CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

A Resource-Aware Dynamic Load-balancing Technique for Deadline Constrained Cloud Tasks

by

Said Nabi

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

in the

Faculty of Computing
Department of Computer Science

2022

A Resource-Aware Dynamic Load-balancing Technique for Deadline Constrained Cloud Tasks

By

Said Nabi

(DCS161005)

Dr. Jerry CHun-Wei Lin, Professor

Western Norway University of Applied Sciences, Norway

(Foreign Evaluator 1)

Dr. Radu Prodan, Professor

Alpen-Adria University Klagenfurt, Klagenfurt, Austria (Foreign Evaluator 2)

Dr. Mohammad Masroor Ahmed

(Thesis Supervisor)

Dr. Nayyer Masood

(Head, Department of Computer Science)

Dr. Muhammad Abdul Qadir

(Dean, Faculty of Computing)

DEPARTMENT OF COMPUTER SCIENCE

CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

ISLAMABAD

2022

This thesis is dedicated to all my family members, especially to my mother

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD

Expressway, Kahuta Road, Zone-V, Islamabad
Phone:+92-51-111-555-666 Fax: +92-51-4486705
Email: info@cust.edu.pk Website: https://www.cust.edu.pk

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in the thesis, entitled "**A Resource-Aware Dynamic Load-balancing Technique for Deadline Constrained Cloud Tasks**" was conducted under the supervision of **Dr. Mohammad Masroor Ahmed**. No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the **Department of Computer Science, Capital University of Science and Technology** in partial fulfillment of the requirements for the degree of Doctor in Philosophy in the field of **Computer Science**. The open defence of the thesis was conducted on **March 10, 2022**.

Student Name : Said Nabi (DCS-161005) _____

The Examining Committee unanimously agrees to award PhD degree in the mentioned field.

Examination Committee :

(a) External Examiner 1: Dr. Zahid Halim
Associate Professor
GIKI, Topi, Swabi _____

(b) External Examiner 2: Dr. Asad Waqar Malik
Associate Professor
SEECS, NUST, Islamabad _____

(c) Internal Examiner : Dr. Muhammad Siraj Rathore
Assistant Professor
CUST, Islamabad _____

Supervisor Name : Dr. Mohammad Masroor Ahmed
Associate Professor
CUST, Islamabad _____

Name of HoD : Dr. Nayyer Masood
Professor
CUST, Islamabad _____

Name of Dean : Dr. Muhammad Abdul Qadir
Professor
CUST, Islamabad _____

AUTHOR'S DECLARATION

I, **Said Nabi (Registration No. DCS-161005)**, hereby state that my PhD thesis entitled, '**A Resource-Aware Dynamic Load-balancing Technique for Deadline Constrained Cloud Tasks**' is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/ world.

At any time, if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my PhD Degree.

(Said Nabi)

Dated: March, 2022

Registration No : DCS-161005

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled "**A Resource-Aware Dynamic Load-balancing Technique for Deadline Constrained Cloud Tasks**" is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/ cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of PhD Degree, the University reserves the right to withdraw/ revoke my PhD degree and that HEC and the University have the right to publish my name on the HEC/ University Website on which names of students are placed who submitted plagiarized thesis.

(Said Nabi)

Dated: March, 2022

Registration No : DCS161005

List of Publications

It is certified that following publication(s) have been made out of the research work that has been carried out for this thesis:-

1. **S. Nabi**, M. Ahmed, "OG-RADL: overall performance-based resource-aware dynamic load-balancer for deadline constrained Cloud tasks," *Supercomputing* https://doi.org/10.1007/s11227-020-03544-z, vol. 77, no. 7, pp. 7476–7508, 2021.

2. **S. Nabi**, M. Ahmed, "PSO-RDAL: Particle Swarm Optimization based Resource and Deadline Aware dynamic Load-balancer for Deadline Constrained Cloud Tasks," *Supercomputing*, https://doi.org/10.1007/s11227-021-04062-2, vol. 78 no. 4, pp. 4624-4654, 2021.

3. **S. Nabi**, M. Aleem, M. Ahmed, A. Islam, A. Iqbal "RADL: A Resource and Deadline-aware Dynamic Load-balancer for Cloud Tasks," *Supercomputing* https://doi.org/10.1007/s11227-022-04426-2, 2022.

4. **S. Nabi**, M. Ahmad, M. Ibrahim, and H. Hamam, "Adpso: Adaptive pso-based task scheduling approach for cloud computing," *Sensors*, vol. 22, no. 3, pp. 920, 2022.

Said Nabi
(DCS161005)

Acknowledgement

I would like to thank all those who extend their contributions in the complete journey of my Ph.D. thesis both directly and indirectly. First of all, I express my sincere gratitude to my thesis supervisor Dr. Masroor Ahmed for his advice, encouragement, and support in completing my doctorate thesis. Secondly, I would like to thank my Ex-supervisor Dr. Muhammad Aleem who extends his efforts in providing me the baseline knowledge for my Ph.D. research.

I am also highly grateful to Dr. Adul Qadir, Dean of Faculty of computing, CUST, for his time, interest, and insightful suggestions during the graduate research seminars. I also want to acknowledge the role of Dr. Arshad Islam, Dr. Altaf Hussain, and all members of the Parallel Computing and Networks (PCN) research lab, who always show their willingness for support and discussion that broaden my horizon.

Finally, I am highly grateful to my mother for her silent support for my passion for pursuing the doctorate while other family members opposed it. I am thankful to my wife and children who sacrificed and suffered during my Ph.D. journey.

Abstract

Cloud computing has emerged as an attractive platform to facilitate the computing needs of users over the Internet. The Cloud service providers acquire the computing resources (software and hardware) and assign them to the users on a pay-per-use mode. To effectively utilize Cloud resources and achieve high user satisfaction, Cloud service providers demand efficient task scheduling algorithms. An efficient task scheduling algorithm should be fair and adaptive to improve resource utilization, meet task deadlines, minimize the makespan, reduce the task response time, and task rejection ratio. There are a number of task scheduling and load balancing algorithms, however, most of these scheduling algorithms fail to achieve efficient resource utilization and load-balancing. The main reason is that these algorithms are not resource and deadline-aware. Moreover, these algorithms execute tasks in batch mode, do not properly monitor/update the Virtual Machines (VMs) load and tasks execution status at run-time. To achieve high resource utilization, load balancing, minimized makespan, reduced task response time, and lower task rejection ratio, there is a need to have such dynamic scheduling algorithms which can monitor and update the VMs load along with tasks execution status at run-time. Furthermore, these algorithms should have a high capability to meet task deadlines and reduce the task rejection ratio. To overcome these issues, a resource aware dynamic load balancing technique for deadline constrained task has been proposed. The contribution of the proposed scheduling scheme has been divided into three parts.

The first part of this thesis presents a Resource Aware Dynamic Load-balancer (RADL) for deadline constrained Cloud tasks. The RADL approach has the ability to evenly distribute the incoming workload of independent and compute-intensive tasks at run-time. In addition, RADL approach has the capability to accommodate the newly arrived tasks (with shorter deadlines) in efficient manner. The proposed approach consists of two schedulers namely RADL-Scheduler and ShifterScheduler (S-Scheduler). RADL-Scheduler allocates incoming tasks to a set of VMs based on minimum completion time. It also monitors/updates the task and VM status. S-Scheduler, finds a suitable position in the task queue of a VM for the incoming

tasks with shorter deadlines. Experimental results show that the proposed approach has attained up to 67.74%, 303.57%, 259.2%, and 146.13% improvement in terms of average resource utilization, meeting tasks deadlines, lower makespan, and task response time respectively as compared to the state-of-the-art tasks scheduling heuristics.

In the literature, a number of task scheduling and load balancing schemes have been proposed. However, the majority of these scheduling heuristics focus either on a single evaluation parameter (i.e., makespan or resource-utilization, etc.) or multiple evaluation parameters individually as a scheduling objective. Improving one parameter may not guarantee an increase in the overall performance of the Cloud. There is a need to have such algorithms that focus on improving the overall performance of the Cloud by taking into account multiple evaluation parameters.

The second part of this thesis attempts to present, an *Overall Performance-based Resource Aware Dynamic Load-balancer* (OG-RADL) for deadline constrained Cloud tasks. OG-RADL has the ability to distribute the workload of independent and compute-intensive tasks according to the resource computation capability at run time. Moreover, a novel normalization technique is proposed that overcome the limitations of existing normalization techniques. The OG-RADL enhance load-balancing, support deadline constrained tasks, and improve the overall performance gain of the Cloud. The experimental result shows that the proposed approach OG-RADL outperforms as compared to existing task scheduling algorithms named DLBA, DC-DLBA, Dy-MaxMin, RALBA, PSSELB, and MODE in terms of the overall performance of the Cloud.

The objective of the Cloud users is to lease optimal resources that meet their demand with minimum cost and time. A number of heuristics and meta-heuristics based approaches have been proposed in the literature. The majority of the existing state-of-the-art task scheduling heuristics optimize a single parameter or multiple non-conflicting parameters like makespan, throughput, response time, etc. However, in the real Cloud scenario, most of the time Cloud users demand for multi-objective based conflicting Quality of Service (QoS) requirements i.e., task execution time and cost. Therefore, there is a need for schedulers that can provide a balanced solution for conflicting parameters. For this purpose, meta-

heuristics based algorithms are considered more efficient to provide an optimized solution for conflicting objectives. In this part of the thesis, a modified PSO based Resource and Deadline Aware dynamic Load-balanced (PSO-RADL) algorithm is proposed. PSO-RADL can provide an optimized solution for the workload of independent and compute-intensive tasks with reasonable time and cost. Experimental results reveal that the PSO-RADL has gained up to 66%, 162%, 56%, 89%, 98%, and 97% enhancement in terms of makespan, average resource utilization, task response time, meeting task deadline, penalty cost, and total execution cost respectively as compared to existing state-of-the-art tasks scheduling heuristics.

Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
ARUR	Average Resource Utilization Ratio
CDC	Cloud Datacenter
CRM	Customer Relationship Management
CSP	Cloud Service Provider
CT	Completion Time
dT	Deadline of Task
ETC	Expected Time to Compute
GoCJ	Google Like Cloud Jobs
HCSP	Heterogeneous Computing Scheduling Problems
MI	Million Instructions
MIPS	Million Instructions Per Second
OG	Overall Gain
PM	Physical Machine
RADL	Resource-Aware Dynamic Load-balancer
RTL	Rejected Task List
SMEs	Small and Medium size Enterprizes
SPQ	Suitable Position in VM tasks Queue
S-Scheduler	Sub-Scheduler
sz	Size of Task in MI
VM	Virtual Machine
VMS	Set/List of VMs in a Cloud data center

Symbols

Cloudlet	Notation for a task in CloudSim
CT_c	Completion time of candidate task to be shifted
CT_{ij}	Completion time of task T_i on VM_j
dT_c	Deadline of candidate task to be shifted
dT_i	Deadline of task T_i
execTime	Task execution time on VM
gbest	Global best
gBestValue	Global best value
gbFMap	Global best based final hashMap
Heterogeneous	VMs with different computation capabilities in MIPs
Itr	Number of iterations
maxItr	Maximum number of iterations
$minCT_{ij}$	Minimum completion time of task T_i on VM_j
$MindT_i$	A task with a minimum deadline in the task queue
$newCT_c$	Completion time candidate task after shifting
P	Particle Position
pbest	Personal best
pBestValue	particle best value
pbMap	Personal best based particle hashMap
pList	Parameters list Velocity
TCT	Task Completion Time Velocity
tList	Tasks list Velocity
V	Particle Velocity
VMRTMap	HashMap that store VM ready time

W	Inertia weight
PosTc	Position of candidate task
vmQ	Virtual machine Queue
newCTk	Completion time of kth task lies behind
TETMap	Task execution time Map

Chapter 1

Introduction

1.1 Overview of Cloud Computing

The idea of distributed computing [1] has emerged to enhance the processing capability of computers with evident benefits. Distributed computing covers both traditional (non-service) oriented and services oriented [2, 3] computing paradigms [4]. The distributed computing evolves into Grid, Cluster [5], and Cloud computing [1]. Grid computing links distinct computers to form a single infrastructure. In [6], grid computing has defined as "*a system that coordinates resources which are not subject to centralized control, using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service*". Grid computing work on collaboration based, having decentralized control, access is more transparent to end user, hard to manage, having a rigid payment mode, and limited support for Quality of Service (QoS) parameters [7]. Grid computing works for service-oriented and non-service oriented applications. The fast development of processing and storage technologies owing to the internet has made the computing resources powerful, easily available and economical to use [8]. This rapid growth in the technology has motivated the start of a new computing era known as Cloud computing [8]. Cloud computing has evolved from Grid computing and is considered as a user friendly version of Grid computing [4] This is because Cloud computing provide hardware level abstraction to the cloud users. Cloud comput-

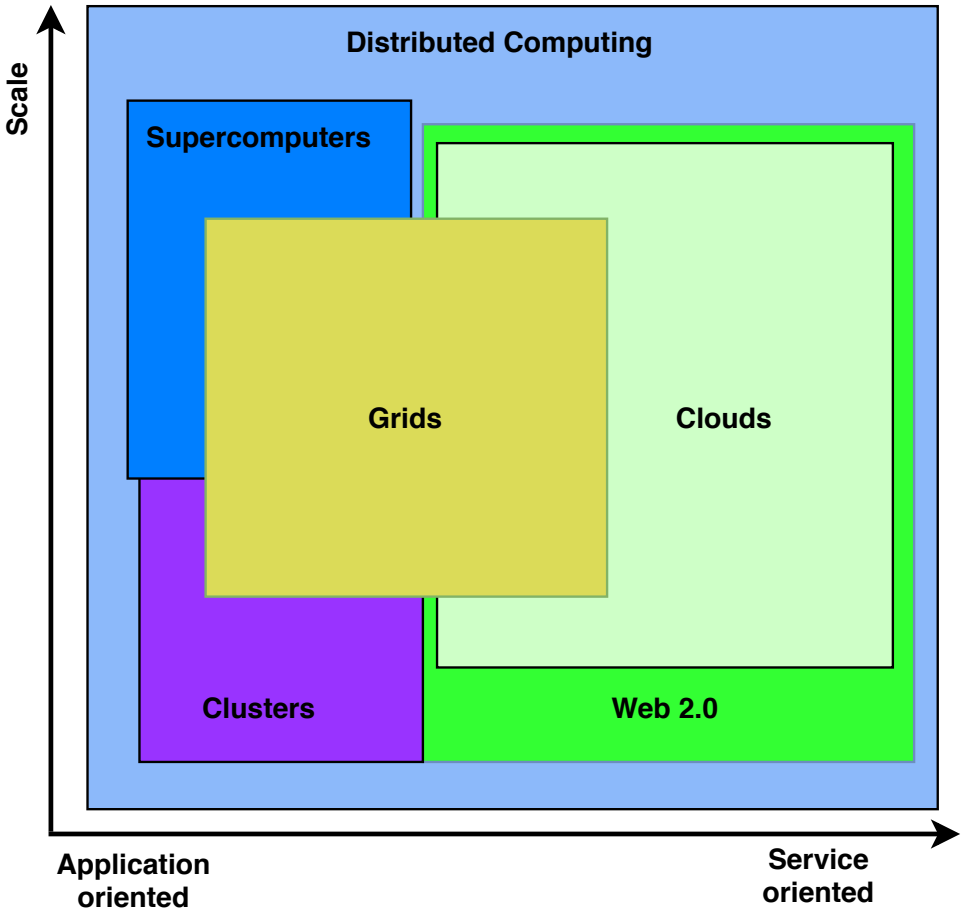


FIGURE 1.1: Clouds and Grids overview

ing is defined by [4] "*A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet*". As compared to Grid computing, cloud is a service-oriented computing paradigm which provide high level hardware abstraction, simple, flexible and user-friendly computing environment.

Figure 1.1 shows overlapping relationship of clouds with other domains of distributed computing. Cluster computing and supercomputing focus only on traditional non-service oriented applications, however, supercomputing is highly scaled than cluster and other computing paradigm. Web 2.0 covers the whole spectrum of service-oriented computing paradigm. However, cloud computing covers large scaled side of service oriented applications. Cloud computing has enabled end users to use resources like CPU and storage for a particular time based on their needs. In this model of computing, there are two key actors [6] (as shown in Figure

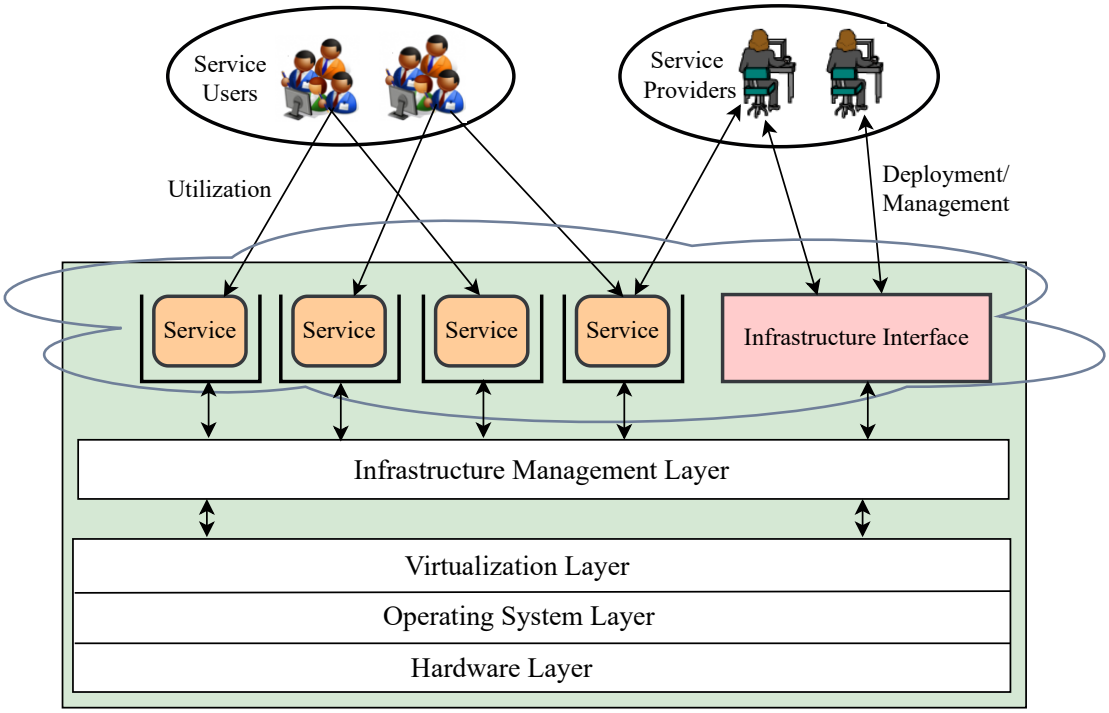


FIGURE 1.2: Cloud Actors [6]

1.2) 1) *Cloud Service Provider* (CSP): cloud service providers deploy the resources like storage and CPU etc. 2) Service users: Service users hire the services provided by the service providers for their temporal needs.

Service providers use internet based-interfaces to make their services accessible to service users. As compared to Grid computing [6, 9, 10], cloud computing provides auto resizing of the virtualized hardware resources which need dynamic reconfiguration in an automatic manner. Infrastructure management interface [9, 11] is an *Application Programming Interface* (API) [11] specification used for cloud infrastructure management. The infrastructure management layer, manage Cloud infrastructure and act as bridge between user services layer and infrastructure layers. Cloud infrastructure layer includes Virtualization layer, Operating system layer, and Hardware layer.

1.1.1 Cloud Services Model

Cloud computing is highly adaptive for diverse needs. To meet these diverse requirements, cloud computing model is implemented in various ways, using different technologies, and provide a number of services [12] like:

- *Infrastructure-as-Services* (IaaS)[11, 13, 14]
- *Platform-as-Services* (PaaS)[11, 13, 14]
- *Software-as-Services* (SaaS) [6, 13, 14]

In the last few years, cloud computing has a high impact on the industry of Information Technology [8]. According to *Gartner Predictions* [15], the market revenue of cloud infrastructure services will grow by 176% in 2021. Because of such benefits of cloud [16], many companies like Microsoft [1], Google [2], and Amazon [3] have started providing powerful, reliable and cost-efficient services to the customers. In SaaS [14], the cloud Service Providers (CSP) provide special purpose softwares to the cloud users (i.e., consumers) which are deployed and running on cloud Infrastructure. Live Mesh from Microsoft and Customer Relationship Management from Saleforce.com are examples of SaaS services. The online alternative to the Microsoft office application i.e., word processor is an example of SaaS services. In SaaS, cloud users are not responsible for the maintenance and management of issues related to the cloud Infrastructure like servers, OS, Networks, development and deployment tools and other applications [6, 13].
PaaS [14] is an other obstruction level offered by the CSP where consumer (cloud user) deploy their own applications developed in any programming language that is supported by CSP provided environment. In PaaS, the cloud user control deployment, hosting, and configuration of user-created applications. Microsoft Azure and Google App Engine are examples of PaaS services. However, in PaaS, cloud users are not responsible for maintaining and managing the cloud Infrastructure like managing OS, servers configuration, and Storages.

