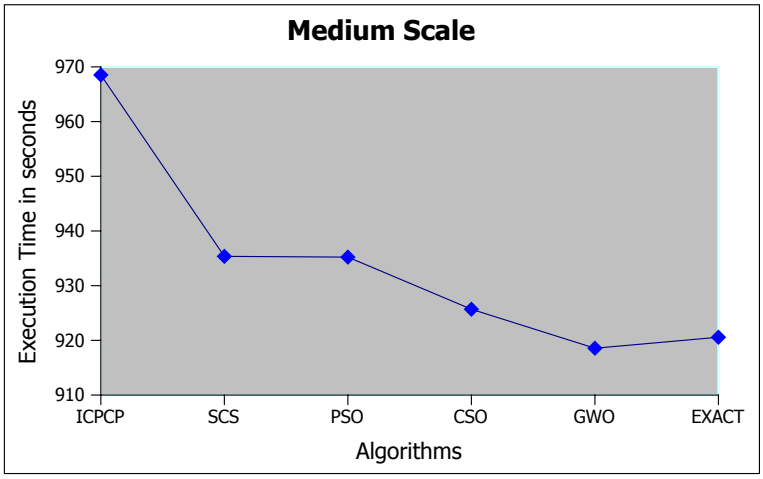


Figure 5.8: Total Execution Time with Medium Scale

From Figures 5.7, 5.8 and 5.9, it is clear that the proposed GWO algorithm takes less execution time to execute a given BoT when compared to state-of-the-art algorithms. In all categories, IaaS Cloud Partial Critical Path (ICPCP) takes much time as compared to other algorithms and it does not consider machines MFLOPS and hence the waiting time increases. Particle Swarm Optimization (PSO) algorithm takes less time when compared to ICPCP as it chooses the VM from each cluster and allocates to the first-best machine. Compared to PSO, Cat Swarm Optimization (CSO) algorithm overcomes the limitations of PSO as discussed in Chapter 4. In

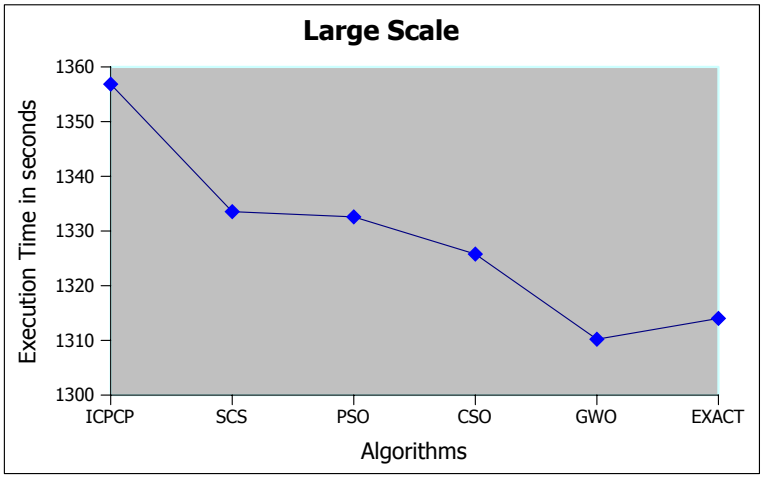


Figure 5.9: Total Execution Time with Large Scale

GWO, we introduced a new term called super α which has a capacity to choose a suitable VM depending on the various factors. Hence, the performance of the GWO is better than the state-of-the-art algorithms considered for the performance evaluation.

Cost Optimization Analysis

In this thesis contribution, we concentrated on scheduling the two workflows i.e. Cybershake and Montage by using our proposed GWO algorithm for cost optimization. Each of Workflows is experimented with different cases in which a number of nodes or tasks are varied and the results are analyzed for average makespan of Montage and Cybershake Workflows, respectively.

As we know that, Workflows have dependent tasks and deadlines to accomplish the execution. Hence, we considered the deadline management while deploying our proposed algorithm. We grouped four machines, i.e. Slow, Medium, Fast and Super-Fast (Chosen from Amazon EC2 units from Table 5.1) and found the scheduling time to fix the deadline. Figure 5.10 shows the deadline criteria of Montage and Cybershake Workflows along with the other algorithms. It is clearly observed from Figure 5.10 that, except ICPCP, all other algorithms fulfill the deadline criteria. Next, we analyze the makespan of Montage and Cybershake Workflows, respectively.

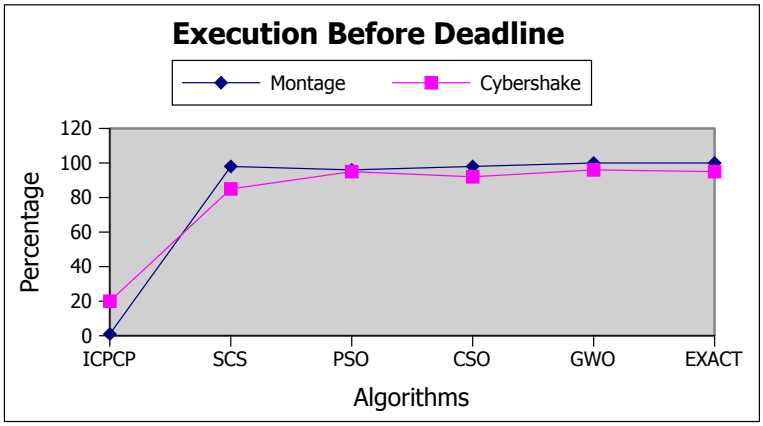


Figure 5.10: Algorithm Deadlines with Workflows

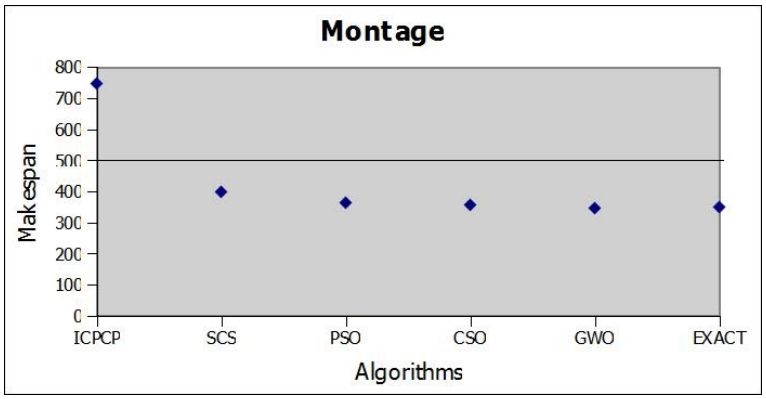


Figure 5.11: Algorithm Deadlines with Workflows

From Figures 5.11 and 5.12, it is clear that all the algorithms have different makespan w.r.t., different interval and deadline. In both Figures 5.11 and 5.12, the horizontal solid line indicates the deadline for that specific workflow. In Montage Workflow, ICPCP algorithm falls above the deadline and hence it fails to meet the deadline criteria and other algorithms lie below the deadline with different makespan.

ICPCP starts executing the critical path from exit nodes, i.e. the node which has no children. Then it immediately assigns with the available VM instance. This causes the delay in meeting deadlines and execution becomes slower for that node. SCS algorithm maps the one-to-one dependency and bundles into a single one, then starts executing the critical path. It chooses the earliest deadline first from the critical path and then starts executing. From the Cybershake Workflow, all the algorithms

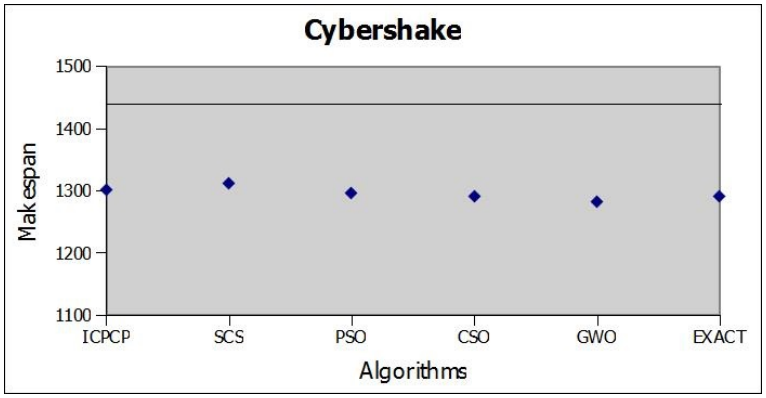


Figure 5.12: Average Makespan of Montage

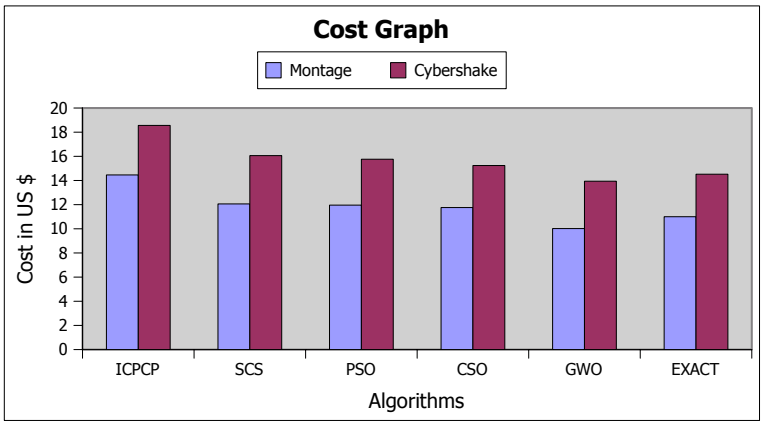


Figure 5.13: Montage and Cybershake Execution Cost Graph

lie below the deadline with varying makespan. The makespan varies because of the dynamic nature of CPU speed, resource utilization etc.

The proposed GWO algorithm executes a series of steps like finding the dependency, critical path, earliest deadline, and the load on each VM instance, last used VM instance from each search space. Then super alpha plays a vital role in scheduling and choosing the right VM for current allocation. Hence, our proposed algorithm never goes beyond the deadline with efficient resource utilization in heterogeneous IaaS cloud platform as compared with other conventional algorithms. Next, we will analyze the cost incurred during the Workflows execution from our proposed GWO and other state-of-the-art algorithms.

For the cost optimization, we considered Amazon EC2 instance pricing model i.e. in dollars. From Figure 5.13, it is clear that even though ICPCP algorithm did not meet the deadline criteria as shown in Figure 5.10 and also it takes the maximum cost for the execution of both Montage and Cybershake workflows. All other algorithms execute the workflow within its given deadline interval and also tries to minimize the execution cost. Scaling Consolidation Scheduling (SCS) is much better than ICPCP and executes in a feasible amount of time. The other Bio-Inspired heuristic algorithms such as PSO, CSO and proposed GWO execute in an optimized way w.r.t. makespan and cost. There is no significant difference between the performance of PSO and CSO w.r.t cost optimization.

The Exact algorithm tries to follow the best critical path (By considering deadline and machine capability) and hence takes more cost when compared to proposed GWO. Further, our proposed algorithm may take slightly high time in computation, but chooses the optimized VM instance for execution. Hence, the proposed algorithm outperforms the other conventional and Bio-Inspired heuristic algorithms with respect to the fast execution and cost optimization.

Statistical Analysis on Workloads

In the cloud environment workloads will vary depending on the applications. The real time applications need resources with no delay and cognitive applications need more number of VMs with dedicated cores in them. Here, for the performance analysis of the proposed algorithms, we followed the same statistical analysis on the workloads as we did in Section 4.8.2 of Chapter 4. We followed the Table 4.6 of Chapter 4, which gives the Variance, the Standard Deviation (SD), the Confidence Level (CL) and the Accuracy of the Workloads (1K, 2K and 3K) taken for the experimentation. Our results show that, irrespective of changes in the Variance, the Standard Deviation and the Confidence Level, the Accuracy results will not deviate and thus achieving the consistent performance.

Table 5.3: Results of T-Test Analysis

Algorithms	**P-Values (Montage)**	**P-Values (Cyber-shake)**
ICPCP - GWO	0.0001	0.0001
SCS - GWO	0.01	0.001
PSO - GWO	0.007	0.0001
CSO - GWO	0.007	0.0001
Exact- GWO	0.00069	0.0001

Statistical Hypothesis Analysis

In this Chapter, we followed the same statistical hypothesis approach as we did in Section 4.8 of Chapter 4. Here, we made statistical analysis by evaluating the proposed GWO algorithm in comparison with state-of-the-art and bench-mark algorithms. For statistical hypothesis, we used T-Test analysis for scheduling and cost optimization. The results of the T-Test analysis for both Montage and Cybershake Workflows are shown in Table 5.3.