In Infrastructure-as-a-Service (IaaS) [11, 14], the Cloud service providers manage and maintain a huge set computing resources such as processing and storage capacity. Virtualization of these resources enables Cloud service providers to split physical resources and build dynamically resizable ad-hoc computing systems. Moreover, virtualization provide scalability in terms of run-time lease and release

[1] https://azure.microsoft.com
[2] https://cloud.google.com
[3] https://aws.amazon.com

of virtual resources and high level of customization according to the users requirements. The Cloud Service Provider (CSP) provide these scalable resources to the consumers based on user demand.

In IaaS, the consumer acquires the CSP provided storage services, processing power, network bandwidth, and other essential resource to run their operating systems and other applications. In this model, consumer is not responsible for managing physical infrastructure of the cloud. OpenNebula, Eucalyptus, Amazon Web Services are the examples of IaaS services.

1.1.2 Cloud Architecture

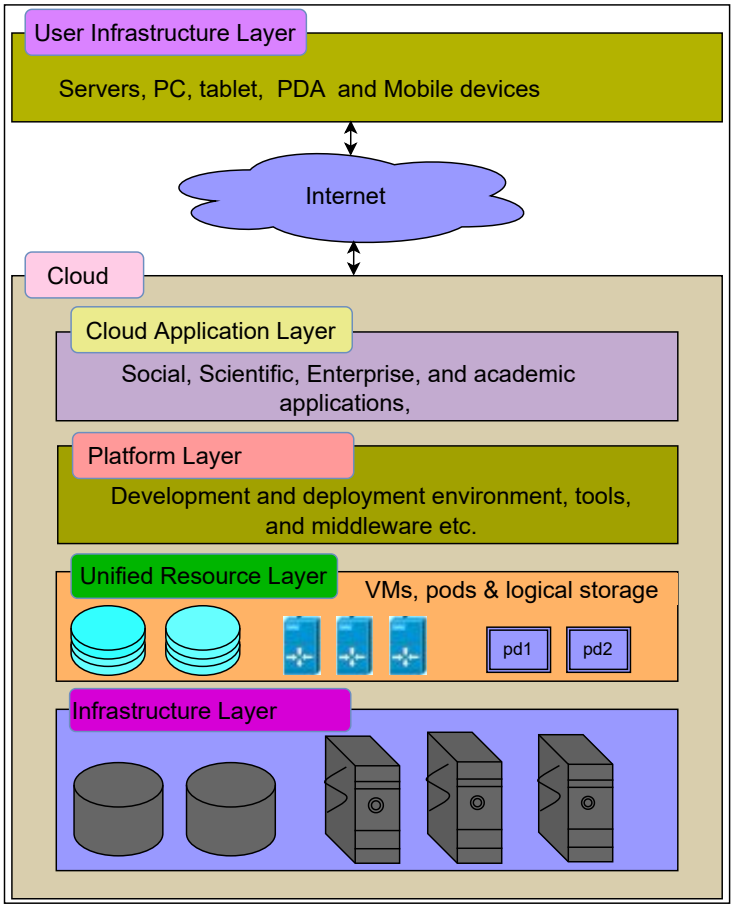


FIGURE 1.3: Cloud Architecture [4, 11]

Figure 1.3 depicts basic design of the cloud related environment. User infrastructure is considered as front end for cloud users. This component of cloud environment comprises of user end servers, PCs, tablets, PDAs, Mobile devices

etc. that is used by the cloud users to access the cloud computing system. Cloud users connects to the Cloud through Internet. Cloud is categorized into four basic layers [4] (shown in Figure 1.3).

Application Layer consists of applications that are directly available to the cloud users. Cloud users are the active users that utilize the Software-as-a-Service (SaaS) applications over the Internet. Cloud service providers may acquire these resources and made available to the cloud user on pay-per-use mode. However, cloud users either hire these applications like Customer Relationship Management (CRM) softwares or deploy their own applications like e-research [17] and e-science [18] among others [11].

Platform layer implement platform level services and provide runtime for hosting, development, deployment,and managing user level services. The most important services at this layer are services discovery, resource management, scheduling, and load-balancing. Unified resource layer comprise of virtual machines, pods, and logical/temporary storages. This layer provide abstraction of the physical resources layer. Cloud Infrastructure layer consists of physical resources that includes physical storage devices and large number of host machines and/or servers. The number of physical storages, servers, host machines depends on size of cloud datacenter.

1.1.3 Cloud Deployment Model

From deployment point of view, clouds computing environments are divided into three different categories which include public cloud, private cloud, Hybrid cloud [14]. This categorization is based on the type of ownerships and right to access the cloud resources.

Public cloud resources are available to every interested users on the Internet and on pay-per-use mode. Public cloud is used by small organizations and individuals who need computing resources on ad-hoc basis. Its because these organizations and individual can't invest such huge amount for purchasing computing resource for their ad-hoc use. On the other hand, large or medium size organizations and governments offices provision public cloud resources for temporal usage along with the maintaining their own private clouds due to the temporal variations on their

service demands [11].

Just like public cloud, access to the private cloud is not available to every one on the Internet. Private cloud is only accessible to cloud users within the organization that own the cloud. Private cloud is own by the organizations and enterprizes like banks etc that need massive computing resources on routine basis.

A hybrid cloud is combination of public cloud and private cloud. This model of cloud is adopted by small and medium size of enterprizes and Govt. organizations for their private use. Along with maintaining their own private cloud, these enterprizes use public cloud temporarily to handle temporal increase in the computing resource usage. Hybrid cloud model is beneficial for govt. organizations and Small Medium Enterprizes (SMEs) that need huge computing power and storage at specific day and time [11].

1.2 Cloud Scheduling and Load Balancing

To deliver better services, there is a need to utilize the available cloud resources in a balanced way [19]. This leads to the importance of selecting the preeminent scheduling algorithm for allocating cloud resources. The main objective of the cloud scheduling algorithms is to allocate a job or task to the most suitable cloud resource.

1.2.1 Tasks Scheduling

There are two levels of mapping involved in the cloud (as shown in Figure 1.4). 1) VM to host mapping that maps VMs on the underlying host machines and 2) Task to VM mapping which allocates tasks to the respective VMs. This research focus on tasks to VMs mapping, where the number and computation power of VMs are pre-determined.Task scheduling is an important factor in cloud computing and a principle means of resource allocation.

Task scheduling is the way of choosing the best suitable available resource and is one of the most challenging issues in the cloud. The task scheduling affects the performance of the cloud and needs to be optimized. The objective of task

scheduling is to balance the load among different resources, improving the utilization of resources, makespan, throughput, and to reduce tasks waiting time. However, task scheduling becomes a more challenging problem when the number of tasks and resources increases [19–22] and increase in the number of users requests for limited computing resources.

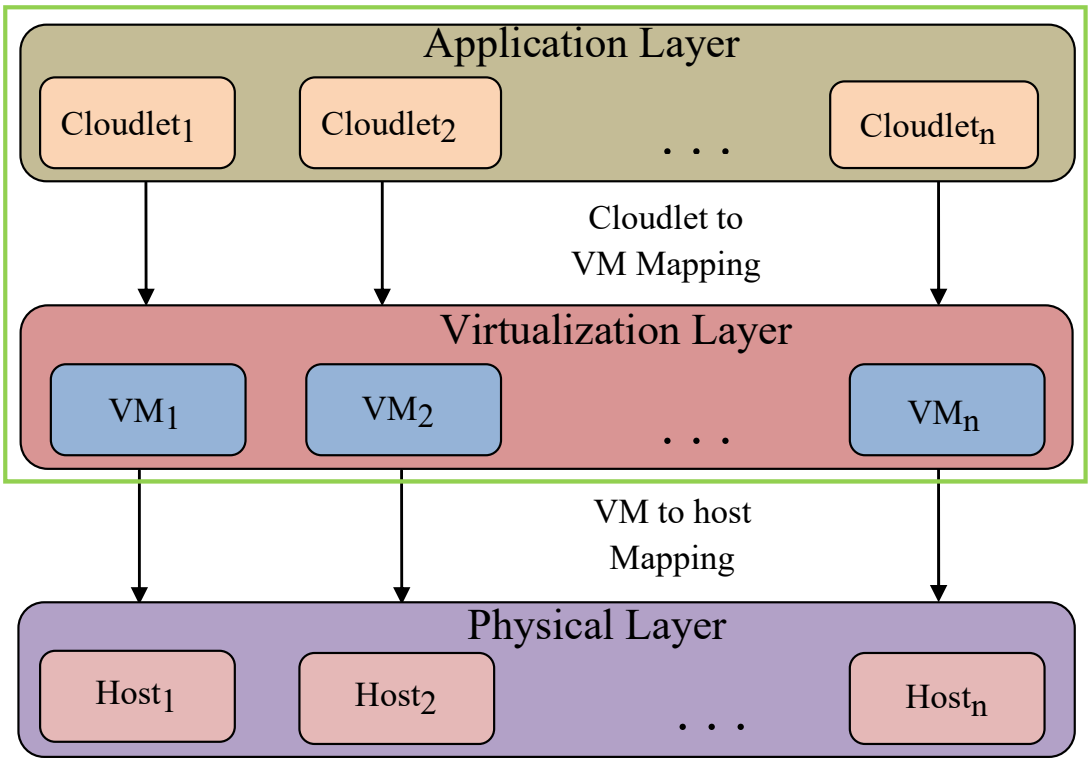


FIGURE 1.4: Levels of Cloud Scheduling

Additionally, the infrastructure management interface layer of the cloud computing framework provides the facility to resize the virtualized hardware resources that need dynamic and automatic reconfiguration [9, 11].

In cloud task scheduling, there is a possibility to assign a large-size task to a slower VM that can increase the overall execution time of the cloud task. Moreover, assigning small-size tasks to faster VMs can increase the task-switching overhead, waiting time for larger tasks, and increase the possibility of task rejection [23]. These problems reducing the overall performance [1] of the cloud and imbalance

the load on VMs. Load balanced [1, 20] task scheduling [22] plays an important role to enable optimal use of the cloud resources and load balancing is discussed in section 1.2.2.

1.2.2 Load Balancing

Load balancing is the distribution of workload among a set of heterogeneous resources according to their computation power. Load balancing is the mapping of workload among a set of resources in such a way that all the resources complete execution of the assigned workload nearly at the same time. It helps in achieving better user satisfaction and improving higher utilization of cloud resource. Load balancing can be at network level, host level and VM level. Network level load balancing means distributing the incoming traffic evenly among different communication channels to reduce unnecessary transmission delay. Host level load balancing enables evenly distribution of workload among various physical host machines. It helps in achieving high utilization of physical resources and avoids over provisioning of any particular host machine.

1.2.3 Types of Task Scheduling

Several task scheduling algorithms have been proposed in the literature. To effectively address the tasks scheduling challenges, task scheduling algorithms should be based on a well-defined set of rules/constraints (heuristics) [24, 25]. These sets of rules/constraints can problems specific or general. There are two broad categories of task scheduling that are Heuristic and meta-heuristic-based task scheduling algorithms. Figures 1.5 and 1.6 show prominent categories and types of task scheduling algorithms in cloud computing.

Heuristic-based algorithms are designed for tackling a specific problem and are known as problem-dependent approaches [7]. The reason for using heuristic-based approaches is that they find a satisfactory solution within limited time and more efficiently. These algorithms are easy to implement and bear low computational cost [26]. Heuristic-based [20] algorithms are more suitable for online mode of task

scheduling as these tasks need a quick system response. Greedy algorithm is a type of heuristic-based algorithm that tends to find a good solution for continuously arriving tasks quickly [27]. The main objective of the heuristic-based algorithm is to reduce the execution time without considering the cost [7].
Heuristics based cloud task scheduling approaches (as shown in Figure 1.5) are categorized into three types.

1.2.3.1 Static Scheduling

Static scheduling [28, 29] is the simplest type of scheduling that maps all the tasks before starting their execution. All information about active resources and jobs must be known in advance to allow allocation of tasks to resources.

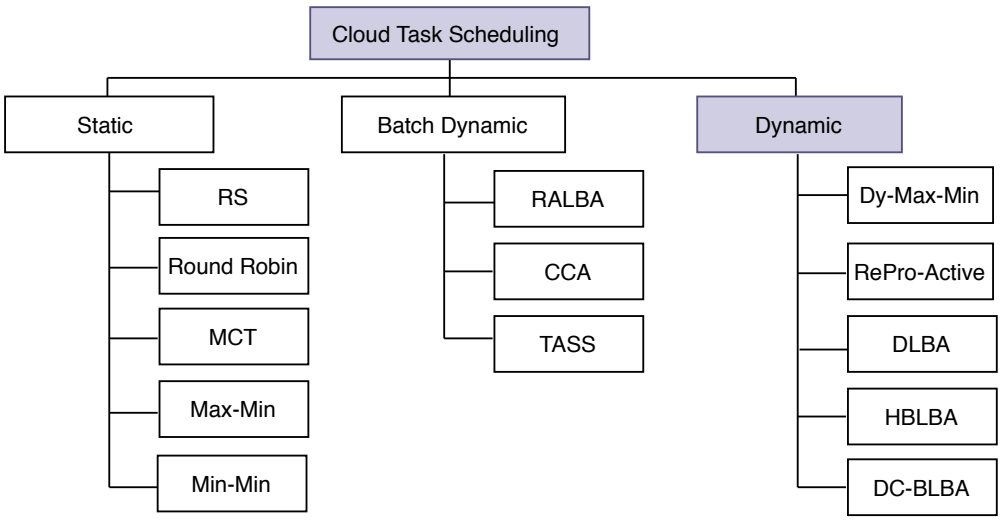


FIGURE 1.5: Heuristics-based Task Scheduling Types in cloud [19, 36]

In static scheduling, once a set of tasks are mapped to the corresponding VMs, their mapping will remain unaltered until the completion of the assigned tasks (the tasks once assigned to the VMs cannot be revoked or altered at run time). Moreover, most of the classical [8, 23, 30, 31] and state-of-the-art static scheduling heuristics [1, 8, 10, 19, 32, 32, 33, 33–35] suffer from issues like poor resource utilization, load imbalance, and unable to meet the deadline constraints for the real-time tasks.

1.2.3.2 Batch Dynamic Scheduling

Batch dynamic scheduling techniques [1, 10, 37] map a batch of tasks on a predefined number of VMs. All tasks in a single batch are statically mapped to VMs before their execution starts. The number of VMs can be changed (increased or decreased) for the next batch based on the computation requirements of a batch. These task mapping approaches result in issues like delayed response time for the new batch formation (i.e., the task which arrives first will have to wait till the formation of the complete batch) and inter-batch under-utilization of resources (i.e., in circumstances where the execution of the first batch is finished and the next batch not formed yet).

In batch dynamic scheduling, tasks of new batches are allocated to virtual machines without considering the current workload of these resources. Therefore, the newly created virtual machines for the latest batch can finish the assigned workload earlier than the existing virtual machines that are busy in executing the previous workload. In batch dynamic based scheduling heuristics, if a task with shorter deadline than the already mapped tasks arrives in later batches, and all the VMs are already overloaded, the newly arrived task will have to wait until the completion of already mapped tasks. This results in violation of task deadline and high task rejection ratio.

Most of the batch dynamic scheduling heuristics [19, 34, 35, 37, 38] provide dynamism at the batch level only and suffer from issues related to batch formation delays, and under-utilization of resources (during the batch formation).

1.2.3.3 Dynamic Scheduling

Dynamic scheduling mechanisms are much more flexible than the previously described categories [23, 28]. Dynamic schedulers have the capability to check, estimate, and update the VMs load during the execution of the tasks either in the reactive or proactive manner [19]. These schedulers can allow prioritization and migration [33, 38] of the already assigned tasks. Moreover, the dynamic scheduling mechanisms generally have the capability of new VMs creation [20, 34, 39],

removal the existing VMs and migration of VMs [40] at run time. However, a number of the existing dynamic scheduling approaches [23] still use an interval-based temporal batch of input tasks that arrived during a particular time period. Majority of the existing dynamic algorithms [19, 20, 33] suffer from the issues like lower resource utilization, load imbalance, and high rejection ratio for the deadline-based tasks[19, 21, 32, 33, 41]. Moreover, most of these approaches do not consider tasks deadlines [33, 34] and lack the task shuffling mechanism to accommodate deadlines.

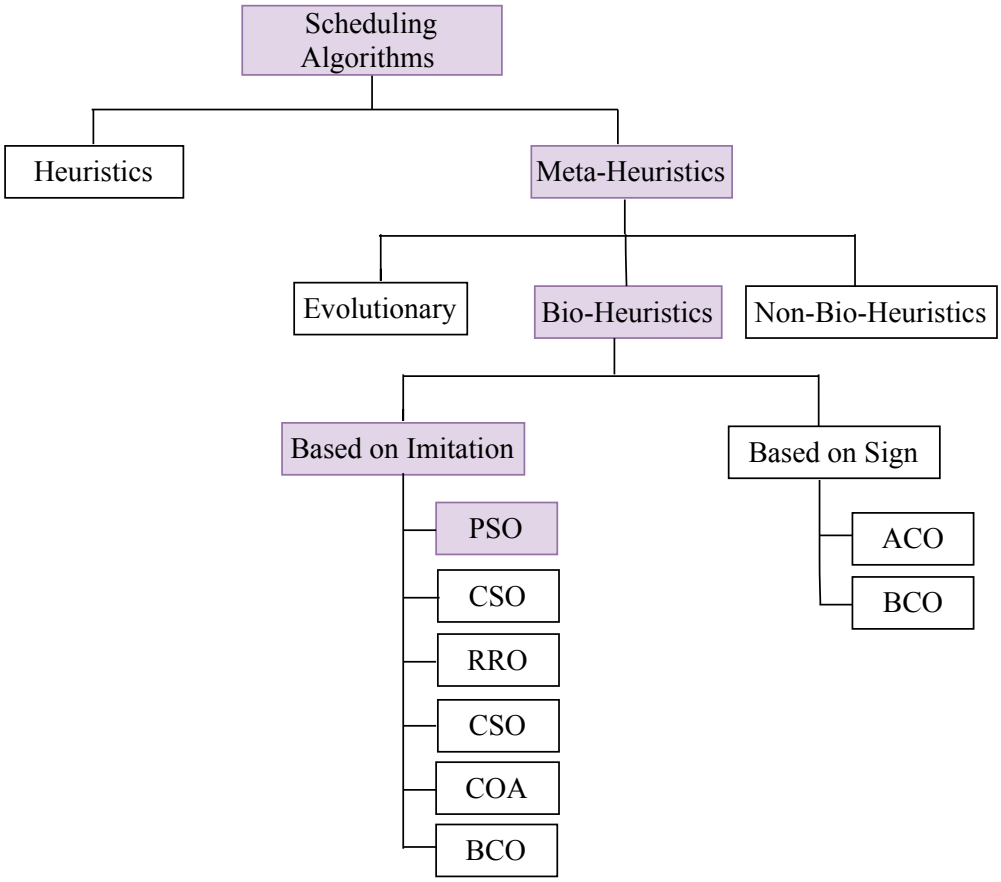


FIGURE 1.6: Meta-Heuristics based Task Scheduling Types in Cloud [19, 41]

1.2.3.4 Meta-heuristic Algorithms

As compared to heuristics-based approaches, *Meta-heuristic* [25] algorithms are problem independent (shown in Figure 1.6). Meta-heuristic algorithms are designed in such a way that can be applied to more than one type of problem. These algorithms are applicable to various domains with optimal performance. Meta-heuristic based algorithms follow a standard set of procedures to solve the

problem. Currently Meta-heuristic algorithms are used for solving problems in various domains like health care [42], diagnosing diseases [42], stock analysis [43], academics [39], fraud and intrusion detections, feature selection [44], solving real-world engineering problems [45], pipe and road problems [45], data classification [46] distributed computing and cloud computing [7]. Meta-heuristic algorithms are classified as Trajectory based (i.e., Simulated Annealing (SA) among others) and population-based like Particle Swarm Optimization (PSO). The most prominent meta-heuristic based task scheduling algorithms include Ant Colony Optimization (ACO) [47, 48], Bee Colony Optimization (BCO), Raven Roosting Optimization (RRO) [49] algorithms, Improved Raven Roosting Optimization (IRRO) [41], Cat Swarm Optimization (CSO) [50], Chicken Swam optimization (CSO)[51], Genetic Algorithm (GA) [52, 53], Particle Swarm Optimization (PSO) [7, 26, 54, 55], honeybee foraging [56], and Simulated Annealing (SA) [57]. ACO based meta-heuristic algorithms have better optimization at early stages, however, the convergence rate of ACO is comparatively slower. As compared to GA, PSO has an easy implementation, fast convergence, and better optimization performance. Particle swarm optimization based algorithms are more popular among these meta-heuristic based algorithms due to its effectiveness for a broad range of applications, simplicity, fast convergence, and easy implementation. Moreover, PSO based algorithms have a sound natural computation background along with better performance.

Computational time is the most important factor in task scheduling in the cloud. Its because cloud is a dynamic computing environment and the cloud scheduling algorithm should be fast enough to be adopted in the real cloud environment. Moreover, these algorithms should provide an optimized solution with fast convergence.

1.3 Motivation

Cloud computing has evolved from distributed computing and become an effective computing platform [4]. Cloud computing allows parallel execution of user's tasks using Virtual Machines (VM). A virtual machine is the processing unit of the cloud that enables faster execution of user's tasks and reduces task response time [58].

To observe the behavior and effect of using a single VM on a single host machine against multiple VMs on the same single host, several experiments have been performed. For these experiments, 100, 250, 500, 750, and 1000 tasks have been used and compared Single Host Single VM (SHMV) and Single Host Multiple VMs (SHMV) in terms of total execution time (makespan) and average task response time (as shown in Figure 1.7).

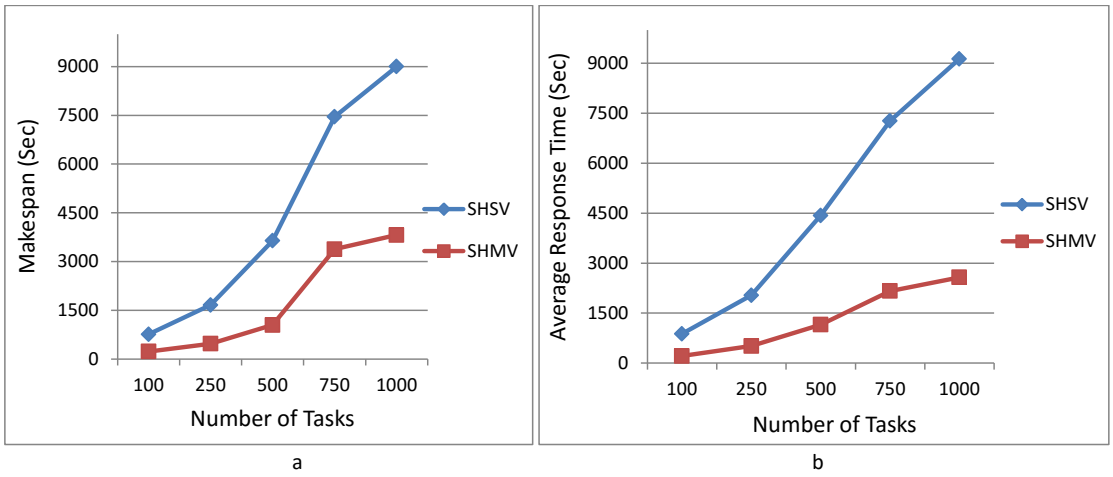


FIGURE 1.7: Makespan and Task Response Time based comparison Results

The experimental result presented in Figure 1.7 shows that SHMV has outperformed in all sets of tasks as compared to SHSV. These results reveal that has attained up to 153% and 215% reduced total execution time (as shown in Figure 1.7 (a)) and average task response time (as shown in Figure 1.7 (b)) respectively. However, balancing the load on the multiple VMs and efficient utilization of virtual resources become more challenging with the increase in the number of VMs used for parallel execution of user's tasks. To overcome these challenges, the role of task schedulers becomes more important. Therefore, this research focuses on improving the performance of cloud task schedulers in terms of meeting user's QoS requirements like minimized makespan, enhanced average resource utilization ratio, reduced task response time, and improved meeting task deadline.

1.4 Problem Statement

The task scheduling aims to enhance user satisfaction by early execution of users' tasks and to improve utilization of cloud resources. The task scheduling becomes

more challenging when the number and diversity of user requests (tasks) increase and to map these tasks on limited computation resources. To overcome the challenges of tasks scheduling, a number of task scheduling techniques have been proposed by different researchers (as presented in chapter 2). These scheduling techniques can be static, batch dynamic, and dynamic.

Static scheduling approaches assign tasks to the VMs statically i.e the tasks once assigned to the VMs can't be revoked or altered at run time. These approaches are unable to dynamically handle any delay in the execution of a particular task. Most of these heuristics suffer from issues like poor resource utilization, load imbalance [1, 23, 33, 38, 59], unable to meet the deadline of newly arrived deadline-based tasks, and the overall performance gain not supported.

Batch dynamic scheduling [1] heuristics provide dynamism at the batch level. In batch dynamic approaches, the number of VMs can be changed for the new batch based on the computation needs of the new batch. However, these approaches suffer from issues like 1) New batch formation-based tasks response time delay where the tasks which arrive first will have to wait till the formation of the complete batch, 2) Inter-batch under-utilization of resources where the execution of the first batch is finished, however, the next batch is not received. In batch, dynamic approaches tasks in new batches are assigned without considering the incomplete tasks of the previous batch where newly created VMs can finish their assigned tasks early than others. This results in the under-utilization of resources. Tasks with shorter deadlines will suffer when VMs are already overloaded and overall gain not computed.

Dynamic scheduling algorithms provide more flexibility than static task scheduling and batch dynamic scheduling algorithms. Dynamism can be achieved in many ways like real time or just-in time based task mapping, run time lease and release of VMs, self adaption, task migration, task sequencing, and updation of VMs status among others. However, majority of the existing dynamic task scheduling approaches [19, 20, 33] still use an interval-based set/group of input tasks arrived during a particular time period. Therefore, these approaches also suffer from issues like new batch formation based response time delay and under-utilization of Cloud resources.

These approaches have the issue like under-utilization of cloud resources, load imbalance, and high task rejection ratio [33, 34]. Most of these approaches do not consider tasks deadlines [19, 21, 32, 33, 41], task shuffling, and overall performance gain of the Cloud. Moreover, majority of these approaches are not resource aware. To overcome issues like under-resourced utilization [19–21, 35], load imbalance [19–21, 35], high makespan [33], mapping of deadline-based tasks [19, 21, 32, 33, 41], and high tasks rejection ratio [33, 34], there is a need to develop a resource-aware cloud scheduler that should have the capabilities to distribute the incoming workload in a balanced manner, improve the resource utilization, and minimize the makespan.

This is because the resource aware schedulers have the ability to distribute the workload according to the computation capacity of the computing resources and update the resource load at run time. Moreover, the proposed scheduler will be capable to accommodate the newly arrived tasks with shorter deadlines than the deadlines of already mapped tasks and reduce the tasks rejection ratio using the existing cloud resources.

1.5 Research Questions

1. How to distribute the incoming workload in a dynamic and balanced manner to improve cloud resource utilization with minimized makespan?
2. How to minimize the response time of incoming deadline-constrained tasks?
3. How to map the newly arrived deadline-constrained tasks to meet the deadlines and to reduce task rejection ratio?
4. How to improve overall performance of the Cloud?

These questions has been answered in the subsequent chapters. Questions 1, 2, and 3 have been answered in chapters 3 and 5. Chapter 4 presents answers of research question no 4.

1.6 Objectives and Significance

Cloud has become an attractive computing paradigm for both *Cloud Service Providers* (CSP) and cloud service users. According to the Gartner [15] forecasts, the market revenue of cloud Infrastructure services will grow up to 176% by 2021. Moreover, Gartner predicts that the end-users public cloud spending will increase by 168 million US dollars in 2022 and in 2026 the public cloud spending will be more than 45% of all Information Technology (IT) spending [60]. This will increase user's resource demand and meeting cloud user's Quality-of-Service (QoS) needs become more challenging.

To get the full benefit of cloud resources and acquire high user satisfaction by executing user tasks within the deadline, CSPs required efficient task scheduling algorithms. For this purpose different task scheduling heuristics have been proposed in the literature in which under-resourced utilization and load imbalance are the key performance issues. Moreover, high task rejection is another important factor for cloud user's dissatisfaction. To address these issues, there is a need for improved cloud tasks schedulers which have the capability to achieve higher resource utilization, load balance, and scheduling for newly arrived tasks with shorter deadlines. To achieve these objectives, heuristics, and meta-heuristics based task scheduling algorithms can be used. However, as compared to meta-heuristics, heuristic-based approaches are simple, fast, and have easy implementation.

The heuristics-based task scheduling algorithm is considered more suitable for providing fast and near-optimal solutions for non-conflicting scheduling objectives. According to the assumption of RADL technique (supported by the literature study) that the scheduling objectives like ARUR, makspan, task response time, and task rejection ratio are non-conflicting. This is because by improving any of these objectives may not directly affect the performance of others. Therefore, a heuristic-based task scheduling technique has been used to achieve these objectives.

1.7 Research Contributions

The major contributions of this research are:

- In-depth critical analysis of static, batch dynamic, and dynamic state-of-the-art heuristic-based task scheduling approaches to identify the strengths and limitations of these approaches. Moreover, a rationalized descriptive review and critical analysis of state-of-the-art meta-heuristics based task scheduling algorithms to identify their application types, scheduling objectives, and limitations of these algorithms.

- Empirical evaluation of the state-of-the-art heuristic-based tasks scheduling algorithms has been performed. Moreover, experimental evaluation of most prominent and state-of-the-art inertia weight strategies for particle swarm optimization based algorithms. The detailed literature survey and empirical evaluation presented in this work (shown in chapter 2) provide a baseline for the resource-aware dynamic load balancer for cloud computing.

- A novel heuristics based dynamic load-balancing scheduler for independent, non-preemptive, and compute-intensive cloud tasks that produce improved resource utilization, reduced task rejection, lower makespan, minimized response time, and overall performance gain of the cloud datacenter (presented in chapter 3 and 4).

- A Particle Swam Optimization (PSO) based novel dynamic load-balancing scheduler for non-preemptive, independent, and compute-intensive tasks-based cloud workload that produces lower task execution time, improved resource utilization, reduced task rejection, and minimized tasks response time.

- Multi-objective based task mapping framework for conflicting parameters like task deadline, task response time, makespan, and cost (presented in chapter 5).

- PSO-RADL scheduling scheme has been proposed for minimizing the tasks penalty cost and tasks execution cost of virtual machines for the Cloud datacenter (discussed in chapter 5).

- A novel inertia weight strategy named Leaner Descending and Adaptive Inertia Weight (LDAIW) is designed and developed that improves the per-

formance of PSO-based algorithms concerning makespan, throughput and ASRUR.

- A novel normalization technique for computing overall performance gain of the cloud that overcomes the limitations of state-of-the-art normalization approaches (chapter 4, Section 4.3.1).

- Empirical investigation and performance evaluation of the proposed scheduling approach against state-of-the-art scheduling heuristics (chapters 3, 4, and 5, Sections 3.3, 4.3, and 5.3). Experimental results reveal that the proposed approach outperform as compared to state-of-the-art task scheduling algorithms in term of higher resource-utilization, lower task rejection, improved makespan, minimized task response time, task penalty cost and tasks execution cost, and improved overall gain.

1.8 Research Evaluation

To empirically evaluate the performance of the proposed approach, the Cloudsim [11, 61] simulator has been used. *Cloudsim* is a renowned open-source Java-based simulator used for performance analysis and modeling of cloud environment and services. The reason for using simulation environment is that using real cloud environment for evaluation of the proposed approach is costly and sometime need expert to configure the cloud environment. Moreover, performing evaluation in real cloud environment incur high monitory cost. This limits the thorough evaluation of the proposed and other contemporary approaches. Similarly, reconfiguration and maintaining a variety of heterogeneity levels even become more expensive. Our implementation extends some of the existing classes like Datacenter, Broker, and cloudlet of cloudSim simulation environment. For experiments, three scientific benchmark datasets are employed which has been published in RALBA [1], GoCJ [59, 62], and HCSP [62]. The first benchmark datasets named as *Synthetic Work-load* [1], which comprise of tasks with different sizes (in MIs) and VMs with heterogeneous computation capability (in MIPs). The second benchmark dataset is a Google like realistic workload published in GoCJ [59] (as shown in Figure 3.3

(b)). The third benchmark dataset HCSP [62] is an Expected Time to Compute (ETC) model in the form of HCSP instances. A simulation environment for all experiments of Synthetic Work-load based and GOCJ benchmark datasets consists of 50 VMs, 30 hosts, and a Datacenter [1]. The simulation environment used for HCSP comprises of 16 Virtual Machines, 30 host machines, a Datacenter.

A number of influential parameters has been used for evaluating and comparing the proposed scheduling algorithm. These evaluation parameters include makespan [1, 19, 34, 35, 63], %age of task rejection [34], Mean ARUR [1, 19, 22, 33, 63], average response time [19, 29, 33, 35], penalty cost [26], and total tasks execution cost [26]. However, most of the time improving one evaluation parameter may effect the performance of other i.e., executing tasks with shorter deadline before tasks with longer or no deadline can increase response time of tasks with longer deadline. This increases the importance to evaluate the overall performance of task scheduling algorithm as well. Overall performance represents the combined effect of the scheduling algorithm for more than one evaluation parameter (inversely related ones). To compute the overall performance of the cloud tasks scheduling algorithms, the results of different parameters need to be normalized. In [36], we have proposed a novel normalization technique for cloud task scheduling which overcome the limitations of state-of-the-art normalization techniques (as shown in equation 4.6 [36]).

1.9 Thesis Organization

The rest of the document is organized as follows:

Chapter1:

Chapter1 presents an overview of cloud, cloud services model, cloud Architecture, cloud deployment model. Task scheduling levels and types i.e., heuristics (static, dynamic, batch dynamic) and meta-heuristics have been discussed in chapter1. This chapter also describes task scheduling issues, problem statements, and research objectives. The contribution of this research is also part of chapter1.

Chapter2:

Chapter 2 provides detail of existing state-of-the-art cloud task scheduling algo-

rithms and highlight limitations and task scheduling issues in cloud Computing. Moreover, this chapter also discusses various types of scheduling algorithms (i.e., heuristic and meta-heuristics based), scheduling objectives, strengths, and limitations of state-of-the-art task scheduling algorithms. Furthermore, the summary and gaps in the studied literature have been discussed in chapter 2.

Chapter3:

Chapter 3 presents the first contribution of this thesis i.e., RADL scheduling technique. Chapter 3 also delineates RADL system architecture, RADL system model, RADL algorithm, complexity analysis, and scheduling overhead analysis of RADL scheduling approach. The experimental setup, workload generation, performance evaluation, and simulation results of RADL scheduler are discussed in chapter 3.

Chapter4:

Chapter 4 presents the second contribution of this thesis i.e., OG-RADL scheduling technique. Chapter 4 also discusses the system architecture, system model, OG-RADL algorithm, complexity analysis, and scheduling overhead. The experimental setup, workload generation, performance evaluation, and results of OG-RADL scheduler are discussed in chapter 4.

Chapter5:

The third contribution of this thesis i.e., RADL-PSO scheduler has presented in chapter 5. Chapter 5 also delineates the system architecture, system model, RADL-PSO algorithm. Chapter 5 presents the experimental setup, workload generation, performance evaluation, and results of PSO-RADL.

Chapter6:

Chapter 6 concludes the thesis and presents potential future directions.

Chapter 2

Literature Review

2.1 Introduction

This chapter presents state-of-art task scheduling algorithms in cloud computing and critically analyzed them by highlighting their strengths and weaknesses. Task scheduling is a significant element of the cloud environment and a challenging task especially when a large number of tasks need to be mapped efficiently on limited Cloud resources. Task scheduling is considered as an NP-hard problem [19, 20, 63]. To solve such problems, a number of heuristics and meta-heuristics based scheduling algorithms proposed in the literature. Some of these algorithms have been shown in Figures 1.5 and 1.6.
Heuristic-based task scheduling approaches provide near-optimal solutions for non-conflicting parameters in cloud computing. These algorithms have simple implementation and lower scheduling complexity and scheduling overhead. The non-conflicting parameters like makespan and throughput among others are the parameters where improving one parameter may not directly affect the performance of other parameters.

Meta-heuristics based task scheduling algorithms are problem independent techniques and can be applied to various domains. These schedulers are considered more suitable for optimizing multi-objective optimization problems with conflicting parameters like execution time, tasks deadline, tasks execution cost in cloud

computing.

2.2 Heuristic based Cloud Task Schedulers

Heuristic-based task scheduling approaches provide near-optimal solutions for non-conflicting parameters in cloud computing. These algorithms have simple implementation and lower scheduling complexity and scheduling overhead.

2.2.1 Static Scheduling Heuristics

Static task scheduling heuristics map all the tasks to VMs before starting tasks execution and the tasks once mapped to VMs cannot be altered during the tasks execution. The newly arrived tasks will have to wait for scheduling until the execution of already mapped tasks is completed.
Fist-In-First-Out (FIFO) is one of the simplest task scheduling heuristics in cloud computing. This technique allocates tasks to the Virtual Machines (VM) using First-Come-First-Out(FIFO) basis. FIFO [23] has less scheduling overhead and simple implementation. However, if tasks with large size (that need a longer time for their execution) are received before the smaller tasks then the smaller tasks received in later stages will have to wait for a longer time. This increase response time of newly arrived deadline-based task and may lead to deadline violation. Moreover, the long wait of user tasks can negatively impact the user experience and lead to high penalty costs for deadline violations.

One of the simplest heuristic is *Opportunistic Load Balancing* (OLB) that allocates tasks to VMs in an arbitrary manner without considering the VMs computation power and already assigned workload [64, 65]. OLB has a simple implementation, low complexity, and minimal scheduling overhead as compared to other scheduling heuristics. OLB scheduling algorithm randomly selects available VMs and assigns tasks to the selected VMs without considering there expected completion time. This results in high makespan and poor resource utilization.

Another basic heuristic is known as *Round Robin* (RR) [66] that distributes the

incoming workload in circular order on predefine number of VMs. RR has minimum scheduling overhead, simple implementation, and lower complexity than the other task scheduling algorithms. However, RR assigns tasks to the VMs in circular order irrespective of the VM computation power and task size (causing load imbalance) [1].

Minimum Completion Time (MCT) [67] algorithm allocates a candidate task to the VM which results in lowest completion time for the task. MCT considers already assigned workload allocated for finding the appropriate VM for the execution. As compared to the RR and RS scheduling heuristics, MCT improves resource utilization and reduces makespan. However, MCT assigns more tasks to the faster VMs and slower VMs remain idle which result in load imbalance [1]. Moreover, assigning more tasks to the already over-loaded machines degrades the performance of those VMs [59, 62].

Max-Min [68] scheduling is based on MCT which assigns a task to the VM that promises minimum expected completion time for that task. Max-Min scheduling heuristic initially takes a set of unscheduled tasks and VMs, and completes the scheduling process in two steps: 1) finds the earliest finish time for a task using all VMs; 2) selects the task with a *maximum earliest finish time* for mapping to the concerned VM. On each scheduling step, the heuristic removes the mapped task from the tasks list and updates ready time of the VM. The Max-Min approach favors larger jobs and penalizes smaller jobs. Moreover, Max-Min scheduling heuristic suffer from load imbalance issue for workload with a high number of large-sized tasks [1].

2.2.2 Batch Dynamic Scheduling Heuristics

In Batch dynamic task scheduling a set of task is collectively mapped to VMs although the execution previous batch is not completed.

A *Resource-Aware Load Balancing Algorithm* (RALBA) [1] provides a balanced distribution of workload according to the VMs computation capability. RALBA is a batch dynamic scheduling technique which maps a batch of independent and non-preemptive jobs on predefined number of VMs. RALBA works in two phases.

In phase1, the tasks are assigned to the VMs based on their computation capacity and computation requirement of the tasks. The second part of the proposed algorithm assigns the remaining tasks to the VMs which give the earliest finish time for executing the task. RALBA claims to achieve high resource utilization as compared to the existing static scheduling algorithms. However, it suffers from the issue like new batch formation-based processing delay, inter-batch under-utilization of the resources and unable to accommodate jobs with deadlines.

Kong et al. [69] have presented a heuristic-based load-balancing technique with zero imbalance mechanism in a cloud computing environment. The working mechanism of the proposed approach considers VM completion time, data transfer bandwidth of VMs, task execution time, and earliest finish time criteria for task scheduling and load balancing. This technique consists of task mapping along with load balancing without involving any priority method. The reason is that including priority methods can increase the computation overhead and can reduce the efficiency of the scheduling algorithm. The proposed approach is evaluated against their counterparts in terms of resource utilization, makespan, and waiting time using the cloudsim toolkit.

Praveenchandar and Tamilarasi [70] have presented an efficient resource allocation and task scheduling scheme has proposed. The proposed approach aims to optimize the power consumption of cloud data centers, reduce task completion time, and improve task response time. The proposed approach receives input tasks in the form of batch and is implemented using a renowned Cloudsim simulator. However, resource utilization, task deadline are not considered.

2.2.3 Dynamic Scheduling Heuristics

Mao et al. [33] has proposed the Max-Min based elastic task scheduling algorithm (ECMM) for load balancing. Max-Min algorithm uses the task and VM status tables to estimate the execution time of tasks and performs real-time load balancing of VMs. VM status table shows the status of VMs which include tasks, VM Id, number of assigned tasks, execution time, VM life-cycle status, etc. Similarly, the task status table contains task Id, VM Id, task execution time, task

completion time, and last update time. The incoming tasks are split into different batches based on time interval and allocate tasks to the VMs using the Max-Min approach without considering task migration and deadlines. Additionally, the employed approach is unable to achieve improved resource utilization, load balance, and minimized makespan.