In the null hypothesis analysis, the threshold value of the P or Significance Level of is considered. In this experiment, we considered $\alpha = 0.05$ which is the standard cutoff for the null hypothesis. The P-value of proposed GWO algorithm in comparison with other algorithms is less than $\alpha = 0.05$ and thus it rejects the null-hypothesis. Hence, our proposed GWO algorithm is more efficient when compared to all other algorithms considered for cost analysis.

Time Complexity Analysis

In both algorithms (Algorithms 5.1 and 5.2), we considered the heterogeneous VMs hosted in the S_z clusters with n number of tasks arriving into the cloud. The proposed GWO algorithms schedule the incoming tasks in an efficient way on the heterogeneous

VMs and execute them by optimizing the cost. Hence, the time complexity of the proposed GWO Algorithm 5.1 is $O(n.S_z)$ since S_z is a constant and hence the complexity of Algorithm 5.1 will be $O(n)$. In Algorithm 5.2, the minimum time complexity will be $O(n^2.S_z)$ as the algorithm iterates for critical path scheduling of the dependent tasks in the Workflows scheduling. Since S_z is a constant and hence, the final time complexity of Algorithm 5.2 is $O(n^2)$.

Summary

In this Chapter, we proposed GWO based algorithms for scheduling the BoT and Workflows. Two QoS parameters considered in this thesis contribution are mainly Time and Cost. Even though our proposed GWO takes slightly high computation time in some cases, but always provides the cost-effective solution. Hence, the proposed GWO based scheduling algorithms outperform all other algorithms considered in terms of better execution time and cost. Further, statistical analysis on workloads shows that, irrespective of changes in the Variance, the Standard Deviation and the Confidence Level, the Accuracy results will not deviate and thus achieving the consistent performance.

In the next Chapter, we propose cost optimized scheduling of the tasks by choosing the VM instances using machine learning techniques (On-Demand and Spot instances). The main focus will be on choosing different types of instances for cost optimization.

Chapter 6

Cost Optimized Scheduling of Spot Instances at Cloud Data Center

VM instances play an important role in executing the tasks in the cloud environment. There are many types of instances offered by different cloud vendors in the market. These VM instances are procured by the clients in many ways to accomplish the tasks which are submitted to the cloud environment. In this thesis contribution, we mainly focused on cost optimization by using the VM instances as efficiently as possible using machine learning techniques.

Here, we considered two types of instances, i.e. On-Demand and Spot instances from the Amazon cloud market. Basically, we need to schedule the tasks on given instances so that we can execute them in a given deadline and we consider MPSO based scheduler which is described in detail in Chapter 4. Further, we use Neural Network (NN) technique to predict the future Spot values so that we can use Spot instances rather than On-Demand instances. Hence, the key contributions of this chapter towards cost optimized scheduling of Spot instances are as follows:

- Prediction of future values of Spot instances by Neural Network based Back Propagation algorithm which utilizes the past Spot pricing history data.
- Migration of VMs from one PM (overloaded) to another PM (underloaded) depending on the cloud resource requirements in terms of CPU and MEMORY.

- Performance analysis of proposed cost optimized scheduling for Spot instances in heterogeneous cloud environment using Spearman's Rho test.

Here, we follow the proposed MPSO task scheduling approach on VMs instances which is described in Chapter 4. The type of instances play a major role is scheduling and as well as cost optimization. We used a NN technique to train the past Spot values so that the next VM instance can be bought at a lesser price. The details of the VM instances and training are discussed in the following sections.

6.1 Proposed Methodology

We used our proposed efficient task scheduling algorithm using Modified Particle Swarm Optimization Technique on VMs instances (Mainly Spot and On-Demand). Further, we used Neural Network based back propagation algorithm for predicting the Spot-instance values for overall cost optimization in comparison with On-Demand instances in the heterogeneous cloud environment. Hence, we will be discussing more on Spot instance prediction and cost optimization rather than scheduling using MPSO.

6.1.1 Types of Instances

In this work, we adapt two types of instances with variable price structure from the Amazon cloud model. The details of two pricing models are as follows.

A. On-Demand Instances: In this type of instance, the user has to pay hourly based on the instance type. Here, the prices for different instances are fixed and the same has to be paid by the client (user) and the price will not vary once the instance starts executing the task. All the instances follow hourly pricing model, i.e. once the instance is given, the user has to pay for one complete hour even though users use the cloud resources even for a minute (On-Demand (url)).

B. Spot Instances: In this type of instance, the users bid for a particular instance and it is made available as long as their bid is higher than the Spot price. Spot prices are fixed by the cloud service provider which is dependent on the instance type and its demand. Spot instances can change during the instance runtime and change dynamically (Javadi et al. (2011)). The Spot price of an instance varies with time and it is different for different instance types. The price also varies between regions and available zones. Here, the user has to participate in bidding (auction) to get the Spot instance and the user can bid for the maximum price that can be paid by the user. Number of bidders also vary depending on the time and region.

The user is provided the resource/instance, whenever the bid is higher than or equal to the Spot price (Cloud (2011)). The demerit of the Spot instance is that whenever the Spot price becomes higher than the current user bid, then Amazon terminates the Spot instance resource and assigns it to the new user who has the highest bid value. When the Spot instances are given back to the cloud provider, then the user has to pay the Spot price that is applicable during the start time. Further, the user will not be charged if the service is terminated by the cloud service provider. However, the user has to pay for a full hour if the Spot instances are abruptly terminated by the user (Yi et al. (2010)).

6.1.2 Proposed Work

In this thesis contribution, we used MPSO technique for scheduling the tasks on the VM instances. Both Spot and On-Demand instances are used during the scheduling. Further, with the NN training model we used Spot instances and later these predicted instances are replaced by On-Demand instances for the purpose of cost optimization. Figure 6.1 shows the overall flow diagram of proposed work.

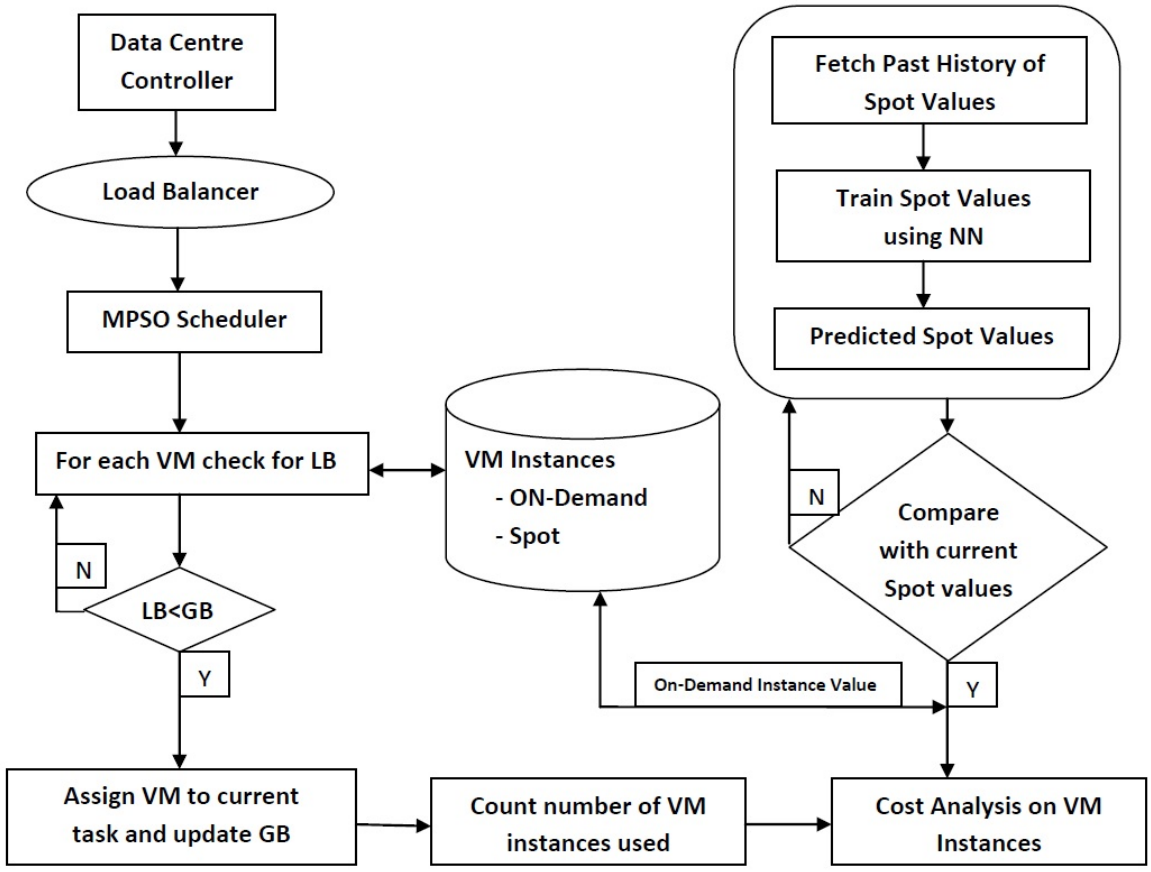


Figure 6.1: Overall Flow Diagram of Proposed Work.

Here, we implemented a scheduling algorithm using On-Demand and Spot instances. The main motto is to utilize the cloud resources in an efficient and intelligent manner. Further, the cost analysis for executing the BoT is evaluated using both On-Demand and Spot instances. It is observed from Figure 6.1 that the tasks in the BoT are coming from cloud users and these are received by cloud Data Center. Further, Load Balancer will choose the suitable Scheduler (using MPSO) for assigning the incoming tasks of BoT to the suitable VM instances (On-Demand and Spot).

Spot instance values are chosen from the past history of a specific region and then these values are used by the Back Propagation algorithm of NN for predicting the future Spot values. New Spot values are compared with current Spot values

for validation. Next, total number of VM instances used for executing the BoT are considered in cost analysis. Then, cost comparison between On-Demand and predicted Spot instances are carried out for performance analysis on cost. Hence, the objectives of the proposed work are as follows.

Objective 1: Efficient Utilization of VMs.

$$VM\ \ Type(i) = Number\ of\ tasks\ handled\ by\ VMs. \tag{6.1.1}$$

VM Type(i) refers to a type of VM used in the experiment. We used six different instances of VMs (T1.micro, M3.large, M3.xlarge, M1.small, M3.2xlarge, M4.4xlarge) and more details are given in Table 6.1.

Objective 2: Minimizing the Execution Cost of BoT.

Each Spot and On-Demand instances has its own price depending on the compute capacity (configuration of VM instance). On-Demand instance prices are static whereas Spot instance prices are dynamic. For cost analysis, we took the number of machine cycles of tasks along with VM instance price at that time interval. The total execution cost of BoT is defined as

$$TotalCost = [\sum_{x=1}^{n} Cycles\ of\ Tasks(x) * cost\ of\ VM\ Instance] \tag{6.1.2}$$

6.1.3 Spot Instance Training and Prediction

In this work, the main focus is on utilizing more Spot instances rather than On-Demand instances. To schedule these instances, we consider proposed MPSO scheduling technique. Spot Instance values change dynamically and on the other hand, On-Demand instance values are static for a particular interval of time. Hence, we use the Neural Network based Back Propagation algorithm for predicting the future Spot

instance values based on the past history of Spot instance values over a period of time. Figure 6.2 shows the complete flow diagram of Spot instances training using a NN Back Propagation algorithm (Wong and Nandi (2004)) for cost optimization.

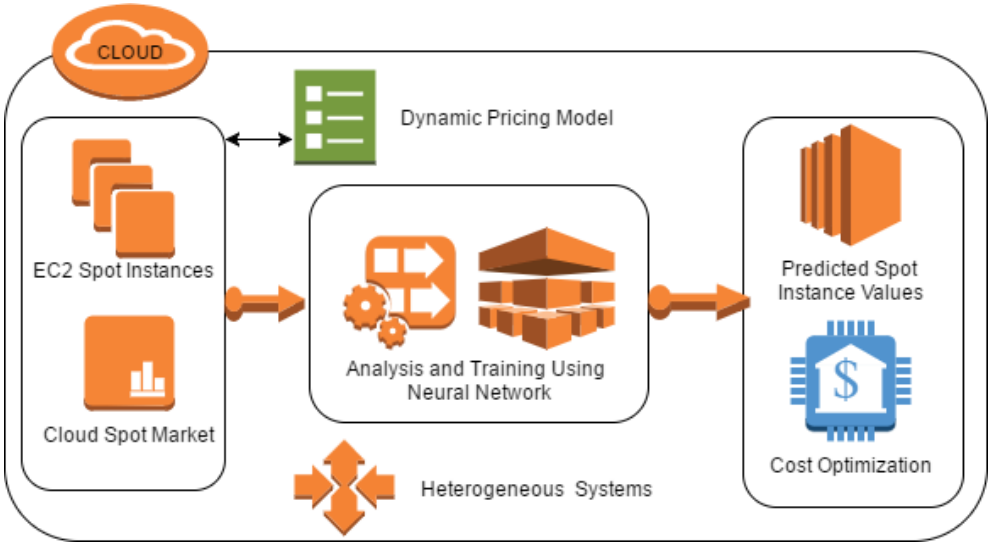


Figure 6.2: Block Diagram of Spot Instance Training and Cost Analysis

Cloud Spot Market: Different cloud vendors offer many types of Spot instances for the clients. Spot Market generally consists of different instances. In this thesis, we consider Spot and On-Demand instances from the Amazon cloud market.