A task migration-based scheduling algorithm named TM-eFCFS is proposed by Panwar and Negi in [38] that employs *First Come First Serve* heuristic as the base algorithm. To achieve faster execution and minimized the makespan, this approach uses non-live migration of tasks in the queue (waiting for the execution turn) or partially executed task to fastest idle VMs. The proposed approach comprises of two algorithms. 1) The algorithm selects incoming tasks and assigns them to the VM which provide *Earliest Completion Time* (ECT) and update VM ready time. The *second algorithm* performs task migration in which the unprocessed or partially processed tasks already assigned to the slower VMs are migrated to a faster machine (employing preemption mechanism). The first algorithm assign tasks based on early completion time and overloads faster VMs. Idle or slower VMs are not considered for the task migration resulting in unbalanced distribution of workload that causes poor resource utilization [32].

Chen et al. [30] proposed a fuzzy control theory-based dynamic resource scheduling technique. This approach predicts the number of concurrent compute-resources required by the users using the historical information, i.e., a number of earlier requested resources, resource types, and the number of online users. The data center monitors the utilization of resources in real-time. This model is used to ensure efficiency, availability of cloud resources, and avoid system overload at peak time. The proposed approach relies on feedback and prediction model; however, such a prediction model is based on the historic resource scheduling information which is difficult to maintain.

In [35], time-efficient dynamic threshold-based load balancing technique has been proposed that targets to avoid tasking allocation to overloaded VMs [20]. In this approach, a newly arrived task is assigned to a VM only if the current load of that VM is less than a predefined threshold. The proposed algorithm also performs task migration for assigning services to the high priority task. When tasks with

short deadlines arrive, they are assigned to the fastest machine without considering the slower machines with minimum load. *Dynamic Load Balancing Algorithm* (DLBA) uses a threshold to check the VM for overloading; however, this approach ignores the under-utilized VMs, which reduce resource utilization and increase load imbalance. Also, the task rejection ratio is higher because the approach does not accommodate jobs considering the deadlines.

A two-stage strategy has been proposed [23] to minimize load imbalance and improve task scheduling performance. The first stage uses the historical scheduling data to classify the tasks using the one of the machine learning classifiers. The use of historical scheduling information helps to create a specific number of VMs types in advance to save the time of VM creation at runtime. In stage 2, the dynamic task scheduling algorithm is proposed for assigning the matched task to the corresponding VM. The key contribution of this paper is to reduce the time cost by creating VMs beforehand and employing the historical scheduling information. Moreover, the task requirements like task deadline and cost are not considered leading to the reduction in the completion time of tasks as well as load of the VMs. However, the history of past activities is not easy to maintain and process in order to make future decisions. Moreover, the accuracy of the obtained prediction may not be satisfactory [19, 59, 62].

Chen et al. proposed a dynamic annexed balance method [32] known as *Cloud Load Balancing* (CLB). The proposed CLB architecture consists of five different levels that, 1) represents the users of the cloud service request, 2) consists of cloud Load Balance Monitoring Platform (CLBMP) that checks service status (i.e. online or offline), determine load of all services, 3) is a *Cloud Load Balance Distribution Platform* (CLBDP) which receives users' requests and is assigned to the hosts, 4) represents the server load information database and also store Priority Service (PS) value of each host, and 5) consists of cloud server which provides storage from the cloud-service pool and responds to user requests. For monitoring the platform, a cloud load-balancing algorithm is employed to get a load of the hosts, PS value, and computer power, and store them in the database. CLBDP receive users' requests for cloud services and shifts them to the hosts using polling (*round-robin* based) method.

In [19], *Simulation-Based Optimization* (SBO) scheduling framework, named as RePro-Active has been proposed, which executes periodically. This approach solves issues like dependency on information on past activities and maintaining historical information. The algorithm starts from current conditions (rather than relying on the history data) and uses the SBO technique that tries to simulate possible prospective events to make better decisions. Although, it avoids dependency on the historical information; however, this approach results in low Average Resource Utilization Ratio (ARUR) as compared to Min-Min scheduling heuristic and load imbalance.

Heuristic-Based Load Balancing Algorithm (HBLBA) [20] employs configuration of host servers and tasks to VMs mapping. The aim of the host server configuration is to create the required number and types of VM instances to serve the batch of the incoming tasks. The input tasks are stored in decreasing order. To reduce the waiting and completion time of the tasks, a queuing model has been adopted for task-to-VM mapping. The length of the host server queues is fixed and the queue length of VMs is dynamic. The key metrics used for evaluation are the waiting time, makespan, *Scheduled Length Ratio* (SLR), CPU utilization of hosts, and VMs. The proposed approach receives tasks in batch mode which results in two issues like batch formation-based response time delay and lower resource utilization. The proposed algorithm allows creating as many instances of hosts with the highest computation power as the data center allows. The slower hosts remain idle due to the unavailability of computing power required for a VM, which give rise to low resource utilization and load imbalance problems.
Kumar and Sharma [34] proposed a deadline constrained dynamic scheduling algorithm which provides scalability by adding and removing VMs at run time. VMs are added based on the average number of the rejected tasks. When tasks are scheduled on the VMs then the task migration is performed from *Overloaded* VM (OVM) to *Under-loaded* VM (UVMs). Load and capacity of each VM are calculated before task migration. The OVMs and UVMs are identified and sorted in descending and ascending order respectively. The threshold of the overloaded and underutilized VMs are taken from the existing approaches and fine-tuned by performing experiments. At the end of each interval, some of the under-loaded

VMs are removed based on the average number of UVM. If the number of rejected tasks is high then the number of new VMs created is higher than required. The creation of unnecessary VMs increase scheduling overhead, load imbalance issue, and resources under-utilization issue. Furthermore, the task rejection ratio is high and the rejected tasks are not reconsidered for re-scheduling.

To reduce makespan and increase the number of tasks that meet their deadlines, [71] has proposed a flexible and elastic task scheduling algorithm in cloud computing. The algorithm has the capability to automatically scale-up and scale-down based on the incoming requests from the service users. Architecture of the proposed approach comprises of components like controller node/scheduler, load analyzer, Elastic Load Balancer (ELB), and a component that perform provisioning and de-provisioning of cloud resources.

In [72], Wang et al. have focused on the parallel execution of deadline-based tasks to improve the performance of cloud task scheduling. In the first phase, deadline aware task scheduling is modeled as an optimization problem and focused on the overall utilization of computing resources. Moreover, Wang et al. have proposed parallelism-awareness-based scheduling methods that are quickly allocated as much as possible resources iteratively to the tasks with shorter deadlines. However, the proposed approach not considering the overall performance gain of cloud.

ControCity a controlling and elasticity-based model has been presented by Ghobaei-Arani et al. [73]. The proposed framework manages the elasticity by using two-component called "elasticity management" and "buffer management". The buffer management works at the application layer and manages the user input queue. The elasticity manager manages the elasticity of the cloud platform using the learning automaton technique. The proposed approach is implemented and evaluated using a cloudsim simulator using ARUR and task response time as evaluation parameters. However, makespan and task deadline not considered.

Alworafi and Mallapa [74] have presented a Quality of Service (QoS) based model for executing user's tasks on VMs and is named as *Deadline and Budget Scheduling* (DBS). The proposed approach aims to reduce total task execution time (makespan) and task execution cost. Based on the user's needs, the input tasks are categorized into three different levels. The user's tasks with cost and budget-

constrained are assigned high priority, tasks with cost-constrained are marked as a fair priority, low priority is assigned to tasks with budget requirements only. Makespan and cost were used as evaluation parameters for the proposed approach as compared to their counterparts. The cloudsim simulator is used to implement and evaluate the proposed approach using a randomly generated set of independent tasks. However, ARUR, task response time, task rejection are not considered for the evaluation of DBS.

The elastic scheduling algorithm for micro-services has been proposed [75]. The proposed approach uses an on-demand model to combine auto-scaling and task scheduling in the cloud environment. An urgency based workflow scheduling is proposed that map tasks and identifies the quantity and type of instances to be scaled-up. The utilization of resources, tasks deadline, and virtual machine cost is considered as scheduling objective for the streaming-based workload of micro-services. However, the overall performance of the cloud is not considered.

In [76], Yazdanbakhsh et al. have proposed a Multi-Objective scheme for Dynamic scheduling with Elastic cloud resources (MODE). MODE has the capability to add and remove virtual machines dynamically in an elastic way. A limit (threshold) has been set to lease new resources or release the existing under-loaded resource. Mean utilization of VMs, makespan, task deadline violation, and total cost are the scheduling objectives of the proposed technique. The proposed approach provides high scalability and reduced task response time. However, due to high scheduling overhead, MODE has high makespan and monetary cost and minimized utilization of cloud resources.
Ibrahim et al. [63] have experimentally evaluated the most prominent static state-of-the-art task scheduling in cloud computing in terms of Average Resource Utilization Ratio (ARUR), makespan, energy consumption, Throughput. Moreover, individual VMs level load-imbalance is evaluated and compared. However, task deadlines and overall performance gain are not considered.

In [77], Shahidinejad et al. have proposed a Quality of Service (QoS) based resource provisioning technique in cloud computing environment. The proposed approach is a hybrid approach of K-means and Imperialist Competition algorithm.

The Imperialist Competition algorithm and K-means are used for clustering the user submitted workload. For efficient resource provisioning, scaling decisions are determined with the help of decision tree algorithm. CPU utilization, task response time, and total execution cost are used as evaluation parameters for the proposed scheduling technique. Cloudsim tolkit has been used for implementation and evaluation of the proposed approach using two realistic workload traces.
In [36], an overall performance based resource aware dynamic scheduling algorithm for cloud computing has been proposed. In this research, the combined effect of different evaluation parameters has been analysed. The idea is that most of time, improving one scheduling parameters can effect the performance of other.To compute the overall performance of cloud task scheduler, the values of different evaluation parameters need to normalized. A novel normalization technique has also been presented.
Nabi et al. [78] have proposed a Dynamic and Resource-Aware Load Balancing Algorithm (DRALBA) for independent and compute-intensive tasks in a cloud environment. DRALBA computes the computation share of each Virtual Machine (VM) and assigns tasks based on VM computation share. On every scheduling decision, the VM load (ready time) is updated. The proposed approach updates VM's computation share and load after a predefined interval. DRALBA has evaluated and compared against their counterparts in terms of total execution time, task response time, average resource utilization ratio, and throughput. However, the proposed approach does not support task deadlines and the task rejection ratio has not been used as a scheduling objective. Moreover, overall performance of the Cloud not computed.

In summary, the literature review revealed that the static and batch dynamic scheduling techniques suffer from issues like poor resource utilization, load-imbalance, and unable to meet the deadlines of newly arrived deadline-based tasks. Moreover, the batch dynamic scheduling heuristics suffers from the issues like new batch formation-based delay (resulting in higher response time) and under-utilization of resources during the formation of a new batch. There are several dynamic tasks scheduling approaches which are batch-based; however, most do not support the deadline-based tasks. There are few approaches which support deadline-based

tasks such as DLBA [35] and DC-BLBA [34]; however, these approaches have high task-rejection ratio and imbalance mapping of jobs.

2.3 Meta-heuristics based Cloud Task Schedulers

Meta-heuristics based task scheduling algorithms are problem independent techniques and can be applied to various domains. These schedulers are considered more suitable for optimizing multi-objective optimization problems in cloud computing. In the last, two decades, meta-heuristic based task scheduling algorithms have become very popular for solving NP-hard problems.

Mousavi et al. [21] have proposed meta-heuristics based task scheduling algorithm that combines *Grey Wolves Optimization* (GWO)and *Teaching Learning Based Optimization* (TLBO) algorithms. TLBO is used to search the problem space, identifying optimal parameters, and provide settings for meeting the problem objective. GWO provided better exploration and exploitation while the TLBO helps in avoiding trapping into the local optima. The objective function of the proposed optimization technique is the maximization of throughput. However, the proposed approach not considering ARUR, task deadline, and task monetary cost.

In [41], Torabi et al. have proposed an approach that uses a combination of *Chicken Swarm Optimization* (CSO) and *Improved Raven Roosting Optimization* (IRRO) algorithms. This approach uses strengths of both IRRO and CSO to provide balance in the local and global search which results in solving premature convergence, reduces the response time, execution time, and improves throughput. By using a hybrid algorithm (IRRO-CSO), a framework named IRRO-CSO *Dynamic Scheduling Framework* (ICDSF) has been proposed for dynamic scheduling of independent tasks in a cloud environment. The results show that the improvement in the execution time of the proposed approach and the existing heuristics is very small and the improvement in the response time is very high. However, this method does not consider resource utilization, monetary cost and tasks deadlines.

Wang et al. [80] have presented an enhanced form of Genetic Algorithm (GA)

TABLE 2.1: Summary of the heuristic based cloud task schedulers

Aproach	Year	Tool	Strengths	Weaknesses
OLB [64, 65]	2009	Custom event-based	Minimal implementation and scheduling overhead	high makespan and under-resource utilization
RR [66]	2008	Cloudsim	Minimal implementation and scheduling overhead	Not resource-aware, load imbalance and high makespan
MCT [67]	2012	Cloudsim	Improved Resource Utilization and makespan than RS and RR	Overload faster VMs, not support deadline based tasks
Max-Min [68]	2012	Cloudsim	Map largest jobs on the fastest VMs, Favors larger tasks	Execution delay for smaller tasks[1], task deadline not supported
Dy-MaxMin [33]	2014	Cloudsim	Compute VMs status after an interval, Real-time load balancing	Under-resource utilization[1][23], Task deadline not considered
TM-eFCFS [38]	2017	Cloudsim	Task migration can be very effective in load balancing	No task migration for slower idle machines, load imbalance and reduced resource utilization [79][23]
DLBA [35]	2017	Cloudsim	Threshold-based mapping, prevent over-provisioning of VMs	threshold partial usage, Lower Resource utilization, load imbalance, high task rejection and makespan
RALBA [1]	2018	Cloudsim	Balanced distribution of load among a predefined number of VMs	High makespan and under-resource utilization, task deadline not supported
TSSLB [23]	2018	Cloudsim	Reduced execution time by creating VM in advance	Dependent on historical information (previous scheduling) [19][20]
Repro-Active [19]	2019	Cloudsim	Independence of the historical information	Poor resource utilization, load imbalance [1], batch based
MODE [76]	2020	Cloudsim	High scalability and reduced task response time	Poor-resource utilization, higher makespan and monetary cost
DRALBA [78]	2021	Cloudsim	Reduced makespan and high throughput	Not support task deadline, and task rejection

named as Look-Ahead Genetic Algorithm (LAGA)for cloud and Grid computing. LAGA is designed for large scale distributed systems and is reputed for reliability and runtime-based execution of tasks in cloud and grid environment. In every generation, the proposed technique computes the order of tasks based on resource completion time. LAGA selects the resource which has a minimum failure rate in the mutation step. The objective function of LAGA comprises the failure rate of tasks and reliability. However, this scheduling technique does not support makespan, ARUR, task response time, and monetary cost.

In [81], Zhang et al. have proposed a particle swarm optimization based tasks scheduling scheme and compared it with a Genetic Algorithm (GA) using the same simulation environment. Minimization of task execution time and maximization of resource utilization are the key scheduling objective of the proposed approach. The experimental results reveal that the PSO provides a better quality solution with lesser time than that of GA in the majority of the test cases. Based on their observations, the Zhang et al. have concluded that the PSO algorithm outperforms GA for large-scale optimization problems. However, task deadlines, task response time, and monetary cost are not supported.

In [82], Khalili et al. have experimentally evaluated the five most prominent Inertia weight strategies using Particle Swarm Optimization (IW-PSO) in terms of makespan. These strategies include *Simple Random Inertia Weight* (SRIW), *Chaotic Random Inertia Weight* (CRIW), *Chaotic Descending Inertia Weight* (CDIW), and (SRIW), *Linear Descending Inertia Weight* (LDIW), *Chaotic Random Inertia Weight* (CRIW), *Chaotic Descending Inertia Weight* (CDIW), and *Adaptive Inertia Weight* (AIW). Inertia weight strategy is used to provide a balance between global and local search. Khalili et al. have concluded that the LDIW inertia weight strategy performs better than other state-of-the-art inertia weighted strategies. However, parameters like ARUR, task response time, task deadline, and monetary cost are not considered.

Alkayal et al. [24] have investigated particle swarm optimization-based algorithms for task scheduling in cloud computing. In this paper, the authors have divided the PSO based literature work based on the number of objectives (i.e., single objective

or multiple objectives) used for optimization. The literature shows that majority of the authors have used standard PSO or modified PSO algorithms. Alkayal et al. have concluded that to balance the workload and improve QoS parameters like task execution time, cloud throughput, task response time, Utilization of cloud resources, and task execution cost need more focus and improvement.

Kumar and Sharma [83] have proposed a particle swarm optimization-based model for resource allocation in the cloud environment. The proposed approach is termed PSO-COGENT and the ultimate objective of the proposed scheduler is to efficiently process the user jobs with minimum cost and energy consumption. The PSO-COGENT is a multi-objective based task scheduling scheme that optimizes multiple parameters like makespan, throughput, task rejection ratio, energy consumption, and task execution cost. The PSO-COGENT algorithm is implemented using the cloudsim simulator and evaluated using random independent tasks. However, the proposed approach does not consider the ARUR and task response time.

Kumar et al. [7] has proposed a Particle Swarm Optimization (PSO) based novel framework for tasks processing and resource mapping (RTPF-PSO) in cloud environment. The aim of this research is to find optimal mapping of users tasks to the requested resources, minimize resource consumption and improve a number of Quality of Service (QoS) parameters. These parameters include reducing total execution time and cost, improving tasks acceptance ratio by meeting user's budget deadline, and enhancing system throughput. The framework comprises of job request handler that accepts user request using user interface and forwards them to the controller node. The controller node is the fundamental component of processing framework that communicate with other component and handle incoming and outgoing requests.

In summary, the literature review revealed that most of the meta-heuristic based task scheduling algorithms are single objective or bi-objective and the majority of these algorithms consider non-conflicting parameters like makespan, throughput, and response time, etc. for performance evaluations. Moreover, only few of

meta-heuristic based algorithms support task deadline and tasks execution cost.

TABLE 2.2: Summary of meta-heuristic-based task schedulers

Aproach	Year	Tool	Strengths	Weaknesses
LAGA [80]	2011	Gridsim	Improved reliability and lower task failure rate	ARUR, makespan,task response times, and monetary cost not supported
PSO-GA [81]	2008	Gridsim	In-depth comparison of PSO and GA	ARUR, task response time, task deadline, and monetary cost are not considered
IW-PSO [82]	2015	Cloudsim	Experimental evaluation of prominent Inertia weights for PSO	ARUR, task response time, task deadline, and monetary cost are not considered
PSO [24]	2017	N/A	Scheduling objectives based categorization	Experimental evaluation not performed
GWO-TLBO [21]	2017	MATLAB, Cloudsim	Avoid trapping into local optima and higher throughput	ARUR, makespan,task deadlines, and monetary cost not supported
ICDSF [41]	2018	MATLAB	Improved task response time and throughput	Not supporting task deadlines, ARUR, and monetary cost
RTPF-PSO [7]	2019	Cloudsim	Reduced execution time and optimized task-resource mapping	task response time, task rejection, and ARUR not supported

2.4 Gap Analysis

The tasks scheduling heuristics like RS [1, 64], Round Robin [66] and MCT [1, 67] are unable to distribute the incoming workload on heterogeneous resources in a balanced way [1, 19, 38, 59]. It is because these heuristics are not resource-aware. Moreover, these algorithms are unable to execute the newly arrived tasks with shorter deadlines within their deadlines. It is because the newly arrived tasks will have to wait until the execution of already assigned tasks. RALBA [1] is a batch dynamic resource-aware scheduling heuristic. However, RALBA cannot adjust the newly arrived tasks with shorter deadlines. The newly arrived deadline-based tasks will wait till completion of already mapped tasks. As compared to the static and batch dynamic scheduling algorithms, dynamic schedulers are much more flexible. Dynamic schedulers have the ability to assess, estimate, and update the VMs and tasks status during the execution. However, a number of the existing dynamic approaches [19, 20, 33] still use an interval-based batch of input tasks arrived in a particular time. Moreover, dynamic task scheduling heuristics also provide flexibility like estimating the real-time workload of VMs, task migration, and new VMs creation. However, majority of the dynamic task scheduling heuristics are still unable to improve utilization of cloud-resources and load balance [19–21, 35], and fails to minimize makespan [33] and task rejection ratio [33, 34].
Most of the dynamic scheduling heuristics [19, 21, 32, 33, 41] do not consider the tasks deadline. The tasks with the shorter deadline may not complete in the due time which may result in user dissatisfaction. Moreover, meta-heuristics-based tasks scheduling algorithms provide a generalized solution that is problem/domain independent. These approaches can provide an optimal and compromised solution for conflicting scheduling objectives. However, these approaches incurred high computational complexity and are difficult to implement. Most of the existing static, batch dynamic, and dynamic heuristics and meta-heuristics based task scheduling approaches perform an evaluation based on either a single parameter or multiple independent parameters. However, these improving one parameter may affect the performance of the other parameter. The literature study conducted in this research reveals that none of the task scheduling techniques has evaluated their scheduling scheme based combined effect of multiple parameters like

makespan, ARUR, task response time, task rejection ratio, and execution cost of task scheduling approaches.

To address these issues and achieve maximum load balance, meet the deadlines of newly arrived tasks with shorter deadlines, and compute the overall performance of the cloud there is a need to have such a scheduling technique which should be resource, deadline aware, and also reduce total execution cost of executing user tasks.

2.5 Summary of the Chapter

In static scheduling techniques, tasks are assigned to the VMs statically which cannot be changed at runtime. These approaches are unable to dynamically handle any delay in execution of a particular task. Most of these heuristics suffer from issues like poor resource utilization, load imbalance [1, 23, 33, 38, 59] and unable to meet deadline of newly arrived deadline-based tasks.

Batch dynamic scheduling [1, 37] heuristics provide dynamism at the batch level. However, these approaches have issues like new batch formation-based delay in response time, inter-batch under-utilization of resources. Moreover, tasks with shorter deadlines may suffer when VMs are already overloaded.

A number of the dynamic approaches like PSSLB [19], PSSELB [19], and HBLBA [20], Max-Min [33] are batch-based which suffers from issues like: 1) Batch formation-based response time delay and 2) Inter-batch-based under-utilization of resources. Most of these approaches not supporting the deadlines based tasks. Approaches like DLBA [35] and DC-DLBA [34] supports tasks deadline. However, tasks rejection is high and the load is still not balanced.

Heuristic-based task scheduling algorithms are fast, simple to implement, and incurred lower computational complexity. These algorithms are more suitable for single objective or multiple non-conflicting objectives. However, heuristic-based task scheduling approaches are not more efficient for solving conflicting parameters-based multi-objective optimization task scheduling problems. Most of the meta-heuristic-based task scheduling algorithms are single objective or bi-

objective and the majority of these algorithms consider non-conflicting parameters like makespan, throughput, and response time, among others for performance evaluations. Moreover, only a few meta-heuristic-based algorithms support task deadline and task execution cost [7, 76, 83]. The subsequent Chapter describes the proposed methodology, based on the discussion presented in this Chapter.

Chapter 3

RADL: A Resource-Aware Dynamic Load-balancer for Deadline Constrained Cloud Tasks

3.1 Introduction

The fast development of processing and storage technologies owing to the Internet has made the computing resources powerful, easily available, and economical to use [8]. This rapid growth in technology has resulted in the birth of cloud computing [8] that enables end users to share resources like CPU and storage for a particular time based on their needs. Cloud computing model comprises of two key actors [6, 7] (as shown in Figure 1.2). 1) *Cloud Service Provider* (CSP): deploy the resources like storage, processors, network etc., 2) Service users/clients: hire the services provided by the service providers for their temporal needs. Service providers use internet based-interfaces to make their services accessible to the service users.

To deliver better services, there is a need to utilize the available cloud resources in a balanced way [19]. This leads to the importance of selecting the preemi-

nent scheduling algorithm for allocating cloud resources. There are two levels of mapping involved in the cloud. 1) VM to host mapping that maps VMs on the underlying host machines and 2) Task to VM mapping which allocates tasks to the respective VMs. This focus of this research is on tasks to VMs mapping, where the number and computation power of VMs are pre-determined. Load balanced [1, 20] task scheduling [22] plays an important role to enable optimal use of the cloud resources.

Cloud task scheduling techniques can be based on heuristics or meta-heuristics algorithms. Heuristic-based task scheduling algorithms provide a near-optimal solution efficiently. These algorithms have a simple implementation with lower complexity and scheduling overhead [7]. However, heuristics-based scheduling algorithms are not suitable for multi-objective based scheduling with conflicting parameters like task execution time and cost. Meta-heuristic based scheduling algorithms can provide an optimal solution for multi-objective based optimization problems with conflicting parameters. In this chapter, a novel heuristics-based dynamic and deadline-aware tasks scheduling algorithm has been proposed. Heuristics-based task scheduling algorithms can be static [28, 29], dynamic [19, 20, 33] and batch dynamic [1, 10, 37].

In this work, we argue that task scheduling algorithms should be resource-aware to achieve better performance. Resource-aware [1, 61] load balancing process includes resource discovery, monitoring the current loads on each resource, and assessing workload to be assigned to a resource. To overcome the load imbalance issue, the tasks should be scheduled on the most suitable VMs considering the resource capacities. Therefore, this research proposes a dynamic resource-aware scheduling algorithm named, *Resource Aware Dynamic Load-balancer* for Deadline Constrained Tasks (RADL) to mitigate the load imbalance issue and to support deadline constraints. RADL scheduling heuristic distributes the incoming workload of independent, non-preemptive, and compute-intensive tasks in a balanced manner. The RADL approach dynamically updates two metrics, i.e., VM-load and ready-time. The prime objective of the RADL scheduler is to increase resource utilization, meet deadlines of newly arrived deadlines-based tasks, and reduce makespan. The proposed approach consists of two schedulers: 1) RADL-

Scheduler: assigns incoming tasks to the pre-defined number of VMs based on the minimum completion time of the tasks. Moreover, the *RADL-Scheduler* monitors, updates, the task and VM status at run-time. Task status table contain tasks related information like Task Id, VM Id, task arrival time, task completion time, and last update time. Similarly, VM status table contain information like VM Id, Number of tasks assigned, total remaining time, VM life cycle status, last update time. 2) *ShifterScheduler* (S-Scheduler), this sub-scheduler places the incoming tasks with a shorter deadline in a suitable position of the task queue of a VM (which provides minimum completion time for the execution of that job). This mechanism helps to execute the newly arrived tasks (with shorter deadlines) in a timely manner. The proposed scheduling mechanism accommodates the newly arrived deadline-based tasks without migrating (the executing jobs) with no additional VM requirement. The employed mechanism (of not migrating executing jobs and creating new VMs) results in the reduced scheduling overhead. In summary, the major contributions of this research are:

- In-depth critical analysis of static and dynamic state-of-the-art scheduling heuristics to identify the strengths and limitations of these approaches.
- A novel dynamic load-balancing scheduler for independent, non-preemptive, and compute-intensive cloud tasks that produce improved resource utilization, reduced task rejection, lower makespan, and minimized response time.
- Empirical investigation and performance evaluation of the proposed scheduling approach against state-of-the-art scheduling heuristics.

Rest of the chapter is organized as follows. Section 3.2 presents RADL system overview and background, RADL system architecture, RADL system model, RADL algorithm, complexity analysis, and scheduling overhead. The experimental setup, workload generation, and simulation results in Section 3.3. Section 3.4 presents RADL Results and discussion. Section 3.5 concludes the chapter and presents potential future directions.

3.2 Proposed Load-balancing Algorithm

This section presents a complete overview of the proposed scheduling technique RADL's system architecture, performance model used, detail description of the algorithm, time complexity, and scheduling overhead analysis.

3.2.1 RADL System Overview and Background

From the literature review, it has been observed that most of the existing scheduling algorithms suffer from issues like poor resource utilization, load imbalance, high makespan, and task rejection ratio. Moreover, these algorithms are also unable to map the newly arrived tasks with shorter deadlines within their deadlines. RADL is a resource and deadline-aware load-balanced scheduling technique and composed of two schedulers: i.e., RADL-Scheduler and S-Scheduler. RADL scheduling scheme picks a single task from the input queue or buffer irrespective of the task arrival rate, tries to map that task on a suitable VM that completes their execution within minimum time and can meet their deadline. According to the assumptions of the proposed technique, the RADL scheduler schedule incoming tasks on pre-defined number of VMs without creating new VMs at run time. Moreover, this is the responsibility of the cloud service provider to manage and decide the number of VMs according to the expected arrival rate of the tasks.
A renowned Cloudsim [11] simulator is used for the simulation of the proposed scheduling algorithms. In cloudsim simulator cloudlet is used as a synonym for the task. Datacenter entity of cloudsim is used to simulate infrastructure level services related to the cloud. Datacenter manages hosts machines and the power of Datacenter is represented by the computation power of host machines and storage servers on that Datacenter. One or more VMs are assigned to each host based on the *Cloud Service Provider* (CSP) defined VMs allocation policy. RADL scheduling scheme assigns tasks to VMs on just-in-time based in order to improve resource utilization reduce task rejection. Just-in-time means that the proposed technique map the tasks when it received without any wait for the formation of batch and also known as online or real time scheduling which helps in preventing wastage of

time and unnecessary delay [84]. Figure 3.1 shows the basic architecture of the Cloud.

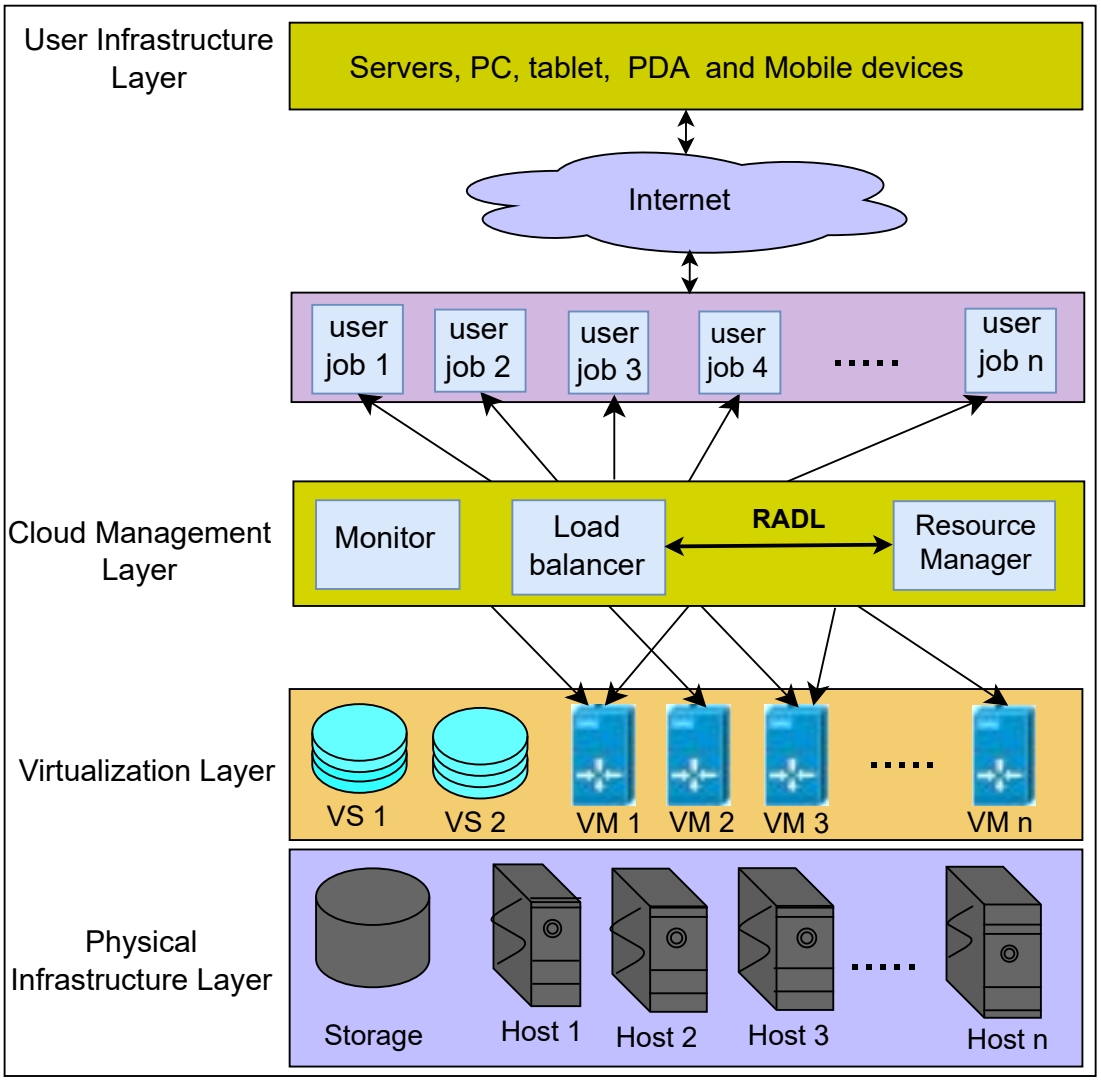


FIGURE 3.1: Basic Architecture of Cloud

The user Infrastructure layer represents the Cloud interface used by the users who communicate with the Cloud through Internet. The cloud management layer comprises of several submodules like resource manager that determines the user requirements in terms of computation resources, provide information of currently available resources, the monitoring module allows a system administrator to initiate and monitor activities of each layer, load-balancer manages the distribution of workload for execution among available virtual resources. RADL scheduling

scheme is proposed at the Cloud management layer to enhance load balancing and improve utilization of Cloud resources. The virtualization layer represents virtual instances of the Cloud resources like virtual storage and virtual machines. The physical infrastructure layer represents the physical infrastructure of the Cloud.

3.2.2 RADL System Architecture

RADL is a dynamic (just-in time based) cloud scheduling approach that maps incoming task in a load balanced manner. The system architecture of the proposed model is shown in Figure 3.2. Some of the notations used in RADL scheduling technique are given in Table 3.1. RADL scheduling technique comprises of two algorithms, RADL-Scheduler and S-Scheduler which consist of following major steps:

1. RADL-Scheduler receives a task with their size sz_i in Million Instructions (MIs) along with their deadline dT_i (in MilliSecond (MS)). A list of VM (VMS) with their computation capability is provided as an input parameter (shown in Figure 3.2, Step 1). VMS represents the actual size of cloud Datacenter.

2. The Completion Time (CT_{ij}) of the received task is calculated for all VMs and store in a storage structure (i.e., hashMap) (presented in Figure 3.2, Step 2). CT_{ij} is the sum of the expected execution time of task T_i on VM_j and current load on the VM_j as shown in Eq. (3.2).

TABLE 3.1: Notations and definitions used in RADL technique.

Notations	**Description**
CT_c	Completion time of candidate task to be shifted
CT_{ij}	Task completion time on VM_j
dT_c	Deadline of candidate task
dT_i	Task deadline in Millisecond (MS)
LR-Scheduler	Sub-scheduler that is invoked to identify SPQ for task T_i
$minCT_{ij}$	Least Completion Time of task T_i on VM_j

MindT_i	Task in the VM task queue having shortest deadline
newCT_c	New completion time of candidate task
pList	List of Parameters
sz_i	Task size in MI
tList	List of tasks

3. This step identifies VM_j which gives expected minimum completion time (minCT_{ij}) for T_i. minCT_{ij} is compared with the deadline dT_i of the task T_i (shown in Figure 3.2, Step 3).

4. This step will be executed based on the true value of condition at step 3. If minCT_{ij} of task T_i is less than the deadline of that task, task T_i is mapped to the VM_j that execute them in minimum time (Figure 3.2, Step 4).

5. When task T_i is mapped to the VM_j, task status and VM status tables are updated (presented in Figure 3.2, Step 5). 6. In case, minCT_{ij} of task T_i is greater than their deadline dT_i, RADL-Scheduler calls S-Scheduler. S-Scheduler computes a suitable position for task T_i in the task queue of VM_j. Task T_i, VM_j, minCT_{ij}, and deadline dT_i are provided as input to the S-Scheduler (shown in Figure 3.2, Step 6). Based on the output (true or false) value of the S-Scheduler, task T_i will be assigned to the VM_j or will select another VM from VMS which gives next minCT_{ij} value for task T_i.

7. S-Scheduler will check the tasks queue of VM_j to identify the task T_c, which has minimum deadline greater than the deadline of task T_i (shown in Figure 3.2, Step 7). If such a task (i.e., candidate task T_c) is found, then new completion time of task T_c (i.e., newCT_c) and new completion time for task T_i (i.e., CT_i), will be computed. The newCT_c and CT_i) will be compared with dT_c and dT_i respectively. In case any of these conditions become false then step 7 will be repeated until all the task queue of VM_j is scanned.

8. In case, both conditions at step 7 are true i.e., task T_i can be adjusted in the task queue of VM_j, then the positions of task T_i and T_c will be updated (shown in Figure 3.2, Step 8).

9. In case, both conditions at step 7 become true and tasks positions are updated as represented in step 8. Task T_i will be mapped to VM_j. Task status and VM

status tables are updated (shown in Figure 3.2, Step 9),.

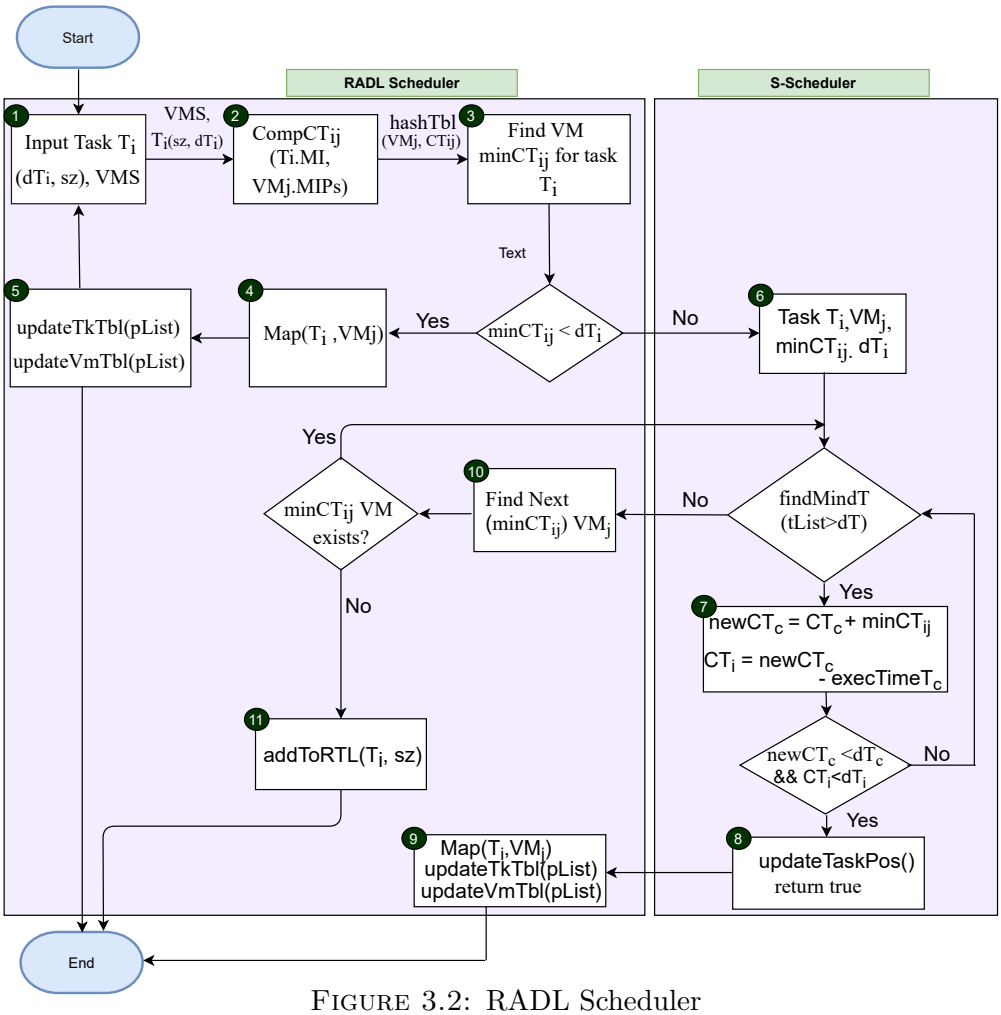


FIGURE 3.2: RADL Scheduler

10. If condition at step 6 becomes false, a new VM that gives next minimum completion time will be identified (shown in Figure 3.2, Step 10). In case, next VM with minimum completion time is found, then steps 7 and 8 will be repeated.
11. In case, none of the VMs is able to execute task T_i within their deadlines, task T_i will be added to the rejected task list and the next task will be served. The process of the task to VM mapping will continue until no task is available for mapping (shown in Figure 3.2, Step 11).

3.2.3 RADL System Model

The basic notations, definitions, and terminologies that have been used in the mathematical expressions of the proposed approach shown in Abbreviations and

Symbols section. To delineate the performance of RADL tasks scheduling technique, a unified cloud system model is formed [1]. A cloud Datacenter comprises on a number of VMs (VMS, as shown in Eq. (3.1), and a VM can be represented as VM_j.

$$\mathrm{VMS} = \mathrm{VM}_1, \mathrm{VM}_2, \mathrm{VM}_3, \ldots \mathrm{VM}_m \tag{3.1}$$

, where m is the total number VMs on the cloud Datacenter ($1 \leq \mathrm{j} \leq \mathrm{m}$). Task T_i represents the task to be mapped on VM_j that executes them in minimum time. CT_{ij} is the expected completion time (depict in Eq. (3.2)) of task T_i on VM_j and mathematically expressed as:

$$\mathrm{CT}_{ij} = \mathrm{vmExecT}_{ij} + \mathrm{vmRT}_j \tag{3.2}$$

Completion time of task T_i on VM_j is the sum of the execution time of task T_i and the current load on that VM. $vmExecT_{ij}$ is the execution time of task T_i on VM_j and $vmRT_j$ is ready time (current load) of VM_j. $vmExecT_{ij}$ is computed using Eq. (3.3), where the task size (in MI) is divided by the computation power of VM_j (in MIPS).

$$\mathrm{vmExecT}_{ij} = \left(\frac{\text{Task size}(\mathrm{T}_i)}{\mathrm{VM}_j.\mathrm{MIPS}}\right) \tag{3.3}$$

, where size of the task is represented as million instructions (MI) and MIPS represens the computation power of VM_j. The Eq. (3.4), represents $vmRT_j$ which is the ready time (already assigned workload) of VM_j and is computed as:

$$\mathrm{vmRT}_j = \sum_{i=1}^{k} \mathrm{vmExecT}_{ij} \tag{3.4}$$

, where k is the total number of tasks assigned to VM_j. $minCT_{ij}$ (shown in Eq. (3.5)) is the completion time of task T_i on VM_j that executes them in least amount of time.

$$\mathrm{minCT}_{ij} = \min\ (\mathrm{CT}_{ij}) \qquad \forall \mathrm{j} \in 1,2,3,\ldots\mathrm{m} \tag{3.5}$$

If deadline of task T_i is less than the completion time of T_i on VM_j which executes them in least amount of time, then a suitable position for T_i will be identified in the task queue of already mapped tasks to VM_j. SPQ represents Suitable Position in VM task Queue and will return true value or false value (depicted in Eq. (3.6))

and represented mathematically as:

$$\text{SPQ} = \begin{cases} 1 & \text{SPQ of } \text{T}_i \text{ identified} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

If the return value is 1 (true) then the task will be assigned to VM_j, VM and tasks status will be updated.

$$\text{VM\_CT}_j = \sum_{i=0}^{k} \left(\frac{\text{Size of } \text{T}_i \text{ (in MI)}}{\text{VM}_j\text{.MIPS}}\right) \tag{3.7}$$

VM_CT_j represents completion time of VM_j which is the sum of completion time of all tasks assigned to VM_j and computed using Eq. (3.7) [1]), where k shows the number of tasks assigned to VM_j.