EC2 Spot Instances: These are the Spot instances offered by Amazon cloud and these instances price change dynamically.

Dynamic Pricing Model: On-Demand instance values are fixed and they do not change suddenly. On the other hand, Spot instance values are dynamic in nature and its value depends on the user bid.

Heterogeneous Systems: In this thesis, we consider the experimental setup in a heterogeneous fashion. The systems have different configurations and we also use hot plugging technique to add required resources.

Analysis and Training Using NN: Spot instance values for the past 60 days are considered from a specific region and fed as input to NN training model. The predicted Spot instance values are tested for future Spot instance values. Our proposed NN

model works efficiently in predicting the Spot values.

Predicted Spot Values: After training, the predicted Spot values are compared with present Spot market values for validation.

Cost Optimization: Predicted Spot instance values are compared with On-Demand instance values and our proposed model insists to use more Spot instances so that the cost can be reduced considerably.

These predicted Spot instance values are validated with future Spot values and then we can easily bid for the Spot instances. In our model, BoT is considered for scheduling on VM instances, i.e. On-Demand and Spot. Further, with our prediction model, we can use Spot instances for executing the tasks rather than using On-Demand instances. Using the Spot instances, we can reduce the execution cost considerably in comparison with On-Demand instances. The details of performance evaluation in terms of experimental setup and results are as follows.

6.2 Performance Evaluation

In this Chapter, the tasks from the BoT are scheduled on the VMs instances (On-Demand and Spot) using Bio-Inspired MPSO scheduling algorithm. Further, we proposed NN based algorithm for predicting the future Spot instances. We conducted the experiment for performance evaluation and the details of experimental setup are as follows.

6.2.1 Experimental Setup

In this work, we used two Physical Machines (PMs) for experimentation. As we discussed in the previous chapters, BoT acts as an input to the cloud data center, and these tasks are to be scheduled on Amazon VM instances (On-Demand and Spot). The tasks in the BoT are categorized into different classes as Small scale, Medium

Table 6.1: Spot Instance Characteristics with US East N. Virginia Region

Instance	**V-CPU**	**Mean**	**Standard Deviation**	**Max**	**Min**
T1.micro	1	0.0075	0.00636	0.012	0.003
M3.large	2	0.0215	0.00354	0.024	0.019
M3.xlarge	4	0.0685	0.04455	0.100	0.037
M1.small	1	0.0075	0.00071	0.008	0.007
M3.2xlarge	8	0.309	0.31537	0.532	0.086
M4.4xlarge	16	0.537	0.59538	0.958	0.116

scale and Large scale. Each scale has tasks with different MIPS (Million Instructions Per Second). On a small scale, MIPS varies from 500 to 1000. With Medium scale, MIPS varies from 1000 to 5000. In Large scale, MIPS is above 5000. We considered different combinations of tasks so that efficient scheduling of these tasks can be done on the suitable VM instances.

As said earlier, we considered two PMs and the performance analysis is carried out in terms of Central Processing Unit (CPU) and Random Access Memory (RAM) utilization of both PMs. Along with the efficient utilization of the cloud resources we ensure that load on both PMs is in a balanced manner. For balanced load, we migrate tasks from overloaded PM to underloaded PMs. We used simple migration policy, i.e. tasks from overloaded PM are migrated to underloaded PM. Migration takes a feasible amount of time, but our main goal is to find the efficient utilization of CPU and RAM from both PMs even after migration of tasks.

The proposed work is experimented on the customized simulation environment written in Python language and also in CloudAnalyst tool from ClouSim. The complete details of the setup are given in Section 4.8 of Chapter 4. Further, we will discuss the types of Spot instances used from Amazon cloud and the details are given in Table 6.1 (Spot (url)).

We considered 6 different mixtures of VM instance types and their respective Spot values for 60 days from US East N. Virginia Region. Table 6.1 gives the details of Spot instance types, values of CPU, mean, Standard Deviation (SD), Minimum and Maximum Spot value of each VM instance considered during the experiment.

6.2.2 Results and Analysis

The proposed work is evaluated using On-Demand and Spot Instances on BoT with respect to four parameters such as average response time, execution time, execution cost and cloud resource utilization (CPU and RAM) with and without VM migration in heterogeneous cloud environment. In this work, we also predict the future Spot values so that efficient, cost management can be achieved. The details of performance analysis of proposed work in terms of QoS parameters are as follows.

Bag-of-Task Scheduling Analysis

As discussed earlier, the tasks in the BoT are scheduled on VM instances using our proposed MPSO algorithm. In this chapter, we focus on the performance analysis of the MPSO algorithm on two parameters, i.e. average response time and execution time. The MPSO algorithm is proposed in Chapter 4 but here it is analysed on different combinations of BoT. Here, the tasks in the BoT are categorized into Small, Medium and Large levels of tasks with different MIPS.

Figure 6.3 shows the average response time and Figure 6.4 shows the execution time analysis of the proposed algorithm in comparison with peer research algorithms. Both Throttled and Round Robin algorithms take almost the same execution time since MIPS values are smaller in Small scale and hence there is no significant difference. In ACO, the choosing of VM instances depends on the pheromone content. Usually in ACO, the initially considered VM instances are reused as they accumulate the pheromone content rather than other VM instances. Further, if the MIPS value

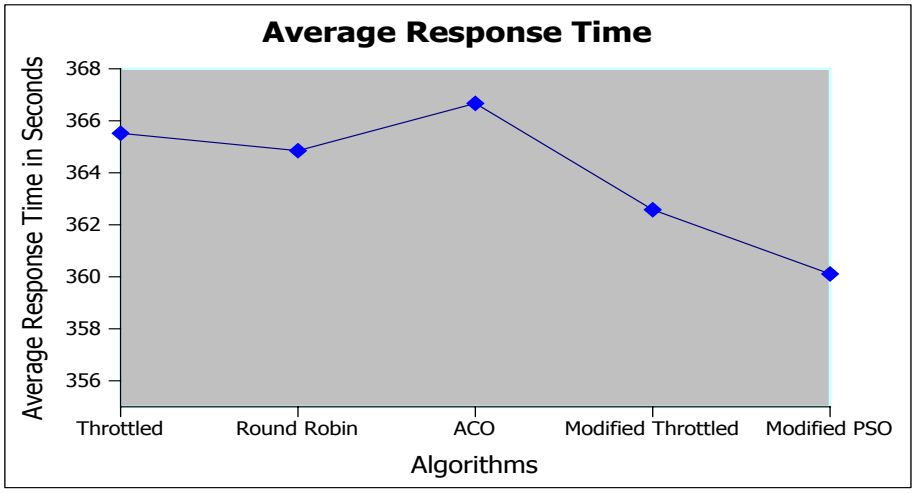
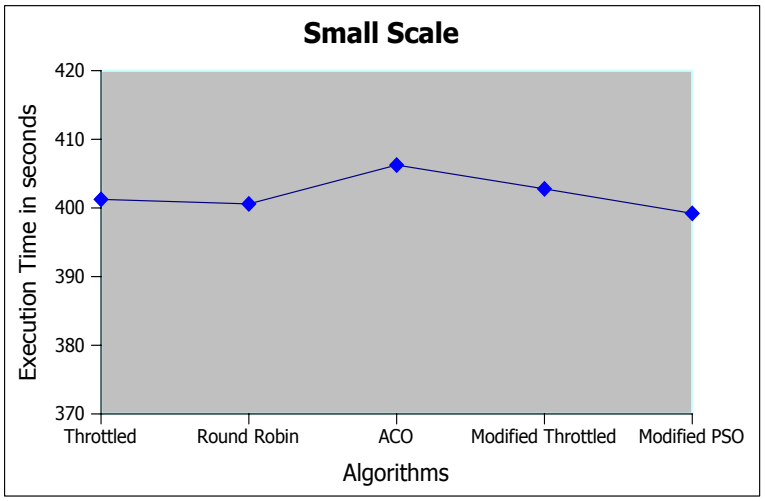


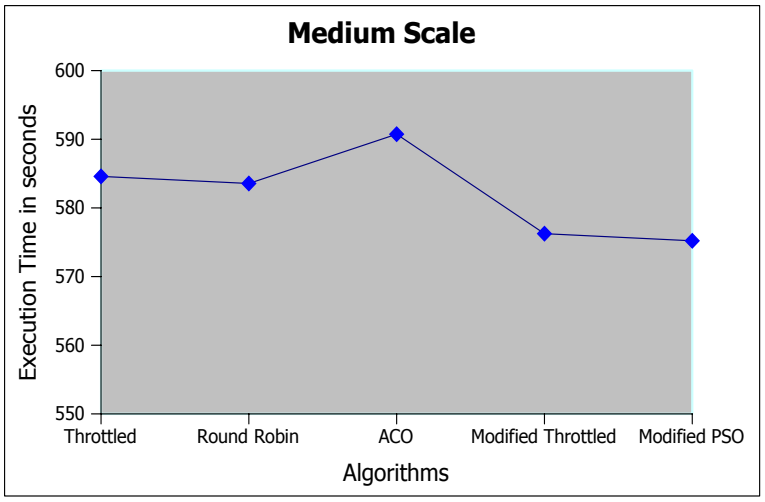
Figure 6.3: Average Response Time Analysis

is more and capacity of VM instance is less, then the attached VM instance takes more time when compared to other high capacity VM instances. Hence, in all three cases (Small, Medium and Large), ACO takes more execution time when compared to peer research algorithms. Modified Throttled algorithm is efficient when compared to Throttled algorithm as it does not start from the first index during scheduling of VM instances and our proposed MPSO chooses the efficient VM instance from each cluster with less execution time and thus outperforms all peer research algorithms considered for performance evaluation.

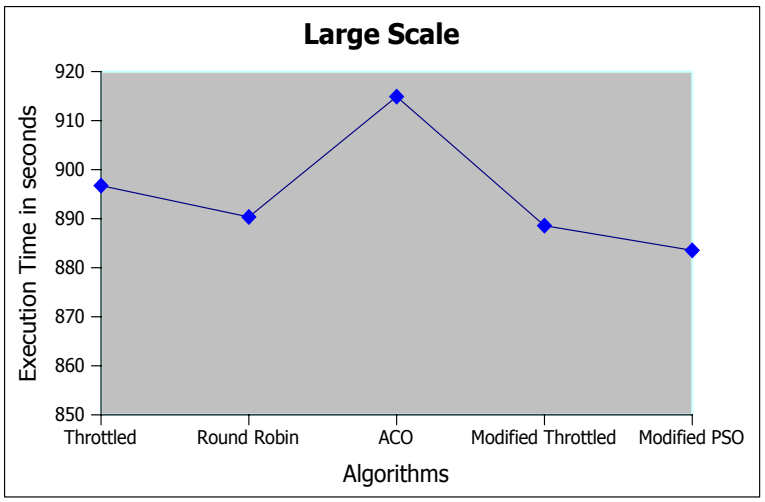
The proposed MPSO algorithm gives better results by checking the state of the VM and it has cluster based comparison of allocating tasks to a VM and thus avoids the server queuing. Cat Swarm Optimization (CSO) takes more time when compared to the proposed MPSO as CSO combines two modes (Seeking and Tracing mode) and hence more time is consumed in tracing mode, hence CSO is not considered for task scheduling. Genetic Algorithm (GA) takes more time during chromosome crossover and mutation operations and hence we did not consider both CSO and GA in our performance evaluation.