Response time of task T_i is the difference of task arrival time and task execution start time, and is computed using Eq. (3.8).

$$\text{RspTime}_i = (\text{execSTime}_i \text{ - } \text{arrTime}_i) \tag{3.8}$$

$RspTime_i$ is the response time of task T_i, $execSTime_i$ and $arrTime_i$ are the execution start time and arrival time of task T_i. Moreover, average response time is computed by dividing sum response time of all mapped tasks on the total number of tasks.

3.2.4 RADL Performance Model

The performance of the proposed approach RADL is evaluated against the state-of-the-art approaches in terms of ARUR, percentage of task rejection, makespan and response time.

Makespan is the maximum time taken by a resource to finish the assigned tasks among all other resources of cloud datacenter i.e., the completion time of a resource (VM) which finish execution of tasks most recently. Mathematical representation of makespan is shown in Eq. (3.9) [1, 23, 34, 59, 62].

$$\text{Makespan} = \max\ (\text{VM\_CT}_j) \qquad \forall \text{j} \in 1,2,3,...\text{m} \tag{3.9}$$

, where m is the total number of VMs in VMS. ARUR represents the overall utilization of cloud resources (as depicted in Eq. (3.10) [1, 19]). ARUR is mathematically expressed as:

$$\text{ARUR} = (\sum_{j=1}^{m}(\text{VM\_CT}_j)) \times \frac{1}{\text{m} \times \text{Makespan}} \tag{3.10}$$

ARUR values lie among 0 and 1, where a higher value shows higher utilization of cloud resources and vice versa. Percentage of task rejection (%ageRT as shown in Eq. (3.11)) [83]) is the ratio between the number of rejected tasks and the total number of tasks and is computed as:

$$\text{RT}_{\%}age = (\frac{\text{RTL size} \times 100}{\text{N}}) \tag{3.11}$$

, where RTList is the rejected tasks list and size of RTList shows the number of rejected tasks.

$$\text{AvgRspT} = \frac{1}{m}\sum_{j=1}^{m}(\frac{1}{k}\sum_{i=1}^{k}\text{RspTime}_{ij}) \tag{3.12}$$

Average Response Time (AvgRspT) (as shown in Eq. (3.12) represent average response time of scheduled tasks [85]. In Eq. (3.12), m represent the total number of VMs and k represent the total number of tasks assigned to each VM. To compute average response time of all the tasks, average response time of each VM is identified by dividing the sum of the response time of all k tasks by total number of assigned tasks (k). Then the sum of the average response time of each VM is divided by the total number of VMs (m).

3.2.5 RADL Algorithms

This section discusses the proposed scheduling technique RADL based on two algorithms RADL-Scheduler and S-Scheduler. The RADL-Scheduler maps tasks based on minimum completion time and updates VM and Task status tables. The proposed S-Scheduler finds a suitable position in the task queue of VM which executes that task in minimum time.

3.2.5.1 RADL Scheduler

To perform mapping of tasks on VMs, RADL-Scheduler (Algorithm 1) receives task T_i with their size in the unit of Million Instructions (MIs), task deadline (dT_i) in Milli Seconds (MS), and a list of VMs (VMS) with their computation power as input parameters. Outputs of Algorithm 1 include mapped tasks on VMs and Rejected Task List (RTL). Lines 1-6 of Algorithm 1 present necessary initialization of the RADL algorithm. The while loop (lines 7-37, Algorithm 1) will continue until there is a task T_i to be scheduled.
For loop (lines 8-11, Algorithm 1) compute the completion time of task T_i on every VM in the VMS using Eq. (3.2), (3.3), and (3.4) and store task completion time of each VM in a hashMap. VM with minimum completion time is identified (using Eq. (3.5)) (lines 12, Algorithm 1). The method at (lines 12, Algorithm 1) takes hashMap as an argument that contains completion times of all VMs for task T_i. The completion time for task T_i is obtained (lines 13, Algorithm 1) and compared with the task deadline (line 14, Algorithm 1). On the true side of the decision (line 14, Algorithm 1), the task is mapped to the VM_j and placed at the end of the task queue of VM_j, task status table, and VM status table is updated (lines 15-17, Algorithm 1). If the decision (line 14, Algorithm 1) is false then the else part (lines 18-36, Algorithm 1) is executed.

The S-Scheduler (line 20, Algorithm 1) is invoked which finds a possible suitable position for task T_i in the task-queue of VM_j. In the task queue of VM_j, the S-Scheduler identifies task with minimum deadline with the constraint of greater than the deadline of new unscheduled task (i.e., T_i) and if such a task is found in the task queue of VM_j then it returns a true value. In case, S-Scheduler returns a true value (line 21, Algorithm 1), task T_i is mapped to VM_j, the tasks and VMs status tables are updated (lines 22-24, Algorithm 1). On the other hand, if the condition (line 21, Algorithm 1) becomes false then the else part (lines 25 to 30, Algorithm 1) of if statement is executed where next VM within $minCT_{ij}$ the VM list is selected (line 26, Algorithm 1). The method at (line 26, Algorithm 1) finds next VM that executes task T_i in a minimum time for the remaining VMs. If VM with next $minCT_{ij}$ is found, their $minCT_{ij}$ is obtained (line 28, Algorithm 1) and

repeat-until loop executes again.

Algorithm 1: RADL Scheduler

Input : VM list with their computation power, Task T_i with size "sz_i" and deadline "dT_i"

Output: Map (T_i, VM_j) i.e., Mapping of Task T_i to VM_j

```
1  CT_ij = 0
2  minCT_ij = 0
3  Vm VM = null
4  RTL = null
5  findSPQ = false
6  VmId = 0
7  while T_i.exists() do
8      for j = 1 to m do
9          CT_ij = computeCT_ij (T_i.MI, VM_j.MIPs)
10         hashMap.add(CT_ij)
11     end
12     VM_j = findVmWithMCT(hashMap)
13     minCT_ij = VM_j.getKeyValue()
14     if minCT_ij < dT_i then
15         Map.add(T_i, VM_j)
16         updateTkTbl(pList)
17         updateVmTbl(pList)
18     else
19         repeat
20             findSPQ = S-Scheduler (T_i, VM_j, dT_i, minCT_ij)
21             if findSPQ == true then
22                 Map.add(T_i, VM_j) updateTkTbl(pList)
23                 updateVmTbl(pList)
24             else
25                 VM_j = findNVmWithMCT(hashMap)
26                 if VM_j != null then
27
28                     minCT_ij = VM_j.getKeyValue()
29                 end
30             end
31         until findSPQ == true or VM_j == null;
32         if findSPQ == false then
33
34             RTL.add (T_i)
35         end
36     end
37 end
```

Steps 2 and 3 shown in Figure 3.2.

Steps 4 and 5 shown in Figure 3.2.

call to the S-Scheduler

Step 9 shown in Figure 3.2.

Represent step 10 of Figure 3.2.

Represent step 11 of Figure 3.2.

3.2.5.2 S-Scheduler

The repeat-until loop is repeated until task T_i and is mapped to a suitable VM or complete list of VMs is checked for adjusting the task T_i to execute it within their deadline. In case, none of the VMs adjust the task T_i within their deadline, then the control moves out of loop (line 19-31), then the task T_i is added to the Rejected Task List (RTL) (line 34, Algorithm 1).
The *S-Scheduler*(Algorithm 2) is invoked by RADL scheduler (Algorithm 1) when the $minCT_{ij}$ of a task T_i is greater than the task deadline dT_i. The input parameters for *S-Scheduler* comprise of task T_i with the deadline (dT_i), VM_j, and $minCT_{ij}$. The *S-Scheduler* return true(1) value or false (0) value as an output. The necessary initialization of Algorithm 2 is represented in lines 1-5.
The candidate task (T_c) which has the lowest deadline value in the task queue of VM_j that is greater than dT_i is identified (line 6, Algorithm 2). In case the candidate task is not found i.e., the condition becomes true (line 7, Algorithm 2), a false value is returned, and control move out of if condition (lines 7-8, Algorithm 2). On the false side of the if statement, the else block (line 9-37, Algorithm 2) will be executed. Execution time of T_i is computed (line 11, Algorithm 2). The new completion time of candidate task T_c (i.e., $newCT_c$) is identified by adding their completion time and execution time of task T_i (line 12, Algorithm 2). Similarly, execution time of candidate task T_c (i.e., $execTimeT_c$) and new completion time ($newCT_i$) of task T_i is obtained at (line 13-14, Algorithm 2). In case, the conditions at line 15 (algorithm 2) are false, the next task with a minimum deadline greater than dT_i is identified and the repeat-until loop (lines 10-20, Algorithm 2) will be executed again. The repeat-until loop is repeated until an appropriate location in the tasks queue of VM_j is determined or the complete queue is scanned and the appropriate location is not found. If the conditions at (line 15 of Algorithm 2) are true, the control moves out of the repeat-until loop and next statement (line 21 of Algorithm 2) after the loop is executed.

The position of the candidate task ($PosT_c$) is obtained from the task execution Map (TEMap) at (line 21 of Algorithm 2). The deadline dT_k, completion time CT_k), and updated completion ($newCT_k$) time of next task to T_c) which is termed

as T_k) the task is obtained (lines 23-25 of Algorithm 2).

Algorithm 2: S-Scheduler

Input : T_i with deadline dT_i, VM_j and $minCT_{ij}$

Output: true or false

```
1  Tc =null
2  newCTc = 0
3  execTimeTi =0
4  CTi = 0                                              }
5  execTimeTc = 0                                       }
6  Tc = findMindTi (tList) > dTi                        }  Represent step 6
7  if Tc == null then                                   }  of Figure 3.2.
8      return false                                     }
9  else
10     repeat                                           }
11         execTimeTi = Ti.MI/VMj.MIPs                  }
12         newCTc = CTc + execTimeTi                    }
13         execTimeTc = TETMap(Tc)                      }
14         newCTi = newCTc - execTimeTc                 }  Represent step 7
15         if (newCTc < dTc && CTi < dTi) then          }  of Figure 3.2.
16             break:                                   }
17         else                                         }
18             Tc = findNextMindTi (tList) > dTi        }
19         end                                          }
20     until tList.hasNext();
21     PosTc = getPosTc(TEMap)
22     for k = PosTc to vmQ.length do
23         dTk = getDeadline(Tk)
24         CTk = getCTk (TEMap)
25         newCTk = CTk + minCTij
26         if (newCTk > dTk) then
27             flag = 0
28             break:
29         end
30     end
31     if (flag == 1) then
32         updateTaskPos()                              }
33         return true                                  }  Represent step 8
34     else                                             }  of Figure 3.2.
35         return false                                 }
36     end
37 end
```

If the condition at line 26 (Algorithm 2) becomes true then the flag value is set to zero and the control moves out of the For loop. if condition at (line 26 of

Algorithm 2) becomes false then the For loop is executed again.
The For loop is executed until the tasks in the queue are checked or the condition at line 26 (Algorithm 2) becomes true. If the condition at line 31 (Algorithm 2) becomes true then the position of the task is updated and a true value is returned (lines 32-33, Algorithm 2). In case, condition at (line 31, Algorithm 2) becomes false, a false value is returned (line 35, Algorithm 2) and control moves to the algorithm 1.

3.2.6 Complexity and Overhead Analysis

To analyze the computational complexity (in terms of time) of the proposed scheduling technique, we consider that N represents the total number of tasks and m represents the total number of VMs in a cloud Datacenter. For a real cloud Datacenter, the following assertion is maintained: N>>m. RADL algorithm finds VM with *Minimum Completion Time* (MCT) for a task after performing m number of comparisons. For N number of tasks, the total number of comparisons has become N(m). In case, the deadline of the task is greater than the minimum completion time of that task then S-Scheduler is invoked which tries to adjust the current task in the task queue of VM with MCT. In the worst case, the complete task queue of the selected VM will be searched by performing N/m comparisons. If the task is not adjusted in the queue of selected VM then the next VM will with the next MCT be selected and in the worst case, all the VMs will be checked by performing m comparisons to adjust the task in any of the total VM which becomes N(m + m(N/m)). In case, a suitable position for the current task is found then in the worst case, all the tasks in the task queue will be checked for violation of task deadline after inserting the new task in the task queue of VM. This is because by inserting a new task in the task queue, the remaining tasks will be moved one step back in the queue. Therefore, the time complexity of the RADL scheduler will become N(m + m(N/m + N/m)) or N(m + m(2N/m)). Thus, the overall complexity of the proposed approach has become $O(mN + N^2) \approx O(N^2)$. Table 3.2 shows complexity analysis of the RADL and other state-of-the-art task

scheduling algorithms.

TABLE 3.2: Computational complexity of scheduling algorithms

Algorithms	Dy-MaxMin [33]	PSSELB [19]	DLBA [35]	RALBA [1]	DC-DLBA [34]	MODE [76]	RADL
Time Complexity	$O(mN^2)$	$O(mN.n^2/2)$	$O(N^2)$	$O(mN^2)$	$O(m^2N^2)$	$O(MN^2)$	$O(N^2)$

TABLE 3.3: Scheduling overhead analysis of algorithms

Algorithms	RADL	PSSELB	DLBA	RALBA	MODE	DC-DLBA
Overhead (N * Dy-MaxMin)	1.90	2.06	2.29	2.32	5.52	7.44

Scheduling execution overhead analysis has been performed for the proposed algorithm (RADL) and state-of-the-art scheduling algorithms such as Dy-MaxMin [33], PSSELB [19], DLBA [35], RALBA [1], and DC-DLBA [34]. Table 3.3 shows the overhead analysis of scheduling algorithms in best to worst order (w.r.t to N times of *Dy-MaxMin* [33]). Dy-MaxMin shows the minimal scheduling overhead as compared to the others. Table 3.3 delineates that DC-DLBA [34] has maximum overhead in terms of scheduling decision, due to run time tasks migration, creation, and removal of VMs. Overhead analysis depicted in Table 3.3 shows that the RADL will perform comparatively in a scalable manner for larger workloads.

3.3 Experimental Evaluation and Discussions

This Section presents the experimental evaluation of the proposed scheduling approach (RADL) and state-of-the-art scheduling technique.

3.3.1 Experimental Setup

To empirically evaluate the performance of the proposed approach, the Cloudsim [11, 61] simulator has been used. *Cloudsim* is a renowned open-source Java-based

simulator used for performance analysis and modeling of environment and services. The experimental environment includes CPU (Intel Dual-Core T4300 2.1 each core, Memory (3.00 GB) HD 320 GB, eclipse Oxygen.3a and Cloudsim 3.0.3 (as shown in Table 3.4). Our implementation extends some of the existing classes like Datacenter, Broker, and Cloudlet. A simulation environment for all experiments consists of 50 VMs, 30 hosts, and a Datacenter. Details of the simulation environment are shown in Table 3.4.

TABLE 3.4: Simulation environment configuration

Simulator/Version	Cloudsim version 3.0.3
Experimental environment	Intel Dual-Core T4300 2.1 each core, Memory (3.00 GB) HD 320 GB
Total host machine	30
Host machine memory	15000 MBs each
Total Virtual Machine	50 heterogeneous VMs (shown in Figure 3.4) for Synthetic and GoCJ workload, 16 VMs for HCSP dataset
Total Cloudlets	500, 600, 700, 900, 1000

Figure 3.4 shows the configuration of heterogeneous VMs used in our experimentation. In our experimentation, two realistic cloud/cluster traces and MapReduce logs of M45 supercomputing clusters based benchmark datasets have been used that includes Google like realistic workload that is GoCJ [59, 62] (as shown in Figure 3.3(b)), and Expected Time to Compute (ETC) model based on HCSP instances [62]. Additionally, synthetic workload based benchmark dataset from [1] has been used for evaluation.

A number of influential parameters has been used for evaluating and comparing the proposed scheduling algorithm. These evaluation parameters include makespan [1, 19, 34, 35, 63], %age of task rejection [34], Mean ARUR [1, 19, 22, 33, 63], and average response time [19, 29, 33, 35]. However, most of the time improving one evaluation parameter may effect the performance of other i.e., executing tasks with shorter deadline before tasks with longer or no deadline can increase response time of tasks with longer deadline. This increases the importance to evaluate the overall performance of task scheduling algorithm as well. Overall

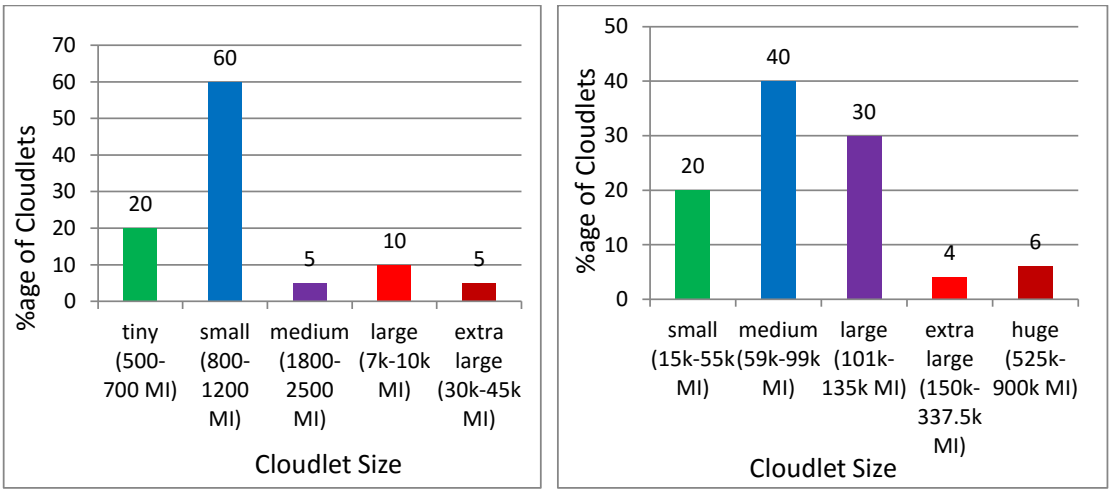
performance represents the combined effect of the scheduling algorithm for more than one evaluation parameters (inversely related ones). The detail regarding the overall performance of tasks scheduling heuristics will be discussed in chapter 4.

3.3.2 Workload Generation

Three scientific benchmark datasets have been used for experimentation. However, these datasets are customized by adding tasks deadline information using pattern followed in [34] for assigning deadlines to the tasks. To incorporate, the deadlines related information in the existing dataset, deadlines are assigned by following the Monte-Carlo simulation technique and the pattern used in DC-DLBA [34]. For this purpose, our implementation extends some of the existing classes of Cloudsim simulation environment like Datacenter, Broker, and Cloudlet class.
The GoCJ [59, 61, 62, 78] dataset is based on real world traces derived from traces of Google Cluster [86] and logs of MapReduce from M45 supercomputing cluster[87]. The GoCJ dataset using realistic high-performance computing cloud-tasks. The GoCJ dataset consists of tasks having different length i.e., 15000 MIs to 900000 MIs (as presented in Figure 3.3 (b)) and is generated using a renown Monte-Carlo simulation method. These tasks have been divided into five categories: small, medium, large, extra-large, and huge size tasks as shown in Figure 3.3 (b). The percentage of these tasks include: 20% tasks are of small size, 40% tasks are of medium size, the percentage of large size tasks are 30%, extra-large size tasks are of 4%, and 6% tasks are of huge size.

The synthetic benchmark dataset generated by [1] using monte-carlo technique. This dataset comprises tasks with different sizes ranges from 1 MI to 45000 MI and categorized as tiny, small, medium, large, and extra-large (as shown in Figure 3.3 (a)). The length of tiny tasks ranges from 1 to 250 MI, small from 800 to 1200 MI, medium from 1800 to 2500 MI, large from 7000 to 10000 MI, and Extra Large (XL) ranges from 30000 to 45000 MI. The synthetic workload based benchmark datasets are a more positively skewed dataset. A positively skewed dataset contains a large number of smaller tasks and only a few large size tasks, however, negatively skewed workload consists of a large number of longer tasks

and few smaller size tasks.