(a) Execution Time Analysis of Small Scale BoT



(b) Execution Time Analysis of Medium Scale BoT



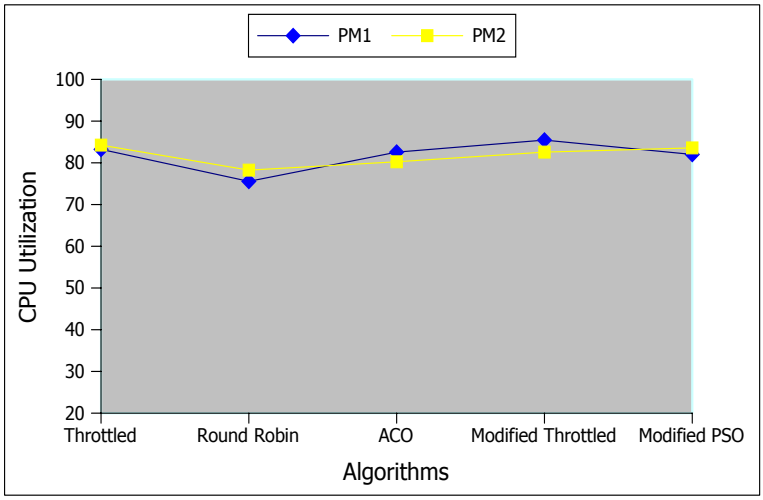
(c) Execution Time Analysis of Large Scale BoT

Figure 6.4: Execution Time Analysis

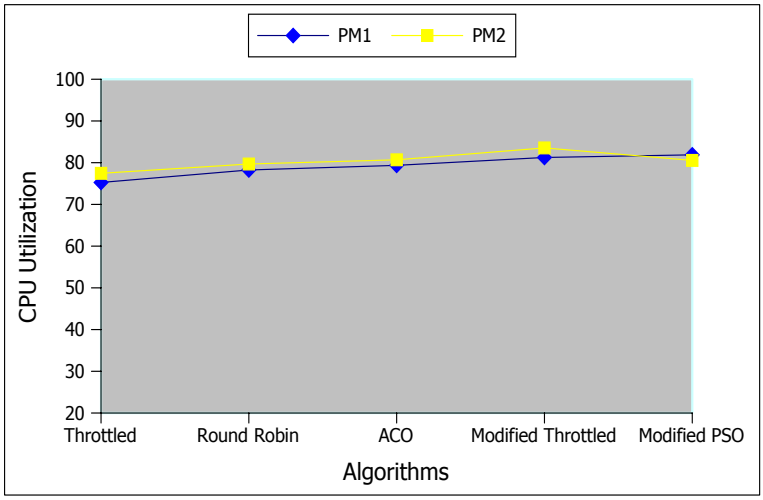
Utilization of Cloud Resources (CPU and RAM)

In the proposed work, the analysis is done on efficient utilization of cloud resources, accordingly we evaluate the CPU and RAM utilization on two PMs. Next, we analyze each PM's average CPU and RAM utilization along with the VM migration. The VM will be migrated from the one PM (overloaded) to another PM (underloaded). Then the VM stops executing the task and migrate it to another VM and it starts executing from the beginning at another PM (underloaded). Further, if the assigned VM in a PM (overloaded) does not start its execution in 30 seconds, then it starts to migrate to another PM (underloaded) for execution. Next, we will analyse the CPU and RAM utilization of PM1 and PM2 respectively.

Figures 6.5a and 6.5b show the average (small scale, medium scale and large scale) CPU utilization of PM1 and PM2 without and with VM migration respectively. From Figure 6.5a, it is clear that almost all scheduling algorithms utilize the CPU in an efficient manner. Our proposed MPSO also utilizes the available CPU in an efficient manner. But ACO underutilizes the CPU as it spends more time in assigning the task to VM and thus causes the delay in executing a request. Usually when compared to ideal scenario, the CPU utilization is more and this is because of tasks belonging to the Large Scale category with high MIPS values. From Figure 6.5b, it is clear that the utilization of PM2 is slightly more than PM1. During execution of BoT, more VMs are migrated from PM1 to PM2 and hence PM2 is more utilized.

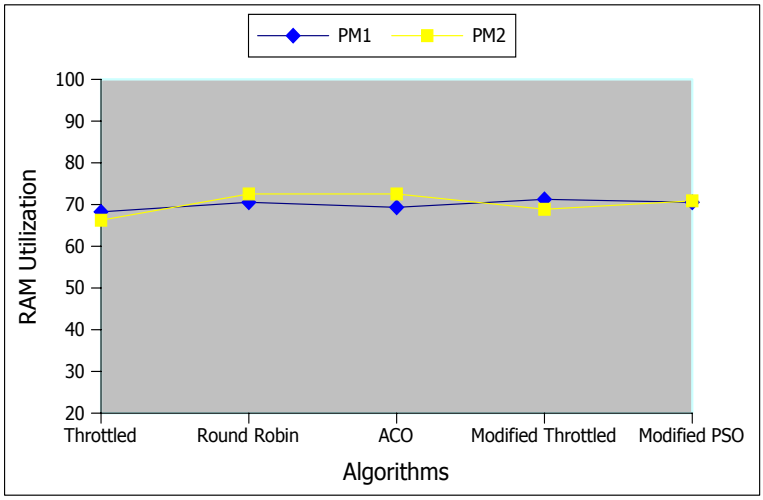


(a) CPU Utilization of PM1 and PM2 without Migration

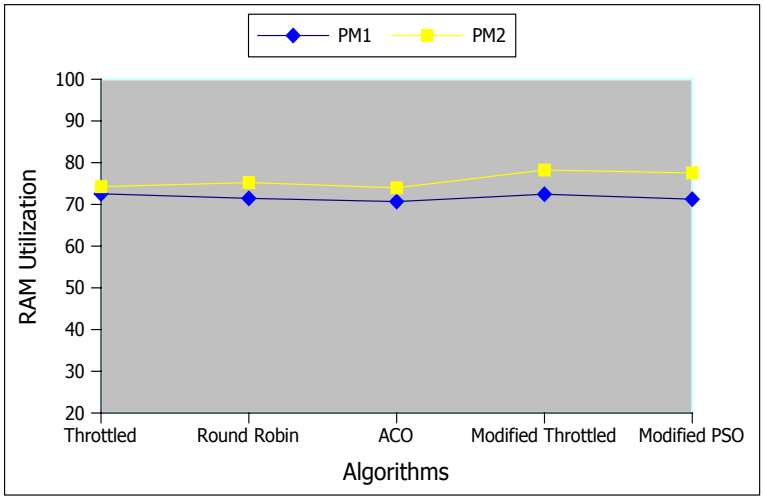


(b) CPU Utilization of PM1 and PM2 with Migration

Figure 6.5: CPU Utilization of PM1 and PM2



(a) RAM Utilization of PM1 and PM2 without Migration



(b) RAM Utilization of PM1 and PM2 with Migration

Figure 6.6: RAM Utilization of PM1 and PM2

Figure 6.6a and 6.6b show the average (small scale, medium scale and large scale) RAM utilization of PM1 and PM2 without and with VM migration respectively. From Figure 6.6a, it is clear that proposed MPSO utilized almost same amount of RAM of both PM1 and PM2. But when compared to Figure 6.6b, RAM utilization is increased due to VM migration from PM1 to PM2. When a VM is about to migrate then its state will be stored in the temporary storage and thus the RAM will be more occupied than usual scenario. Hence, RAM utilization of both PM1 and PM2 is more.

Table 6.2: Predicted and Actual Spot Instance Values

VM Instance Type	**Predicted Spot Value**	**Exact Spot Value**
T1.micro	0.055	0.06
M3.large	0.0175	0.018
M3.xlarge	0.0568	0.058
M1.small	0.0085	0.007
M3.2xlarge	0.201	0.23
M4.4xlarge	0.175	0.198

However, RAM utilization of PM2 is more than that of PM1 as more number of VMs are migrated from PM1 to PM2.

Execution Cost Analysis using On-Demand and Spot Instances

Here, the analysis is done on the execution cost using On-Demand and Spot instances using our proposed MPSO based scheduling algorithm on Small, Medium and Large Scale BoT. As explained, we used Neural Network back propagation algorithm to predict the future Spot instance values and accordingly we chose the suitable Spot instances rather than On-Demand instances to reduce the total cost.

Table 6.2 shows the predicted Spot values using Neural Network algorithm and these are compared with exact Spot values for accuracy. From Table 6.2, it is clear that for all types of instances, the predicted Spot value is almost similar to the exact Spot value. Hence, there is a high chance of getting the Spot instances, if we bid just above the predicted Spot values. By doing this we are able to get the Spot instances for the execution of the BoT and thus cost can be reduced. Further, we analyzed how cost is reduced if we use only Spot instances rather than On-Demand instances for the execution of different categories of the BoT.

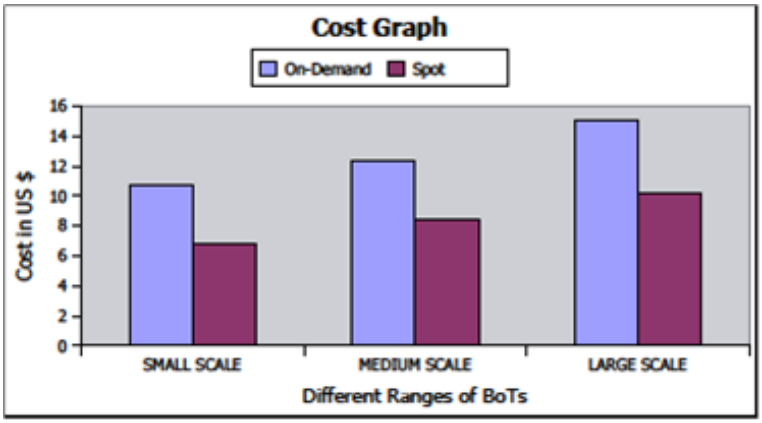


Figure 6.7: Cost Analysis of On-Demand and Spot Instances

Figure 6.7 shows the cost analysis of Small, Medium and Large Scale categories of BoT when these are executed on On-Demand and Spot instances. For the cost analysis, we have considered Predicted Spot values by assuming that these are available until the execution of assigned tasks from the BoT is completed. As mentioned earlier, tasks from different categories of BoT consist of variable MIPS for the execution. From Figure 6.7, it is clear that for different categories of BoT, the cost of On-Demand instances is more when compared to that of Spot instances. Before prediction of Spot values, we are supposed to consider mixing of On-Demand and Spot instances and we are not sure whether Spot instance remains unchanged until the execution of tasks on the BoT is completed or not. After predicting the Spot values, we are able to use Spot instances rather than On-Demand instances by bidding them at higher value than the predicted value and thus we ensure that Spot instance does not rollback until it completes the execution of a task. It is clear that if we use Spot instances rather than On-Demand instances, we can reduce the cost by 38.23% during execution. Hence, our proposed scheduling algorithm and prediction model not only utilizes the cloud resources (CPU and RAM) efficiently but also reduces the cost considerably.

Further, we carried out statistical analysis on predicted and actual Spot instances using Spearman's Rho test. Since the results obtained are in the form of ordinal and hence, we used Spearman's Rho test (Tayyab et al. (2016)).

Statistical Analysis

In the statistical analysis, two entities are evaluated about a given workload or population so that we can determine the best supported entity for different cases. Further, Spearman's Rho test based statistical analysis is carried out between the predicted and actual Spot prices. The Spearman's Rho is a non-parametric test used to measure the strength of association between two entities, where the value of R = 1 means a perfect positive correlation and the value of R = -1 means a perfect negative correlation. In this experiment, the R value is 0.94286, i.e. the actual and predicted values are perfectly correlated with the error rate of 5.714 %.

6.3 Summary

In this thesis contribution, the tasks of BoT are scheduled using MPSO algorithm on different VM instances offered by Amazon. The main focus is on using Spot instances rather than On-Demand instances for cost optimization. We proposed a NN based model for predicting the Spot instances, which utilizes the past history from a specific region. Further, statistical analysis on workloads shows that, irrespective of changes in the Variance, the Standard Deviation and the Confidence Level, the Accuracy results will not deviate and thus achieving the consistent performance on different QoS parameters. Our predicted Spot values are compared with future Spot values for performance evaluation. Hence, if we use Spot instances rather than On-Demand then we can significantly reduce the cost incurred during the execution of tasks.

Chapter 7

Testing and Validation

In this chapter, the main focus is on testing and validation of our proposed scheduling, resource allocation and management algorithms using the real cloud platform. In Chapter 3 and 4, we proposed scheduling, resource allocation and management algorithms which were tested on both CloudSim and PySim customized simulators. Further, the simulation results demonstrate that the proposed algorithms outperform the state-of-the-art algorithms. Now, our proposed algorithms are validated using the IBM cloud setup. Hence, the main contributions of this chapter are as follows.

- Testing of the proposed algorithms such as MPSO, MCSO and HYBRID (MPSO+MCSO) on customized cloud setup called PySim.
- Further, validation of aforementioned algorithms (MPSO, MCSO and HYBRID (MPSO+MCSO)) using the IBM cloud setup.

The proposed algorithms are already explained in detail in the Chapters 3 and 4. Here, the results of the proposed algorithms are tested and validated using the IBM cloud setup and thus demonstrated that these algorithms play an important role in efficient scheduling, resource allocation and management of cloud resources.

7.1 Validation of Proposed Algorithms

As mentioned above, the proposed algorithms are tested and validated using IBM cloud data center environment. During validation, our algorithms faced few challenges, i.e. comparison between simulation environment and a real cloud setup is a complex process. Next, the proposed algorithms are briefly discussed along with validation challenges followed by results and analysis.

7.1.1 Modified Particle Swarm Optimization

The proposed MPSO algorithm is used to schedule the incoming tasks on the VMs in a heterogeneous cloud environment. For the scheduling, we used different categories of BoT with varied number of MIPS. The algorithm gives better efficiency in the customized cloud environment using PySim and then the results are validated with IBM cloud setup.

In the IBM cloud, we followed the same setup with the same process which we implemented in customized cloud setup and there are a few challenging issues (network connectivity and synchronization). Then, we successfully fixed these issues and found that the proposed MPSO algorithm worked efficiently in a real cloud environment. Further, we will study an example for explaining our IBM cloud setup and then we will explain another algorithm, i.e. HYBRID (MPSO+PCSO) which is also tested and validated on the IBM cloud setup.

7.1.2 Real Cloud Setup

Figure 7.1 shows the complete flow of the proposed algorithms along with the experimental setup using the IBM cloud. Figure 7.1 has two PMs which can host a number of VMs and the cloud also supports auto scaling which is an important property of dynamic cloud. The block diagram contains scheduler in which we embedded our

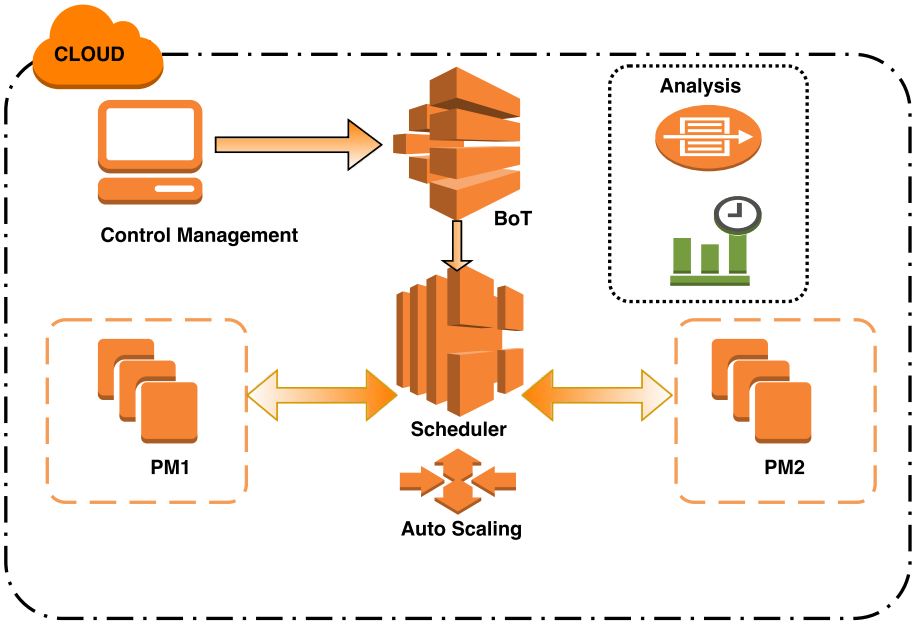


Figure 7.1: Block Diagram of Real Cloud Setup

proposed algorithms. The control management gives the load, i.e. BoT are fed into the cloud which is in turn received and managed by the scheduler for processing. As discussed earlier, different categories of BoT are taken for the validation. The top right block does the analysis part in which we keep track of resource utilization as well as average response time of BoT. The results of this experiment are compared with the simulated results from PySim for validation. The complete details are given in the following Section.

There is no much deviation in the results obtained by IBM machines when compared to the simulation results of the PySim based cloud setup. Table 7.1 gives the configuration details of the IBM cloud setup used for the experiment. The servers used in the IBM cloud setup are of different configurations such as one server has 3067 MIPS with 2 cores and storage capacity of 250 Giga Bytes (GB) along with 8 GB RAM. Similarly, another server has 4025 MIPS with 4 cores and 500 GB of storage along with 16 GB RAM. Both servers can host several VMs of different configurations in the given cloud setup.

Table 7.1: Configuration Details of IBM Cloud Setup

Machine Type	**CPU CORES**	**MIPS**	**RAM**	**Storage(GB)**
x3100M3E5540	2	3067	8	250
x3400M2EX5570	4	4025	16	500

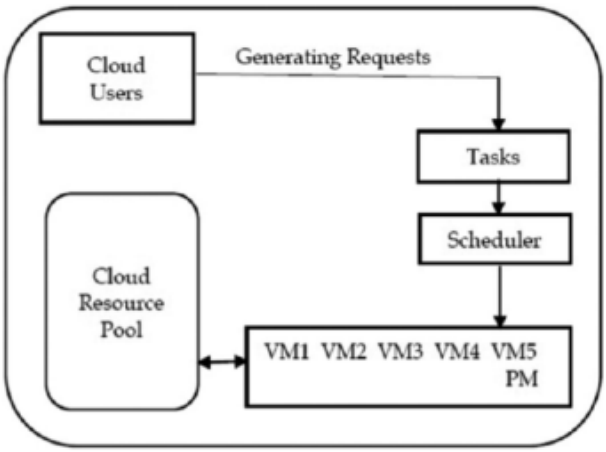


Figure 7.2: Model for Explaining Our Proposed Work

7.1.3 Resource Allocation and Management Using HYBRID (MPSO+MCSO) Bio-Inspired Algorithm

The proposed HYBRID (MPSO+MSCSO) algorithm combines the approaches of MPSO and MCSO and applies them parallel in the hybrid approach. As discussed in Chapter 4, the proposed HYBRID algorithm is more efficient than MPSO, MCSO and outperforms the other state-of-the-art algorithms. Now, we will quickly recap the proposed HYBRID algorithm along with a small example for better understanding of its overall flow.

Figure 7.2 shows an example for explaining our proposed work. The incoming task demands n number of cloud resources where n = *[0,9]* and allocation of *'n'* resources are provided by either cloud resource pool or by the VMs. Each cluster contains different VMs with the best two n resource values represented as e*excess_res1* and *excess_res2*, respectively, and the remaining resources available with VMs are stored in *excess_res3[]*. During resource allocation, the on-demand resources are compared with

excess_res1 and *excess_res2* and *excess_res3[]*, respectively by our proposed MPSO, MCSO and HYBRID (MPSO+MCSO) algorithms. The *excess_res1* and *excess_res2* mappings are used by the MPSO algorithm where as *excess_res3[]* mapping is used by the MCSO algorithm. On the other hand, HYBRID (MPSO+MCSO) algorithm uses all the three aforementioned resource mappings*[]*.

7.1.4 Cloud Orchestration

Cloud orchestration is the use of programming technology (even scripting) to manage the interconnections and interactions among workloads on public and private cloud infrastructure. It connects the incoming tasks into a cohesive workflow to accomplish a goal, without violating the service level agreements.

Cloud orchestration is typically used to provision, deploy or start servers; acquire and assign the storage capacity; manage the networking; create the VMs; and gain access to the specific software or task on the cloud services. Every job execution in the cloud orchestration is accomplished by three main entities namely: service, workload and resource modules from the orchestration framework. An orchestration platform can integrate the permission checks for security and compliance.

Cloud orchestration technology must work with heterogeneous systems, potentially servicing a global cloud deployment in different geographical locations and with different providers. Many cloud orchestrator users run public cloud and private deployments.

Two major components of cloud orchestration are:

Software Components: These mainly deal with the core part of the task. The complete code of the task (written in high level language or scripting) to be executed should be embedded in this block. One complete workflow can have many independent software components.

Templates: Templates can be defined as ready to serve blueprints which can be provisioned on any hypervisors which are connected to the cloud orchestrator.

Once the job execution flow is fixed then complete steps are written as a software components. Further, these software components are combined and finally made as a executable templates.

As we mentioned in the previous sections, the complete flow of both MPSO and HYBRID (MPSO+MCSO) algorithms are written as software components. Each software component defines the complete flow of the algorithm along with dependencies. Further, these software components are attached to the specific template for the deployment. We created two templates for both the algorithms and then provisioned these cloud enabled templates on to the physical server. These templates are generic and can be provisioned on any type of hypervisor. For the experimentation, the templates are provisioned on the VMware cloud orchestrator and results are discussed in the next section.

7.1.5 Results and Analysis

Results are analyzed for two different cases. First case w.r.t PySim cloud setup and the second case w.r.t. IBM cloud setup. Different parameters are considered for evaluation such as utilization of cloud resources and average response time, and the results of proposed MPSO based scheduling algorithm in comparison with state-of-the-art algorithms are given in Tables 7.2 and 7.3, respectively.

From Tables 7.2 and 7.3, we can observe that proposed MPSO algorithm is tested and validated on both PySim and IBM cloud platforms. It is clearly observed from Tables 7.2 and 7.3 that the proposed MPSO does not deviate from the PySim results. From Table 7.2 it is clear that in both real and simulated cloud platforms the proposed MPSO uses an almost equal number of VMs. But in IBM cloud setup the VMs utilization is not exactly same as that of the PySim setup, but they are in a balanced

Table 7.2: Utilization of VMs

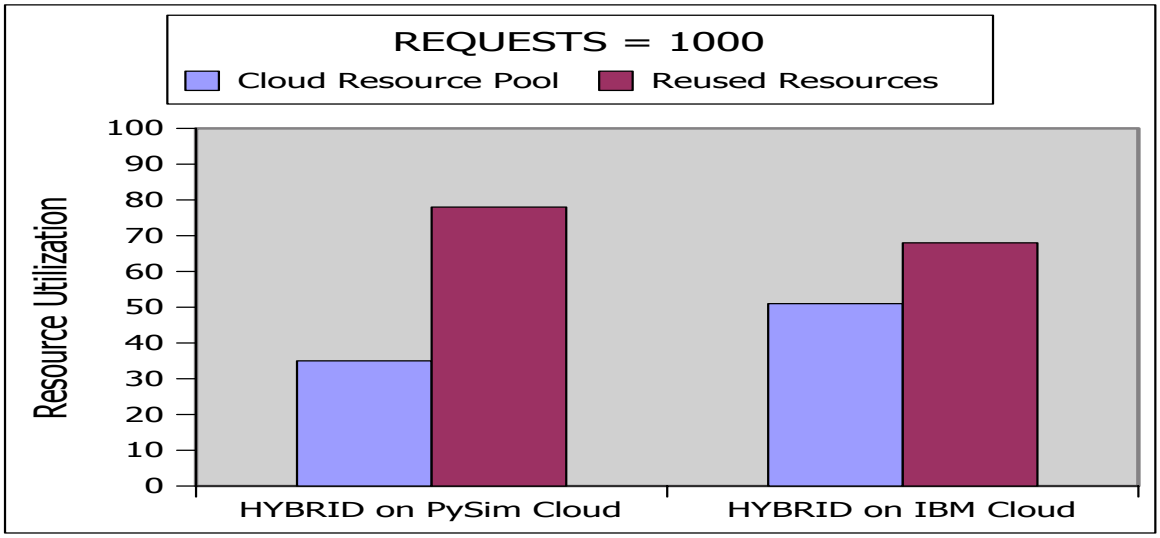
SL. No	PySim Cloud Setup Results					IBM Cloud Setup
	Throttled	Round Robin	ACO	Exact	Proposed MPSO	Proposed MPSO
VM1	1182	254	356	253	254	234
VM2	76	254	289	254	254	264
VM3	8	253	389	254	254	249
VM4	2	253	180	254	253	263
VM5	0	254	54	253	253	258

Table 7.3: Average Response Time Analysis (in ms)