(a) Synthetic Workload dataset (b) GOCJ dataset

FIGURE 3.3: Number and sizes of Cloudlets for Synthetic Workload and GoCJ [59] dataset

To execute these tasks, Virtual Machines (VMs) with heterogeneous computation capability has been used. The computation power of the VMs used for tasks execution ranges from 100 *Million Instructions per Second* (MIPS) to 4000 MIPS. The Figure 3.4) shows the number of VMs with their computation power.

The Heterogeneous *Computing Scheduling Problem* (HCSP) [59, 62] benchmark dataset is based on an *Expected Time to Compute* (ETC) model in the form of HCSP instances. The ETC values are generated in such a way that had provided an accurate way for approximating the real behavior of the heterogeneous computing environment [59]. The HCSP instances are categorized into three complexity levels (small, medium, and large size instances).

The small size HCSP instances include 512 to 2048 tasks and 16 to 64 machines. The medium size HCSP instances range from 4096 to 8192 and 128 to 256 machines. The large size HCSP instances contain tasks ranging from 16384 to 32768 and number of machines ranging from 512 to 1024. HCSP instances are named as c-hilo, c-lohi, s-hilo, s-lohi, i-hilo, and i-lohi. This naming scheme follows C-THMH pattern where c represents consistency type (c used for consistent, s for semi-consistent and i represent inconsistency), TH and MH shows task and machine level heterogeneity, respectively. Similarly, hi represent high heterogeneity

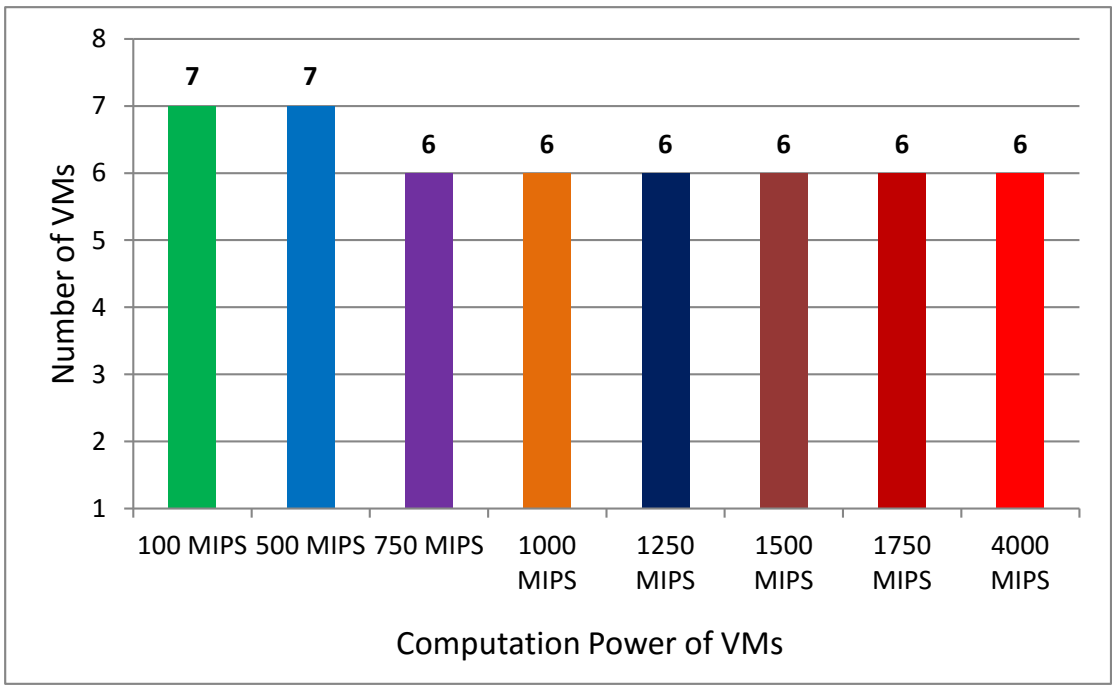


FIGURE 3.4: Computation power of VMs

and li used for low heterogeneity. The VMs computation power used for HCSP instances in [62] ranging from 19 MIPS to 7000 MIPS. In this research, we have used small size HCSP instances with small complexity level and include c-lohi, c-hilo, i-lohi, and i-hilo.

3.3.3 Simulation Results

This section presents simulation results of the proposed approach RADL and a comparison of these results with the existing state-of-the-art DC-DLBA [34], DLBA [35], Dynamic Max-Min (Dy-MaxMin) [33], PSSELB [19], RALBA [1], and MODE [76]. The performance evaluation metrics include makespan [1, 19, 34, 35], %age of task rejection [34], Mean ARUR [1, 19, 33], and average response time [19, 33, 35]. The higher value of ARUR shows better performance, however, lower value of makespan, task rejection, and task response time shows better performance.

Experimental results based on makespan using Synthetic workload [1] dataset is shown in Figure 3.5 (a). Synthetic workload dataset is a positively skewed dataset where most of the tasks are smaller in size and only few tasks are of

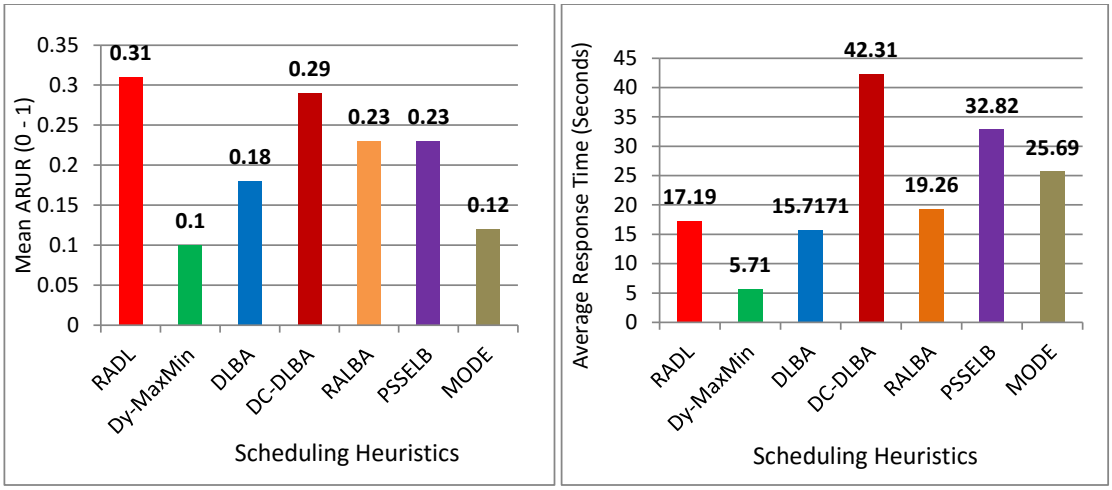


FIGURE 3.6: ARUR and Task Response Time results for Synthetic dataset based executions

larger size. Figure 3.5 (a) reveals that the proposed approach RADL has attained 43.38%, 95.78%, 259.2%, 49.78%, 156.28%, and 378.21% lower makespan and 42.86%, 145%, 303.57%, 562.14%, 1195.71%, and 857.14% reduced task rejection (as shown in Figure 3.5 (b)) using the Synthetic workload dataset as compared to Dy-MaxMin [33], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and MODE [76] respectively.

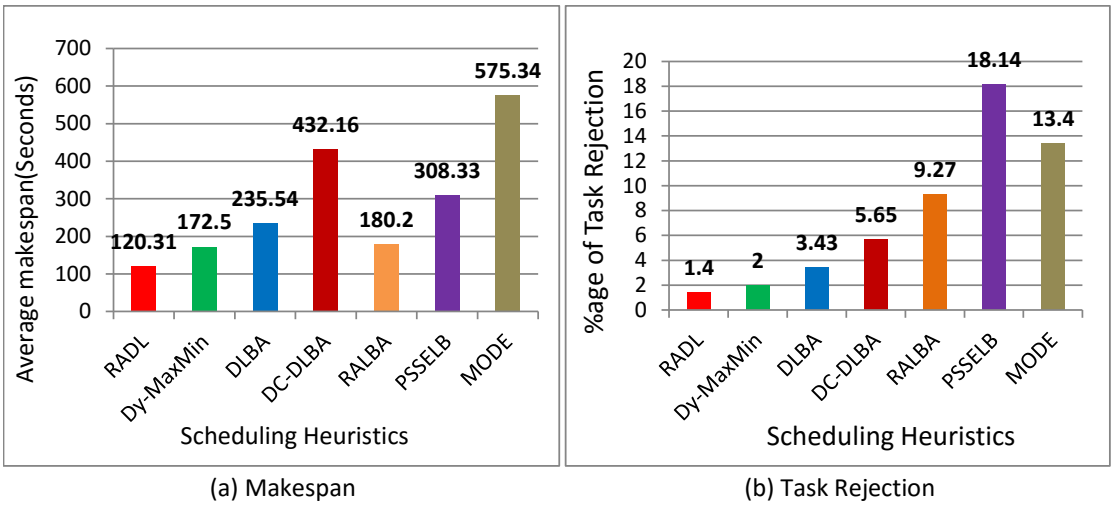
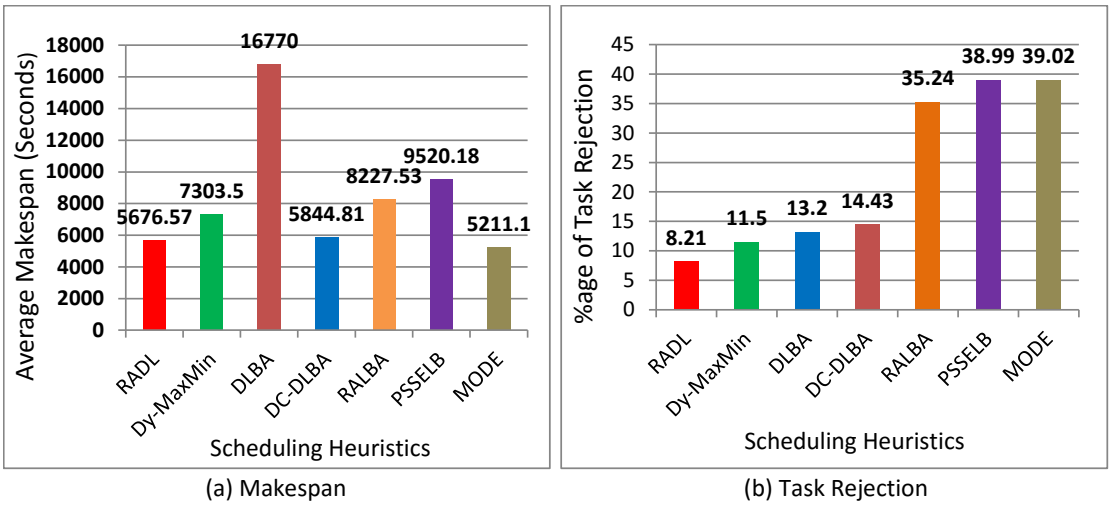


FIGURE 3.5: Makespan and Task Rejection results for Synthetic dataset

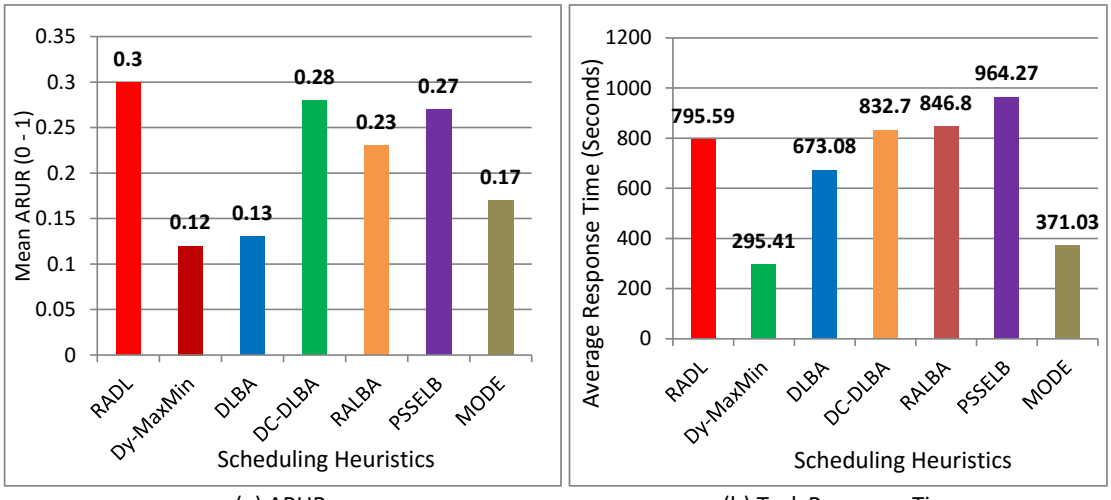
Figure 3.6 (a) shows that RADL has gain 67.74%, 41.93%, 6.45%, 25.80%, 25.80%, 67.29% higher performance in terms of mean resource-utilization for the Synthetic workload [1] dataset as compared to Dy-MaxMin [33], DLBA [35], DC-DLBA

[34], RALBA [1], PSSELB [19], and MODE [76], respectively. The results presented in Figure 3.6 (b) reveal that the proposed RADL mechanism has attained 146.13%, 12.04%, 90.92%, and 49.44% improved task response time as compared to DC-DLBA [34] RALBA [1], PSSELB [19], and MODE [76], respectively for the Synthetic workload dataset. The Dy-MaxMin scheduling algorithm performs well in terms of response time as compared to the RADL approach and other state-of-the-art approaches. It's because, Dy-MaxMin update task and VM status table at runtime that helps to schedule tasks in realistic expected completion time. However, the performance of the Dy-MaxMin is poor for other parameters like ARUR, makespan, and task rejection among others.



(a) Makespan (b) Task Rejection

FIGURE 3.7: Makespan and Task Rejection results for GoCJ dataset



(a) ARUR (b) Task Response Time

FIGURE 3.8: ARUR and Task Response Time results using GoCJ datasets

Second benchmark dataset used for evaluation of the proposed approach is GOCJ dataset proposed in [59] which comprise of large number of larger tasks as compared to small size tasks. Experimental results show that the proposed approach RADL has attained 28.66%, 195.42%, 2.96%, 44.94%, 67.71%, -8.19% lower makespan (as presented in Figure 3.7 (a)) and 40.07%, 60.78%, 75.76%, 329.23%, 374.9%, and 375.27% reduced task rejection (as shown in Figure 3.7 (b)) for the execution of GoCJ benchmark dataset as compared to Dy-MaxMin [33], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and MODE [76], respectively. Figure 3.8 (a) shows that RADL scheduling scheme has attained 60%, 56.67%, 6.67%, 23.33%, 10%, and 43.33% higher ARUR for the execution of GoCJ dataset as compared to the Dy-MaxMin [33], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and MODE [76], respectively. For the execution of GOCJ dataset, the RADL approach results in 4.66%, 6.43%, and 21.20% improved response time than DC-DLBA [34], RALBA [1], and PSSELB [19], respectively (as presented in Figure 3.8 (b)).

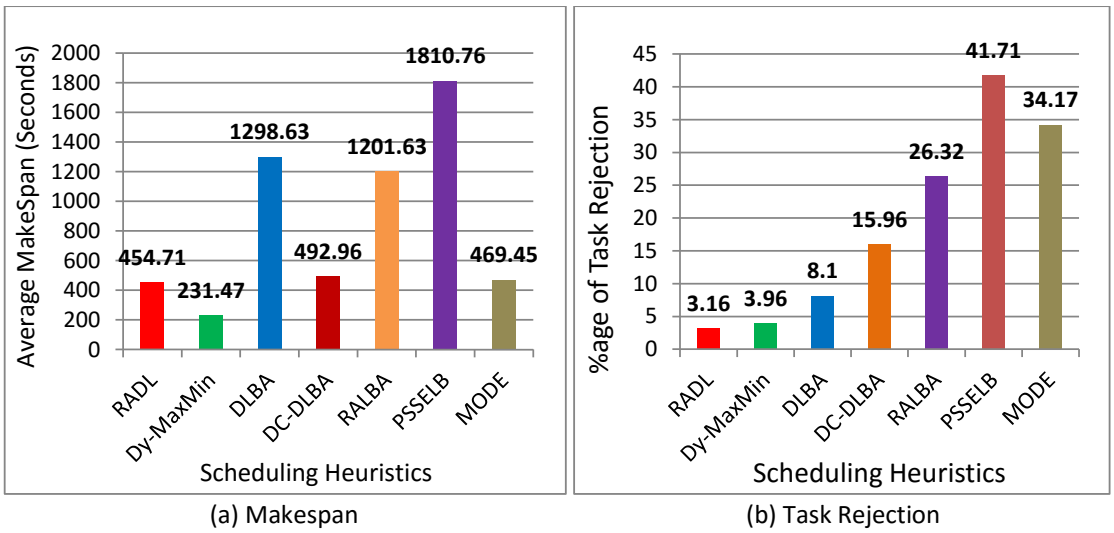


FIGURE 3.9: Makespan and Task Rejection results using HCSP dataset

However, response time of the proposed approach is higher than the Dy-MaxMin [33], MODE [76], and RALBA [1]. This is because, reduction in the response time is the core objective of the Dy-MaxMin [33] and MODE [76] create more new VMs when the load increases which results in reduced response time. Similarly, RALBA [1] not supporting the deadline.

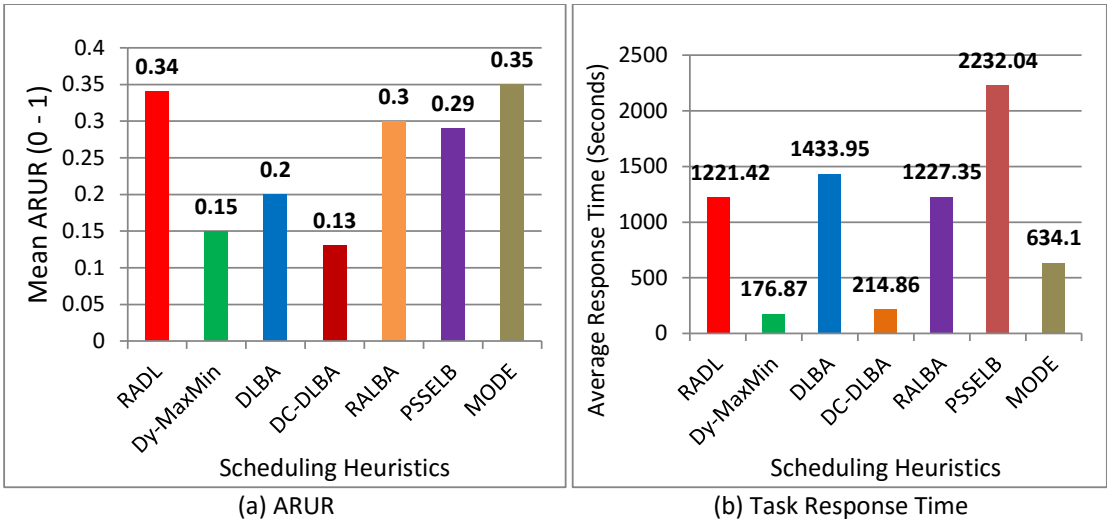


FIGURE 3.10: ARUR and Task Response Time results using HCSP dataset

Heterogeneous Computing Scheduling Problem (HCSP) [59, 62] benchmark dataset consist of HCSP instances and is based on an *Expected Time to Compute* (ETC) model. For the execution of HCSP dataset (as shown in Figure 3.9 (a)), the proposed scheduling heuristic RADL has attain -49%, 185.59%, 8.41%, 164.26%, 298.22%, and 3.24% reduced makespan and 25.31%, 156.33%, 405.06%, 732.91%, 1219.9%, and 981.32% reduced task rejection as compared to the Dy-MaxMin [33], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and MODE [76], respectively. Figure 3.10 (a) shows that RADL heuristic has attained 55.88%, 41.17%, 61.76%, 11.76%, and 14.7% higher performance in terms of mean resource utilization for HCSP [62] dataset as compared to Dy-MaxMin [33], DLBA [35], DC-DLBA [34], RALBA [1] and PSSELB [19], respectively. Figure 3.10 (b) shows that RADL has achieved 17.4%, 1%, and 82.74% improved task response time than DLBA [35], RALBA [1] and PSSELB [19] using HCSP benchmark dataset. The DC-DLBA [34] and MODE [76] have lower task response time than the proposed technique (as shown in Figure 3.10 (b)). This is because, these techniques create new VMs at run time when the workload on VMs exceed the given threshold or the average number of rejected tasks reached to the predefined limit. This results in reduced task response time for these approaches.

3.4 Results and Discussion

In-depth analysis of the experimental results reveals that the proposed scheduling technique (RADL) has attained significant improvements in terms of resource utilization, meeting task deadlines, and reducing makespan. It is because the proposed approach is resource and deadline aware. Experiments have been performed on three different benchmark datasets. The GoCJ dataset used in the experimentation consists of relatively larger size tasks as compared to Synthetic workload. Moreover, the HCSP instances based dataset used for experimentation, consisting of c_hilo, c_lohi, i_hilo, and i_lohi where c and i show consistency level of the workload, where c represents consistent workload and i is used for inconsistent workload, lo represent low heterogeneity and hi represent high heterogeneity of the workload and resources.