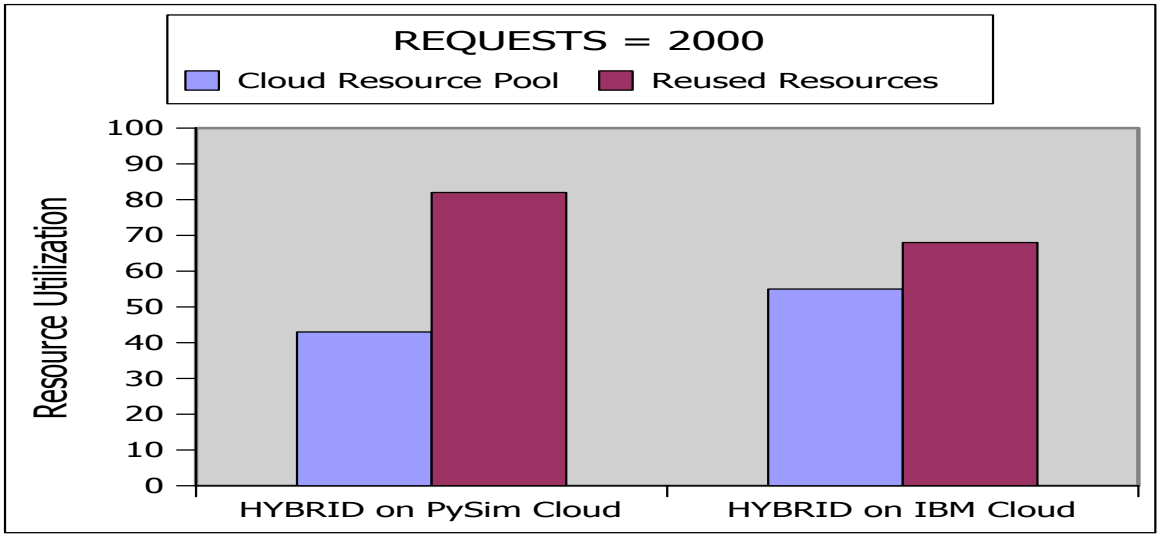
PlatForm	Algorithm	Average Response Time
Pysim Cloud Setup	Throttled	365.52
	Round Robin	364.85
	ACO	362.67
	Exact	365.87
	Proposed MPSO	360.11
IBM Cloud	Proposed MPSO	370.58

way. Further, the proposed algorithm is tested and validated on both cloud platforms (IBM and PySim) for average response time analysis and the results are shown in Table 7.3. The average response time of MPSO on IBM cloud platform is more when compared to PySim platform and this is because of some practical difficulties in real cloud platforms (network connectivity and synchronization). The experiment was conducted on PySim platform by making some assumptions (same inter arrival time and RTT remains same). But in IBM cloud platform, there were some challenges (varied inter arrival time between tasks and network delay from cloud resource pool) by which the results are not exactly same as that of the customized simulation setup. Next, we will analyze the validation of HYBRID (MPSO+MCSO) resources allocation and management algorithm.

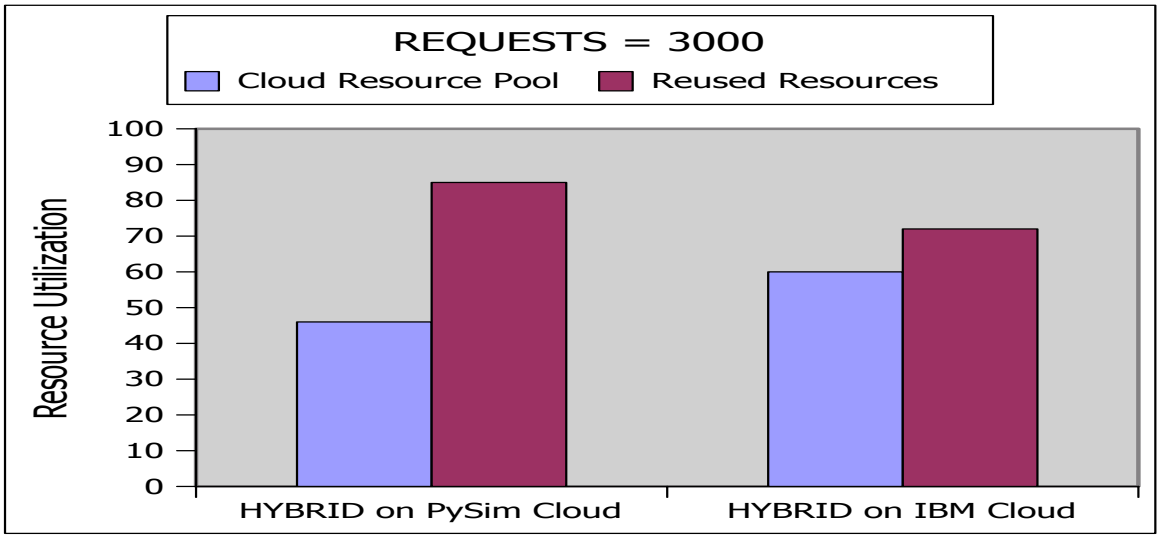
The HYBRID (MPSO+MCSO) resource allocation and management algorithm is experimented on both cloud platforms (IBM and PySim). This HYBRID algorithm is capable of combining the merits of both MPSO and MCSO and gives the better efficiency in utilizing the cloud resources. In customized cloud setup (PySim), our proposed HYBRID algorithm is efficient in utilizing the cloud resources from the VMs buffer rather than the cloud resource pool. Figures 7.3a, 7.3b and 7.3c show the resource utilization for 1000, 2000 and 3000 BoT on both platforms. It is clearly observed from Figure 7.3 that there is a less reuse of the resources from the VMs in the case of IBM cloud platform. On the other hand, the reuse of cloud resources is more in the case of PySim platform. There is some delay in fetching the resources from the resource pool when testing the proposed HYBRID algorithm in real cloud setup. However, our proposed HYBRID resources allocation and management algorithm achieves 76% accuracy when validated with IBM cloud setup.



(a) Resource Utilization with 1000 Tasks



(b) Resource Utilization with 2000 Tasks



(c) Resource Utilization with 3000 Tasks

Figure 7.3: Average Resource Utilization Analysis

7.2 Summary

In this contribution, testing and validation is done using customized PySim and IBM cloud setup. Two algorithms mainly MPSO and HYBRID (MPSO+MCSO) are experimented on IBM cloud setup to validate our results. Both the algorithms are converted into templates using software components and provisioned on the VMware cloud orchestrator. The validation on the VMware hypervior is consistent and the templates can be provisioned on the other suitable hypervisors. Further, the performance analysis on QoS parameters (Reliability and Time), i.e. utilization of cloud resources and average response time is carried out using different workloads. Our proposed algorithms outperform state-of-the-art algorithms and thus achieve overall efficiency of 85% in real IBM cloud setup.

Chapter 8

Conclusion and Future Directions

The efficient scheduling, load balancing and resource utilization (VMs, CPU, RAM, etc.) at the cloud data center is an important and challenging problem in the cloud computing environment. Cloud receives clients' requests at a rapid rate, and scheduling algorithms allocate these tasks to the VMs and thus maintains a balanced load on the PMs. Further, the type of the input (BoT or Workflows) should be handled in an intelligent manner while scheduling at the cloud data center. Since, these algorithms fall under NP-Hard/NP-Complete complex class and hence, there is a scope for further improvement. Therefore, the algorithms should be multi-objective and also should be capable of optimizing QoS parameters (Reliability, Time, Throughput, Cost, etc.) including hot plugging techniques.

Scheduling plays a major role in the cloud data center as it consists of large pool of resources. In our work, we proposed a few scheduling algorithms based on optimizing the specific QoS parameter. Once the scheduling is done, the proper utilization of the cloud resources is a challenging task and this is handled by our proposed resource allocation and management algorithms. Along with resource management, choosing of right VMs instance will affect the optimization of the cost QoS parameter.

Further, many VMs instances (On-Demand, Reserved and Spot) are offered by the cloud service providers and utilization of these VMs instances also play a major role at the cloud data center. The choosing of suitable VMs instance is application

specific and further proper scheduling and prediction (Spot) of these VMs instances will help in reducing the total cost.

Hence, the research work in this thesis is directed towards the design and development of task scheduling (BoT and Workflows), load balancing and resource utilization algorithms in both homogeneous and heterogeneous cloud computing environments. Further, thesis contributes to the design and development of a fault tolerant system with VMs migration to guarantee the task completion without the violation of SLA using On-Demand and Spot instances.

The first set of contributions of this thesis address the scheduling of independent tasks (BoT) in the cloud computing environment which optimizes the QoS parameters (Reliability, Time, Throughput). Here, three scheduling algorithms (Modified Throttled, VM-Assign and DCBT) are proposed for efficient scheduling of tasks and further optimal utilization of cloud resources is taken care. The proposed algorithms outperform other state-of-the-art techniques (Throttled, Round Robin, Active-VM) with better execution time, average response time and thus efficiently utilize the VMs (above 88%) at the cloud data center. Here, only VMs are considered as cloud resources for performance analysis. Algorithms work efficiently on independent tasks and further can be improved to handle dependent tasks which are complex in nature. Apart from VMs, the cloud data center has many resources which are demanded by the incoming tasks and managing these resources (mainly CPU and MEMORY) is a challenging problem.

The second set of contributions of this thesis address the Bio-Inspired scheduling of independent (BoT) tasks along with the efficient utilization of cloud resources (CPU, VMs and Memory) in the heterogeneous cloud environment. The proposed MPSO scheduling algorithm outperforms the state-of-the-art techniques (Throttled, Round Robin, ACO, Exact Algorithm) in terms of VMs utilization and improved average response time. Further, MPSO, MCSO and HYBRID (MPSO+MCSO) algorithms are proposed for efficient utilization of cloud resources using hot plugging

technique. The proposed MPSO resource allocation and management algorithm efficiently utilizes the cloud resources from the VMs buffer rather than taking it from the cloud resource pool. However, MPSO has some limitation in (unmatched resource) and this is addressed by the proposed MCSO algorithm. Further, the proposed HYBRID (MPSO+MCSO) algorithm outperforms the state-of-the-art techniques (MPSO, MCSO, RR, ACO, Exact) in utilizing the cloud resources. The proposed algorithms are analyzed using both sequential and parallel approaches while assigning the resources during the task execution (1000, 2000 and 3000 tasks are considered for performance analysis). Our proposed algorithms achieve 84% in utilizing the resources from the cloud resource pool. Further, the proposed HYBRID (MPSO+MCSO) algorithm is validated with statistical analysis (T-Test) for null hypothesis by comparing with other algorithms. Many applications are hosted on the cloud data center and fetching / execution of these applications is a complex process. The resources must be prioritized and should be reliable at any point of time. Nowadays multimedia applications are high in demand and need to be served with no delay. So combinations of different applications must be considered for robust algorithmic framework to work in the cloud environment. Here, BoT (independent tasks) are considered as an input to the cloud environment and in the next contribution we consider both BoT and Workflows (dependent tasks).

The third set of contributions of this thesis address the scheduling of BoT and Workflows using Bio-Inspired GWO techniques to optimize both the execution time and execution cost (without the violation of the SLA). Here, we used Amazon VMs instances during execution and results demonstrate that the proposed algorithms outperform the state-of-the-art techniques (ICPCP, SCS, PSO, CSO, Exact) with better execution time (for BoT) and reduced cost (for Workflows). The proposed GWO algorithm efficiently utilizes the VMs instances during execution of tasks from the Workflows and hence cost of executing the Scientific Workflows (Montage and Cybershake) is reduced considerably (by 11%). Here, we simulated the environment

by considering the reserved VMs instances from Amazon. However, different VMs instances from different cloud vendors must be considered for more accurate results. Sudden change in the resource requirements affects the cloud data center performance and hence always there should be a reserved resource pool. Nowadays many real time applications are hosted on the cloud environment so apart from BoT and Workflows as a input, the dynamic workloads must be considered. Next, we consider different types of instances (On-Demand and Spot) and further, we focus on predicting the Spot instances to optimize the cost QoS parameter.

The fourth set of contributions of this thesis address the efficient utilization of cloud resources by VMs migration and also optimize the total execution cost by using more Spot instances rather than On-Demand instances. Here, we used ANN based back propagation algorithm for predicting the future Spot instance values and further these predicted Spot values are compared with actual values. During VMs migration, the role of hypervisor plays an important role and migration between different hypervisors is a challenging problem for further improvement. Apart from On-Demand and Spot instances, the other instances can be explored which suits for dynamic workloads for real time cloud applications. Further, we use Spearman's Rho test for statistical validation and our predicted values correlate with actual values by 94.28%. Further, we compared the cost analysis of On-Demand and Spot instances while executing different ranges of BoT (Small, Medium and Large) and the results demonstrate that the Spot instances can reduce the cost by 38.23%.

The proposed algorithms in this thesis are experimented on both CloudAnalyst (CloudSim based) and customized PySim (Python based) simulators. The experimental results demonstrate that the proposed algorithms outperform the state-of-the-art approaches in terms of optimizing QoS parameters (Reliability, Time, Throughput, Cost, etc.). *Hence, the fifth contribution of this thesis* is to test and validate the MPSO (scheduling) and HYBRID (MPSO+MCSO) (resource allocation and management) algorithms on the real cloud setup (IBM cloud platform). The experiments

are conducted on cloud orchestrator with the help of software components and templates. The experimentation is carried out in the VMware orchestrator with ESXi hypervisor. There can be generic templates which can be provisioned on any end point hypervisors. Further, the same experiment can be conducted on the different cloud orchestrators for performance analysis and a few more software components can be added to handle the different conditions during the deployment. The validation on the VMware hypervior is consistent and the templates cab be provisioned on the other suitable hypervisors. Further, the experimental results demonstrate that the proposed MPSO and HYBRID (MPSO+MCSO) algorithms achieve overall efficiency of 85% in terms of scheduling and resource allocation management when compared to the results of simulation setup.

In summary, we proposed algorithms for scheduling, load balancing and resource allocation and management in the cloud computing environment using BoT (independent tasks) and Workflow (dependent tasks) as workloads. Experimental results demonstrate that our proposed algorithms outperform the state-of-the-art techniques. However, further improvement of the proposed algorithms can be considered with the following future research directions.

- To validate our proposed Bio-Inspired scheduling and load balancing algorithms in different real cloud setups and workflow engines.

- To design and develop efficient resource allocation and management policy for the federated cloud environments. Further, a framework can be designed for handling real time cloud application in which the resource demands change dynamically.

- Procuring of suitable VMs instances from the different cloud vendors can be automated using the artificial intelligence techniques. A generic framework can be designed using the machine learning approach to choose the suitable algorithm for a specific workload.

- Building cognitive based smart frameworks/templates for executing different platform applications in the hybrid cloud environments and also in the different cloud orchestrators.

- Using cognitive approaches (mainly using the deep neural networks), the data center can be trained to make its own decision in choosing, reserving the appropriate cloud resources at a given time. For the real time cloud applications, the availability of desired resources are mandatory and should be served with no delay.

References

Aceto, G., Botta, A., De Donato, W., and Pescapè, A. (2013). Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115.

Achar, R., Thilagam, P. S., Soans, N., Vikyath, P., Rao, S., and Vijeth, A. (2013). Load balancing in cloud based on live migration of virtual machines. In *India Conference (INDICON), 2013 Annual IEEE*, pages 1–5. IEEE.

Addis, B., Ardagna, D., Panicucci, B., Squillante, M. S., and Zhang, L. (2013). A hierarchical approach for the resource management of very large cloud platforms. *IEEE Transactions on Dependable and Secure Computing*, 10(5):253–272.

Ahmad, B., Yazidi, A., Haugerud, H., and Farokhi, S. (2017). Orchestrating resource allocation for interactive vs. batch services using a hybrid controller. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 1195–1203. IEEE.

Akbari, M. and Rashidi, H. (2016). A multi-objectives scheduling algorithm based on cuckoo optimization for task allocation problem at compile time in heterogeneous systems. *Expert Systems with Applications*, 60:234–248.

Al Nuaimi, K., Mohamed, N., Al Nuaimi, M., and Al-Jaroodi, J. (2012). A survey of load balancing in cloud computing: Challenges and algorithms. In *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on*, pages 137–142. IEEE.

Arabnejad, V., Bubendorfer, K., and Ng, B. (2017). Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources. *Future Generation Computer Systems*.

Baker, T., Al-Dawsari, B., Tawfik, H., Reid, D., and Ngoko, Y. (2015). Greedi: An energy efficient routing algorithm for big data on cloud. *Ad Hoc Networks*, 35:83–96.

Baker, T., Ngoko, Y., Tolosana-Calasanz, R., Rana, O. F., and Randles, M. (2013). Energy efficient cloud computing environment via autonomic meta-director framework. In *Developments in eSystems Engineering (DeSE), 2013 Sixth International Conference on*, pages 198–203. IEEE.

Bera, S., Misra, S., and Rodrigues, J. J. (2015). Cloud computing applications for smart grid: A survey. *IEEE Transactions on Parallel and Distributed Systems*, 26(5):1477–1494.

Bilgaiyan, S., Sagnika, S., and Das, M. (2014). Workflow scheduling in cloud computing environment using cat swarm optimization. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 680–685. IEEE.

Binitha, S., Sathya, S. S., et al. (2012). A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering*, 2(2):137–151.

Bitam, S. and Mellouk, A. (2012). Its-cloud: Cloud computing for intelligent transportation system. In *Global Communications Conference (GLOBECOM), 2012 IEEE*, pages 2054–2059. IEEE.

Bittencourt, L. F., Madeira, E. R., and Da Fonseca, N. L. (2012). Scheduling in hybrid clouds. *IEEE Communications Magazine*, 50(9).

Bittencourt, L. F. and Madeira, E. R. M. (2011). Hcoc: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, 2(3):207–227.

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press.

Cerroni, W. (2014). Multiple virtual machine live migration in federated cloud systems. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 25–30. IEEE.

Chen, X., Chen, S., Tseng, F.-H., Chou, L.-D., and Chao, H.-C. (2013). Minimizing virtual machine migration probability for cloud environments. In *High Performance*

Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, pages 1430–1436. IEEE.

Cheng, D., Rao, J., Guo, Y., Jiang, C., and Zhou, X. (2017). Improving performance of heterogeneous mapreduce clusters with adaptive task tuning. *IEEE Transactions on Parallel and Distributed Systems*, 28(3):774–786.

Chimakurthi, L. et al. (2011). Power efficient resource allocation for clouds using ant colony framework. *arXiv preprint arXiv:1102.2608*.

Chunlin, L., Xin, Y., Yang, Z., and Youlong, L. (2017). Multiple context based service scheduling for balancing cost and benefits of mobile users and cloud datacenter supplier in mobile cloud. *Computer Networks*, 122:138–152.

Cloud, A. E. C. (2011). Amazon web services. *Retrieved November*, 9:2011.

Daochao, H., Chunge, Z., Hong, Z., and Xinran, L. (2014). Resource intensity aware job scheduling in a distributed cloud. *China Communications*, 11(14):175–184.

de Souza, F. R., Miers, C. C., Fiorese, A., and Koslovski, G. P. (2017). Qos-aware virtual infrastructures allocation on sdn-based clouds. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 120–129. IEEE Press.

Di, S. and Wang, C.-L. (2013). Error-tolerant resource allocation and payment minimization for cloud system. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1097–1106.

Díaz, J. L., Entrialgo, J., García, M., García, J., and García, D. F. (2017). Optimal allocation of virtual machines in multi-cloud environments with reserved and on-demand pricing. *Future Generation Computer Systems*, 71:129–144.

Domanal, S. G. and Reddy, G. R. M. (2015). Load balancing in cloud environment using a novel hybrid scheduling algorithm. In *Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on*, pages 37–42. IEEE.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE.

Gonçalves, G., Santos, M., Charamba, G., Endo, P., Sadok, D., Kelner, J., Melander, B., and Mångs, J.-E. (2012). D-cras: Distributed cloud resource allocation system. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 659–662. IEEE.

Goudarzi, H., Ghasemazar, M., and Pedram, M. (2012). Sla-based optimization of power and migration cost in cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, pages 172–179. IEEE.

Gunarathne, T., Wu, T.-L., Qiu, J., and Fox, G. (2010). Mapreduce in the clouds for science. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 565–572. IEEE.

Gutierrez-Garcia, J. O. and Ramirez-Nafarrate, A. (2013). Policy-based agents for virtual machine migration in cloud data centers. In *Services Computing (SCC), 2013 IEEE International Conference on*, pages 603–610. IEEE.

Hans, R., Steffen, D., Lampe, U., Stingl, D., and Steinmetz, R. (2016). Short run: Heuristic approaches for cloud resource selection. In *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on*, pages 569–576. IEEE.

Hong, C.-H., Spence, I., and Nikolopoulos, D. S. (2017). Fairgv: fair and fast gpu virtualization. *IEEE Transactions on Parallel and Distributed Systems*, 28(12):3472–3485.

HoseinyFarahabady, M. R., Lee, Y. C., Zomaya, A. Y., Tari, Z., and Song, A. (2016). A model predictive controller for contention-aware resource allocation in virtualized data centers. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*, pages 277–282. IEEE.

Hsiao, H.-C., Chung, H.-Y., Shen, H., and Chao, Y.-C. (2013). Load rebalancing for distributed file systems in clouds. *IEEE transactions on parallel and distributed systems*, 24(5):951–962.

Hussain, H., Malik, S. U. R., Hameed, A., Khan, S. U., Bickler, G., Min-Allah, N., Qureshi, M. B., Zhang, L., Yongji, W., Ghani, N., et al. (2013). A survey on

resource allocation in high performance distributed computing systems. *Parallel Computing*, 39(11):709–736.

Hwang, K., Dongarra, J., and Fox, G. C. (2013). *Distributed and cloud computing: from parallel processing to the internet of things*. Morgan Kaufmann.

Indukuri, R. K. R., Varma, P. S., and Moses, G. J. (2012). Performance measure of multi stage scheduling algorithm in cloud computing. In *Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on*, pages 8–11. IEEE.

Jamshidi, P., Ahmad, A., and Pahl, C. (2013). Cloud migration research: a systematic review. *IEEE Transactions on Cloud Computing*, 1(2):142–157.

Javadi, B., Thulasiramy, R. K., and Buyya, R. (2011). Statistical modeling of spot instance prices in public cloud environments. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 219–228. IEEE.

Jeyarani, R., Nagaveni, N., and Ram, R. V. (2012). Design and implementation of adaptive power-aware virtual machine provisioner (apa-vmp) using swarm intelligence. *Future Generation Computer Systems*, 28(5):811–821.

Jung, S.-M., Kim, N.-U., and Chung, T.-M. (2013). Applying scheduling algorithms with qos in the cloud computing. In *Information Science and Applications (ICISA), 2013 International Conference on*, pages 1–2. IEEE.

Karunakaran, S. and Sundarraj, R. P. (2015). Bidding strategies for spot instances in cloud computing markets. *IEEE Internet Computing*, 19(3):32–40.

Katyal, M. and Mishra, A. (2014). A comparative study of load balancing algorithms in cloud computing environment. *arXiv preprint arXiv:1403.6918*.

Kliazovich, D., Arzo, S. T., Granelli, F., Bouvry, P., and Khan, S. U. (2013). e-stab: Energy-efficient scheduling for cloud computing applications with traffic load balancing. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 7–13. IEEE.

Kryftis, Y., Mastorakis, G., Mavromoustakis, C. X., Batalla, J. M., Rodrigues, J. J., and Dobre, C. (2017). Resource usage prediction models for optimal multimedia content provision. *IEEE Systems Journal*, 11(4):2852–2863.

Kundu, A. and Ji, C. (2012). Swarm intelligence in cloud environment. *Advances in Swarm Intelligence*, pages 37–44.

LD, D. B. and Krishna, P. V. (2013). Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, 13(5):2292–2303.

Leivadeas, A., Papagianni, C., and Papavassiliou, S. (2013). Efficient resource mapping framework over networked clouds via iterated local search-based request partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1077–1086.

Liu, L., Zhang, M., Lin, Y., and Qin, L. (2014). A survey on workflow management and scheduling in cloud computing. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 837–846. IEEE.

Liu, Z. and Wang, X. (2012). A pso-based algorithm for load balancing in virtual machines of cloud computing environment. *Advances in Swarm Intelligence*, pages 142–147.

MacVittie, L. (2012). Cloud balancing: The evolution of global server load balancing. *white paper F5 Networks*.

Maechling, P., Deelman, E., Zhao, L., Graves, R., Mehta, G., Gupta, N., Mehringer, J., Kesselman, C., Callaghan, S., Okaya, D., et al. (2007). Scec cybershake workflowsautomating probabilistic seismic hazard analysis calculations. *Workflows for e-Science*, pages 143–163.

Mahalle, H. S., Kaveri, P. R., and Chavan, V. (2013). Load balancing on cloud data centres. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(1).

Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management science*, 44(5):714–729.

Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61.

Mohamed, N. and Al-Jaroodi, J. (2011). Delay-tolerant dynamic load balancing. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 237–245. IEEE.

Mondal, B., Dasgupta, K., and Dutta, P. (2012). Load balancing in cloud computing using stochastic hill climbing-a soft computing approach. *Procedia Technology*, 4:783–789.

Neumann, F. and Witt, C. (2013). Bioinspired computation in combinatorial optimization: algorithms and their computational complexity. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, pages 567–590. ACM.

Nishant, K., Sharma, P., Krishna, V., Gupta, C., Singh, K. P., Rastogi, R., et al. (2012). Load balancing of nodes in cloud using ant colony optimization. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 3–8. IEEE.

On-Demand, A. (url). ec2price.com.

Pace, F., Venzano, D., Carra, D., and Michiardi, P. (2017). Flexible scheduling of distributed analytic applications. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 100–109. IEEE Press.

Pahlevan, A., Qu, X., Zapater, M., and Atienza, D. (2017). Integrating heuristic and machine-learning methods for efficient virtual machine allocation in data centers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Pandey, S., Wu, L., Guru, S. M., and Buyya, R. (2010). 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*, pages 400–407. IEEE.

Papagianni, C., Leivadeas, A., Papavassiliou, S., Maglaris, V., Cervello-Pastor, C., and Monje, A. (2013). On the optimal allocation of virtual resources in cloud computing networks. *IEEE Transactions on Computers*, 62(6):1060–1071.

Pillai, P. S. and Rao, S. (2016). Resource allocation in cloud computing using the uncertainty principle of game theory. *IEEE Systems Journal*, 10(2):637–648.

Poola, D., Ramamohanarao, K., and Buyya, R. (2014). Fault-tolerant workflow scheduling using spot instances on clouds. *Procedia Computer Science*, 29:523–533.

Poola, D., Ramamohanarao, K., and Buyya, R. (2016). Enhancing reliability of workflow execution using task replication and spot instances. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(4):30.

Priya, V. and Babu, C. N. K. (2017). Moving average fuzzy resource scheduling for virtualized cloud data services. *Computer Standards & Interfaces*, 50:251–257.

Radojević, B. and Žagar, M. (2011). Analysis of issues with load balancing algorithms in hosted (cloud) environments. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 416–420. IEEE.

Rasti-Barzoki, M. and Hejazi, S. R. (2015). Pseudo-polynomial dynamic programming for an integrated due date assignment, resource allocation, production, and distribution scheduling model in supply chain scheduling. *Applied Mathematical Modelling*, 39(12):3280–3289.

Ribas, M., Furtado, C., de Souza, J. N., Barroso, G. C., Moura, A., Lima, A. S., and Sousa, F. R. (2015). A petri net-based decision-making framework for assessing cloud services adoption: The use of spot instances for cost reduction. *Journal of Network and Computer Applications*, 57:102–118.

Rodriguez, M. A. and Buyya, R. (2014). Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 2(2):222–235.

Rubio-Montero, A., Huedo, E., and Mayo-García, R. (2017). Scheduling multiple virtual environments in cloud federations for distributed calculations. *Future Generation Computer Systems*, 74:90–103.

Singh, D., Singh, J., and Chhabra, A. (2012). High availability of clouds: Failover strategies for cloud computing using integrated checkpointing algorithms. In *Communication Systems and Network Technologies (CSNT), 2012 International Conference on*, pages 698–703. IEEE.

Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. (2009). Virtual infrastructure management in private and hybrid clouds. *IEEE Internet computing*, 13(5).

Spot, A. (url). ec2price.com.

Suresh, A. and Vijayakarthick, P. (2011). Improving scheduling of backfill algorithms using balanced spiral method for cloud metascheduler. In *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, pages 624–627. IEEE.

Taneja, M. and Davy, A. (2017). Resource aware placement of iot application modules in fog-cloud computing paradigm. In *Integrated Network and Service Management (IM), 2017 IFIP/IEEE Symposium on*, pages 1222–1228. IEEE.

Tang, S., Yuan, J., Wang, C., and Li, X.-Y. (2014). A framework for amazon ec2 bidding strategy under sla constraints. *IEEE Transactions on Parallel and Distributed Systems*, 25(1):2–11.

Tayyab, M., Zhou, J., Zeng, X., Ahmed, I., and Adnan, R. (2016). Application of statistical nonparametric tests in dongting lake, china: 1961–2012. In *Knowledge Engineering and Applications (ICKEA), IEEE International Conference on*, pages 197–201. IEEE.

Thiam, C., Da Costa, G., and Pierson, J.-M. (2013). Cooperative scheduling anti-load balancing algorithm for cloud: Csaac. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 433–438. IEEE.

Thinakaran, P., Gunasekaran, J. R., Sharma, B., Kandemir, M. T., and Das, C. R. (2017). Phoenix: a constraint-aware scheduler for heterogeneous datacenters. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 977–987. IEEE.

Tsai, C.-W., Huang, W.-C., Chiang, M.-H., Chiang, M.-C., and Yang, C.-S. (2014). A hyper-heuristic scheduling algorithm for cloud. *IEEE Transactions on Cloud Computing*, 2(2):236–250.

Users, I. (url). http://www.internetworldstats.com/stats.htm.

Wang, S.-C., Yan, K.-Q., Liao, W.-P., and Wang, S.-S. (2010). Towards a load balancing in a three-level cloud computing network. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 1, pages 108–113. IEEE.

Wang, W., Wang, Q., and Sohraby, K. (2017). Multimedia sensing as a service (msaas): Exploring resource saving potentials of at cloud-edge iot and fogs. *IEEE Internet of Things Journal*, 4(2):487–495.

Wei, L., Cai, J., Foh, C. H., and He, B. (2017). Qos-aware resource allocation for video transcoding in clouds. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(1):49–61.

Wei, X., Fan, J., Lu, Z., Ding, K., Li, R., and Zhang, G. (2013). Bio-inspired application scheduling algorithm for mobile cloud computing. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on*, pages 690–695. IEEE.

Weinman, J. (2015). Cloud pricing and markets. *IEEE Cloud Computing*, 2(1):10–13.

Wickremasinghe, B., Calheiros, R. N., and Buyya, R. (2010). Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 446–452. IEEE.

Wong, M. D. and Nandi, A. K. (2004). Automatic digital modulation recognition using artificial neural network and genetic algorithm. *Signal Processing*, 84(2):351–365.

Wu, T.-Y., Lee, W.-T., Lin, Y.-S., Lin, Y.-S., Chan, H.-L., and Huang, J.-S. (2012). Dynamic load balancing mechanism based on cloud storage. In *Computing, Communications and Applications Conference (ComComAp), 2012*, pages 102–106. IEEE.

Wuhib, F., Stadler, R., and Spreitzer, M. (2012). A gossip protocol for dynamic resource management in large cloud environments. *IEEE transactions on network and service management*, 9(2):213–225.

Xu, H. and Li, B. (2013). Anchor: A versatile and efficient framework for resource management in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1066–1076.

Xu, X., Dou, W., Zhang, X., and Chen, J. (2016a). Enreal: An energy-aware resource allocation method for scientific workflow executions in cloud environment. *IEEE Transactions on Cloud Computing*, 4(2):166–179.

Xu, Z., Stewart, C., Deng, N., and Wang, X. (2016b). Blending on-demand and spot instances to lower costs for in-memory storage. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE.

Ye, K., Jiang, X., Huang, D., Chen, J., and Wang, B. (2011). Live migration of multiple virtual machines with resource reservation in cloud computing environments. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 267–274. IEEE.

Yi, S., Andrzejak, A., and Kondo, D. (2012). Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Transactions on Services Computing*, 5(4):512–524.

Yi, S., Kondo, D., and Andrzejak, A. (2010). Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 236–243. IEEE.

Yu, J., Buyya, R., and Tham, C. K. (2005). Cost-based scheduling of scientific workflow applications on utility grids. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8–pp. Ieee.

Zhang, Z. and Zhang, X. (2010). A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation. In *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, volume 2, pages 240–243. IEEE.

Zhao, J., Yang, K., Wei, X., Ding, Y., Hu, L., and 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.

Zheng, J., Ng, T. E., Sripanidkulchai, K., and Liu, Z. (2013). Pacer: A progress management system for live virtual machine migration in cloud computing. *IEEE transactions on network and service management*, 10(4):369–382.

Zheng, X.-l. and Wang, L. (2016). A pareto based fruit fly optimization algorithm for task scheduling and resource allocation in cloud computing environment. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 3393–3400. IEEE.

Zhou, A. C., He, B., and Liu, C. (2016). Monetary cost optimizations for hosting workflow-as-a-service in iaas clouds. *IEEE transactions on cloud computing*, 4(1):34–48.

Zhou, W., Yang, S., Fang, J., Niu, X., and Song, H. (2010). Vmctune: A load balancing scheme for virtual machine cluster using dynamic resource allocation. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 81–86. IEEE.

Zuo, X., Zhang, G., and Tan, W. (2014). Self-adaptive learning pso-based deadline constrained task scheduling for hybrid iaas cloud. *IEEE Transactions on Automation Science and Engineering*, 11(2):564–573.

Publications

International Journals

1. Shridhar G.Domanal, G. Ram Mohan Reddy and Rajkumar Buyya, A Hybrid Bio-Inspired Algorithm for Scheduling and Resource Management in Cloud Environment, *IEEE Transactions on Services Computing,* March 2017, DOI: 10.1109/TSC.2017.2679738.

2. Shridhar G. Domanal, G. Ram Mohan Reddy, An Efficient Cost Optimized Scheduling for Spot Instances in Heterogeneous Cloud Environment, *Elsevier Future generations Computer Systems.* (Accepted with Minor Revision)

3. Shridhar G. Domanal, G. Ram Mohan Reddy, A Novel Bio-Inspired Scheduling Algorithm for Scientific workflows and Bag of Tasks in Cloud Environment, *Springer Journal of Grid Computing.* (Under Review)

Conference Publications

1. Shridhar G.Domanal and G. Ram Mohan Reddy, Load Balancing in Cloud Computing using Modified Throttled Algorithm, *in Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference on.* IEEE, 2013, pp. 1-5, October 16-18, Bangalore, India. [**Citation: 72**]

2. Shridhar G.Domanal and G. Ram Mohan Reddy, Optimal Load Balancing in Cloud Computing By Efficient Utilization of Virtual Machines, *in 2014 Sixth International Conference on Communication Systems and Networks (COMSNETS).* IEEE, 2014, pp. 1-4, January 5-8, Bangalore, India. [**Citation: 60**]

3. Shridhar G.Domanal and G. Ram Mohan Reddy, Load Balancing in Cloud Environment using a Novel Hybrid Scheduling Algorithm, *in Cloud Computing in Emerging Markets (CCEM), 2015 IEEE International Conference on,* IEEE, 2015, pp. 37-42, Bengalore, India. (**Acceptance ratio $<$ 10%**)

4. Ashwin T S, Shridhar G.Domanal and G. Ram Mohan Reddy, A Novel Bio-Inspired Load Balancing of Virtual Machines in Cloud Environment, *in Cloud Computing in Emerging Markets (CCEM), 2014 IEEE International Conference on,* IEEE, 2014, pp. 1-4, Bangalore, India. [**Citation: 07**]

5. Shridhar G.Domanal and G. Ram Mohan Reddy, An Efficient Cost Optimized Scheduling Algorithm for Scientific Workflows and Bag of Tasks on Clouds, *in the 13th IEEE International Conference on Services Computing,* 2016, San Francisco, California, USA.[**Accepted**]

6. Shridhar G.Domanal and G. Ram Mohan Reddy, DCBT: A Novel Hybrid Scheduling Algorithm for Load Balancing in Cloud Environment, *in the 5th International Symposium on Frontiers in Ambient and Mobile Systems (FAMS)* Procedia, Elsevier Science, 2015, London, UK.[**Accepted**]

Brief Bio-Data

Shridhar G. Domanal
Research Scholar
Department of Information Technology
National Institute of Technology Karnataka Surathkal
P.O.Srinivasanagar
Mangalore - 575 025
Phone: 09739314932
Email: shridhar.domanal@gmail.com

Permanent address

Shridhar G. Domanal
Behind BLDEA's Engg College Boys Hostel
SHIVAKRUPA
Sajjan Colony
Vijayapur - 586 103
Karnataka.

Qualification

M.Tech in Computer Science & Engg, Gogte Institute of Technology (VTU), Belagavi, 2011.

B.E. in Computer Science & Engg, BLDEA's College of Engg & Tech (VTU), Vijayapur, 2009.