DC-DLBA [34] achieved higher makespan and response time on synthetic workload and comparatively lower makespan for GoCJ and HCSP datasets. This is due to a large number of smaller-size tasks in the synthetic workload where task rejection is comparatively small and few new VMs are created. RADL has achieved 259.2% improved makespan, 6.45% higher ARUR, 146.13% improved response time, and 303.57% lower task rejection on synthetic workload as compared to DC-DLBA [34]. Similarly, DC-DLBA [34] attained lower resource utilization on GoCJ and HCSP datasets as compared to synthetic workload. Moreover, DC-DLBA [34] has achieved lower task response time for the HCSP dataset and comparable task response time for the GoCJ dataset. This is because DC-DLBA [34] creates more VMs due to a large number of tasks in the HCSP dataset.

Dy-MaxMin [33] attained lower response time due to run-time updates of the task and VM status-tables which result in the mapping of tasks on realistic expected completion time. However, resource utilization is low and task rejection is high. This is because Dy-MaxMin neither creates new VMs nor gives priority to the tasks with shorter deadlines. The RADL attained 67.74%, 60%, and 55.88% higher resource-utilization than Dy-MaxMin [33] on synthetic, GoCJ, and HCSP datasets, respectively. Also, RADL attains 43.38% and 42.86% improved makespan and task rejection, respectively for synthetic workload dataset. Dy-

MaxMin [33] has a lower makespan for the execution of HCSP datasets. This is because the task of the HCSP dataset is of medium sizes.

The scrutinized results show that PSSELB [19] achieved comparatively better resource utilization than Dy-MaxMin [33], DLBA [35], RALBA [1], and MODE [76] for the execution of synthetic and GoCJ datasets and comparable results for HCSP datasets. This is because improving resource utilization is one of the key objectives of PSSELB [19]. However, the RADL scheduler outperforms as compared to PSSELB in terms of resource utilization, makespan, task response time, and task rejection for all three benchmark datasets. This is because the PSSELB [19] follow the Max-Min scheme which selects the largest tasks possibly assigns some of the larger tasks to the slower machine and also increases waiting time for the smaller size tasks which results in load imbalance. The percentage of task rejection is high for all three datasets because the PSSELB [19] do not consider deadline-based tasks.

DLBA [35] has gained lower task response time as compared to PSSELB [19], RALBA [1], DC-DLBA [34], and RADL for the execution of synthetic workload and GoCJ datasets. This is because the reduction in waiting time is the key scheduling objective of DLBA [35] and comparatively lower scheduling overhead for these datasets. However, RADL has achieved 41.93%, 56.66%, 41.18% higher ARUR, 95.78%, 195.42%, 185.95% improved makespan, 145%, 60.78%, and 156.33% reduced task rejection as compared to the DLBA [35] on synthetic workload, GoCJ, and HCSP datasets respectively.

Experimental results show that the RALBA [1] achieved improved performance in terms of makespan, and resource utilization as compared to PSSELB [19] and DLBA [35] and higher resource utilization than Dy-MaxMin [33] for all the three datasets. This is due to the resource awareness of the RALBA [1] scheduling scheme. The scrutinized results show that the RADL attained 49.78%, 44.94%, and 164.26% improved makespan, 25.8%, 23.33%, and 11.76% improved resource utilization, 12.04%, 6.44%, and 0.048% reduced task response time and 562.14%, 329.23%, and 732.91% lower task rejection ratio as compared to RALBA [1] on synthetic workload, GoCJ, and HCSP datasets, respectively. This is because RALBA [1] is based on the Max-Min scheme which increases waiting time for smaller tasks

and is not deadline aware.

MODE [76] has achieved a lower response time for GoCJ, HCSP, higher response time for Synthetic workload. This is because MODE [76] lease new VMs at run time for the comparatively high workload. Moreover, MODE [76] has shown poor performance in terms of resource utilization and task rejection ratio for all the three benchmark datasets. This is because the MODE [76] scheduling technique is not resource and deadline aware.

The overall in-depth analysis of the experimental results shows that RADL outperforms in terms of Resource utilization, meeting task deadlines, makespan, and task response time. However, RADL can not support workflow-based dynamic and SLA-aware task scheduling.

3.5 Chapter Summary

Cloud computing has become an attractive platform for cloud service providers and service users. To get the full benefit of cloud and achieve high user satisfaction, cloud service providers demand load balancing in terms of higher resource utilization and executing users' tasks within their deadlines. This leads to a need for efficient task scheduling algorithms. To achieve higher resource utilization and meeting task deadlines, several tasks scheduling algorithms have been proposed. However, the majority of these approaches are unable to achieve high utilization of cloud resources and to meet task deadlines.

In this chapter, a *Resource-Aware Dynamic Load-balancer for Deadline Constrained Cloud Tasks* (RADL) technique has been proposed. The proposed approach is evaluated against state-of-the-art scheduling heuristics like DC-DLBA [34], Dy-MaxMin [33], DLBA [35], RALBA [1], and PSSELB [19] using three benchmark datasets. Experimental results show that the proposed approach outperforms as compared to the state-of-the-art tasks scheduling algorithms in terms of ARUR and in meeting the task deadlines, makespan, and task response time.

The proposed approach reduces task rejection by maximizing the utilization of existing resources and adjusting the already mapped tasks. However, this may

increase task rejection and task scheduling overhead if all the newly arrived tasks having shorter deadlines. In the future, we intend to extend the proposed approach for workflow-based (dependent tasks) dynamic task scheduling.

Chapter 4

OG-RADL: Overall Performance Based Resource Aware Dynamic Load-balancer for Deadline Constrained Cloud Tasks

4.1 Introduction

Overall Gain (OG) is the overall performance of the cloud that compute and evaluate the combine affect of multiple evaluation parameters like Average Resource Utilization Ratio (ARUR), makespan, task response time, task rejection ratio, and throughput among others. The majority of state-of-the-art task scheduling heuristics improve a single parameter like makespan, resource utilization, or task waiting time as scheduling objectives. However, focusing on a single parameter can affect the performance of others, which can degrade the overall performance of the task scheduling algorithm. For example, sometimes improving cloud resource utilization and meeting tasks deadline can affect the makespan and task response time. A number of state-of-the-art scheduling algorithms consider more than one evaluation parameter as a scheduling objective, however, these approaches evaluate each parameter individually. There is a need to evaluate and increase overall performance [72] by combining multiple evaluation parameters like cloud resource

utilization, task rejection ratio, makespan, and task response time among others. In this chapter, a resource-aware dynamic scheduling algorithm named *Overall Performance-based Resource Aware Dynamic Load-balancer* (OG-RADL) for deadline constrained tasks is proposed. The proposed approach minimizes the load-imbalance issue, supports deadline-based tasks, and improves the overall cloud performance. Moreover, a novel normalization approach is also proposed that overcome the limitations of existing normalization techniques like Min-Max, Z-score, and Advanced Min-Max Z-score Decimal Scaling (AMZD). The prime objective of the OG-RADL scheduler is to improve cloud resource utilization, minimize task rejection, and enhance the overall performance of the cloud. In summary, our contributions to this research are as follows:

- Extensive analysis of state-of-the-art static, dynamic, and batch dynamic scheduling heuristics to identify their strengths and weaknesses.
- A novel and dynamic load-balancing algorithm has been proposed for none-preemptive, independent, and compute-intensive tasks that maximize utilization of cloud resources, reduce task rejection and increase the overall performance of cloud datacenter.
- Overall performance (i.e., Overall Gain) is computed by using a novel normalization technique that overcomes the limitations of state-of-the-art normalization approaches.
- Comprehensive evaluation and performance analysis of the proposed approach against existing state-of-the-art cloud task scheduling algorithms has been conducted using three benchmark datasets.

The rest of the chapter is organized as: The introduction part is discussed in Section 4.1. Section 4.2 presents the proposed load-balancing algorithm, OG-RADL system overview and background, OG-RADL system architecture, system and performance model, and OG-RADL algorithm. Experimental evaluation and discussion that includes experimental setup, workload generation, normalization of evaluation parameters, and simulation results are discussed in Section 4.3, Section

4.4 presents Simulation results and discussion.Section 4.5 concludes the chapter and discusses the future directions.

4.2 Proposed Load-balancing Algorithm

This Section presents a complete overview of the proposed scheduling technique OG-RADL that includes, the system architecture of the proposed approach, system and performance model, OG-RADL algorithm, algorithmic complexity, and scheduling overhead analysis.

4.2.1 OG-RADL System Overview and Background

The literature study reveals that the majority of state-of-the-art task scheduling heuristics suffer from issues like under-utilization of cloud resources, load-imbalance, high task rejection ratio, and makespan. Moreover, most of these algorithms are unable to map the newly arrived tasks with shorter deadlines within their deadlines. In this chapter, an Overall Performance based Resource Aware Dynamic Load-balancer is proposed that improves utilization of cloud resources, reduces makespan, enhances meeting task deadlines, and improves overall performance gain of the cloud. The OG-RADL comprises two schedulers: i.e., OG-RADL-Scheduler and *Shifter Scheduler* (S-Scheduler). The proposed approach is implemented using a renowned Cloudsim [11] simulator. Cloudlet is used as a synonym for the task in the Cloudsim simulator. A Cloud datacenter entity is used to simulate cloud infrastructure level services. The data center is composed of host machines and storage servers. The power of the datacenter is represented by the computation capabilities of all of the host machines on that datacenter. VMs are created on each host based on the *Cloud Service Provider* (CSP) defined VMs allocation policy. OG-RADL scheduling scheme map tasks to VMs on a Just-in-Time based to improve the tasks response time of deadline constrained tasks, reduce task rejection, and enhance overall cloud performance.

4.2.2 OG-RADL System Architecture

OG-RADL is a dynamic cloud scheduling scheme that maps incoming tasks (on a Just-in-Time basis) in a balanced way. Figure 4.1 presents the system architecture of the proposed model. The basic notations, definitions, and terminologies that have been used in the mathematical expressions of the proposed task scheduling technique are given under the heading Symbols. The proposed technique consists of two algorithms, OG-RADL-Scheduler and S-Scheduler. Steps 1 to 7, 11, 12, and 13 of the Figure 4.1 are similar to the Figure 3.2 and have been discussed in section 3.2.2. However, Figure 4.1 is extended by adding steps 8, 9, 10, 14, 15, and 16.

Steps 8, 9, and 10 of Figure 4.1 are used to check the deadline violation of the remaining tasks in the task queue of VM_j after shifting one step back in the task queue. This process is repeated until a task is found whose deadline is violated or the complete queue of tasks is searched. At step 10 the flag value is set to zero or one. If the flag value is zero (0), then steps 6 to 10 or 12 are executed again. In case, the flag value is equal to 1 (true) then tasks positions are updated as represented in step 11. Task T_i will be mapped to VM_j. Task status and VM status tables are updated (shown in Figure 4.1, Step 11)

In step 14, the values of evaluation parameters are ARUR, makespan, average task response time, and average task rejection are computed. In the next step (i.e., step 15), the values of makespan, average task response time, and average task rejection are normalized using Eq. 4.6. In step 16, the overall performance gain is computed using Eq. 4.2.

4.2.3 OG-RADL System Model

This Section presents the OG-RADL performance model and the system model using mathematical expressions. To describe, the performance of the OG-RADL, a unified cloud system model is formed. A cloud datacenter consists of a set of VMs (VMS, as shown in Eq. (3.1) and VM is represented as VM_j.

CT_{ij} is the expected completion time of task T_i on VM_j and mathematically

expressed using Eq. (3.2), vRT is the VM ready time i.e., the current workload of the VM_j and is computed using Eq. (3.4)

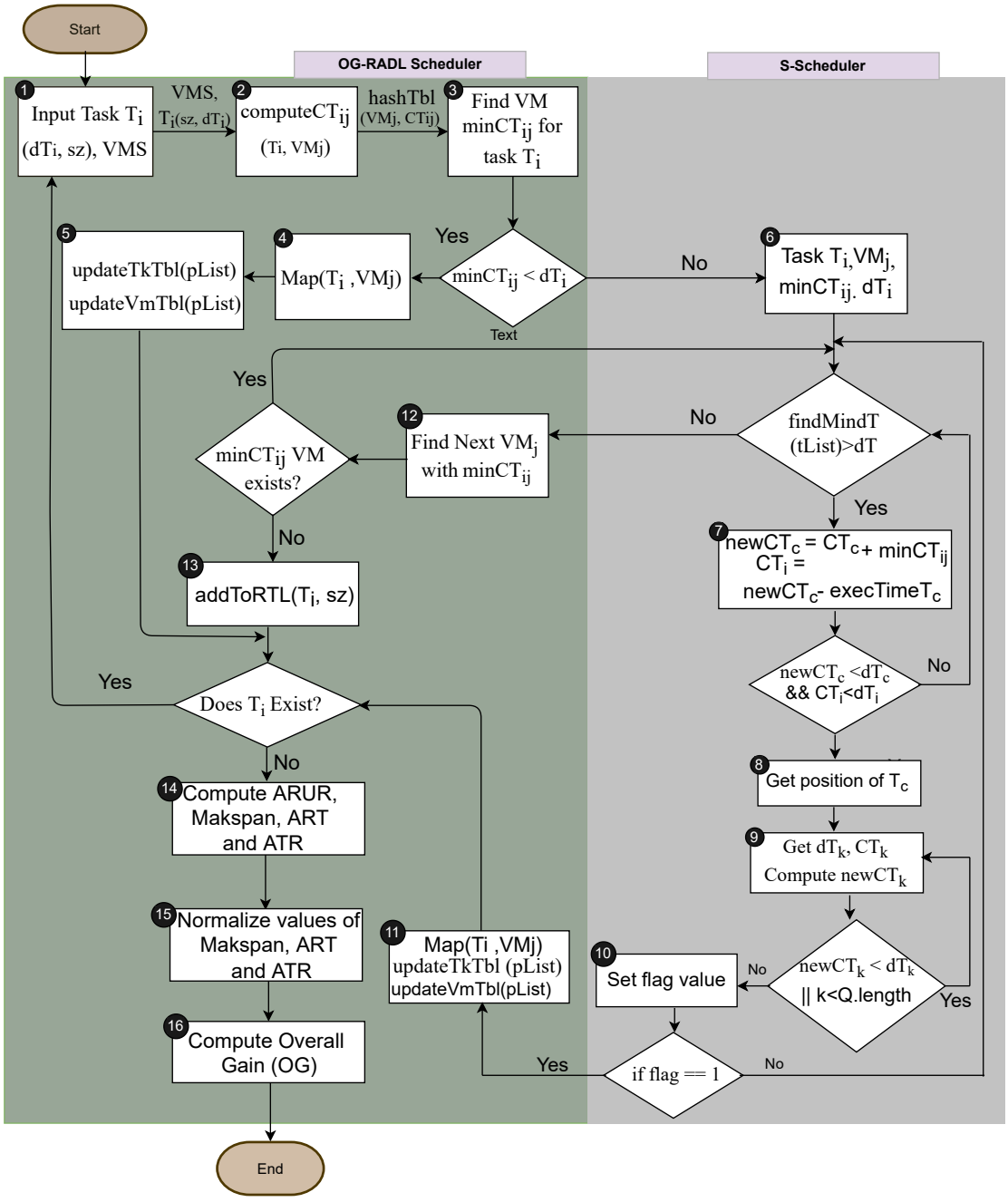


FIGURE 4.1: OG-RADL Scheduler

$vmExecT_{ij}$ is the execution time of task T_i on VM_j. The $vmExecT_{ij}$ is computed (as depicted in Eq. (3.3) [1]) as dividing the task size (in MI) by computation power of VM (in MIPS). The Eq. (3.5) shows that $minCT_i$ is the minimum completion time of task T_i on VM_j that execute the task T_i in the least amount of time. In case the completion time of VM_j is greater than the deadline of task

T_i then a suitable position for T_i will be checked in the tasks queue of VM_j. This will return a true value or false value (as shown in Eq. (3.6)).

The Eq. (3.7) [1] shows that VM_CT_j is the expected completion time of all assigned tasks to VM_j and is used for mapping tasks to the VM that executes them in minimum time. Therefore, during mapping, VM idle time is not considered. However, the actual completion time of VM_j is calculated after the execution of all assigned tasks that comprise tasks execution time, VM idle time, and other unexpected delays, etc., as well. Moreover, the proposed approach updates VM status at run time to get a more realistic expected completion time of VMs.

The new expected completion time of candidate task T_c ($newCT_c$) is the updated completion time (after shifting) of task T_c and is computed using Eq. (4.1).

$$newCT_c = CT_c + minCT_{ij} \tag{4.1}$$

, where CT_c is the completion time of candidate task and $minCT_{ij}$ is the minimum completion time of task T_i on VM_j.

$RspTime_i$ represents the Response time of task T_i which is the difference of task execution start time and task arrival time. $RspTime_i$ is computed using Eq. (3.8).

4.2.4 OG-RADL Performance Model

The performance of the proposed technique OG-RADL is evaluated against the state-of-the-art approaches in terms of ARUR which is computed using Eq. (3.10) [1, 19], makespan (mathematically represented using Eq. (3.9) [1, 23, 34, 59, 62]), percentage of task rejection (computed using Eq. (3.11)), and average task response time which computed using Eq. (3.12).

$$OG = \left(\frac{\sum_{i=0}^{p}(p_i)}{p}\right) \tag{4.2}$$

Overall Gain (OG) is the overall performance results of the proposed approach and is one of the contributions of this research which is published in [36]. The OG

is computed mathematically using Eq. (4.2). In Eq. (4.2), p represents the total number of performance evaluation parameters like ARUR, makespan, %ageTR, AvgRT, and p_i shows the value of each parameter. The value of p_i lies between 0 and 1 and is normalized using Eq. (4.6).

4.2.5 OG-RADL Algorithms

This section discusses the proposed scheduling technique OG-RADL which has two algorithms OG-RADL scheduler and S-Scheduler. OG-RADL scheduler assign tasks to VMs based on *Minimum Completion Time* (MCT) and update VM and task status. S-Scheduler investigates a suitable position for task T_i in the VM task queue which executes that task in the least time.

4.2.5.1 OG-RADL Scheduler

To perform mapping of tasks on VMs, OG-RADL scheduler (Algorithm 3) receives task T_i with their deadline (dT_i) in *Milli-Second* (MS) and task size in *Million Instructions* (MIs), and set of VMs (VMS) with their computation capability in *Million Instructions per Second* (MIPs) as input parameters. Outputs of Algorithm 3 include mapped tasks on VMs and *Rejected Task List* (RTList). Lines 1 to 3 of Algorithm 3 represents the necessary initialization of the OG-RADL scheduler. The while loop (lines 4-35, Algorithm 3) is similar to that of algorithm 1 which is discussed in detail at section 3.2.5.1.

Lines 36-39 of algorithm 3 are extending algorithm 1. Lines 36-38 of algorithm 3 are executed when there is no un-mapped task to be scheduled and the values of Average Resource Utilization Ratio (ARUR), Makespan, percentage of Task Rejection (%ageTR), and Average Response Time (AvgRT) are computed (lines 36-38, Algorithm 3) respectively. The overall gain is calculated at line 39 Algorithm 3. The makespan$'$, %ageTR$'$, and AvgRT$'$ are the normalized values of makespan, percentage of task rejection (%ageTR), and average task response time (AvgRT).

Algorithm 3: OG-RADL Scheduler

Input : VM list with their computation power, Task T_i with size "sz" and deadline "dT_i"

Output: Map (T_i, VM_j) i.e Mapping of Task T_i to VM_j

```
1  CT_ij = 0, VmId = 0, minCT_ij = 0;
2  Vm V = null, RTL = null, findSPQ = null;
3  ARUR= 0, Makespan=0, %ageTR =0, AvgRT=0;
4  while T_i != null do
5      for j = 1 to m do
6          CT_ij = computeCT_ij (T_i, VM_j);
7          VmId = VMList.getVmId(j);
8          hashMap.add(VmId, CT_ij);
9      end
10     VmId = findVmWithMCT(hashMap);
11     minCT_ij = hashMap.get(VmId);
12     if minCT_ij < dT_i then
13         Map.add(T_i, VM_j);
14         updateTkTbl(pList);
15         updateVmTbl(pList);
16     else
17         repeat
18             findSPQ = S-Scheduler (T_i, VM_j, dT_i, minCT_ij);
19             minCT_ij);
20             if findSPQ == true then
21                 Map.add(T_i, VM_j);
22                 updateTkTbl(pList);
23                 updateVmTbl(pList);
24             else
25                 V_j = findNVmWithMCT(hashMap);
26                 if V_j != null then
27                     minCT_ij = V_j.getKeyValue();
28                 end
29             end
30         until findSPQ == true or V_j = null;
31         if findSPQ == false then
32             RTL.add (T_i);
33         end
34     end
35 end
36 Makespan = getMakespan(vList, cList);
37 %ageTR = getPoTR(vList, cList);
38 AvgRT = getAvgRT(vList, cList);
39 double OG = (ARUR + Makespan' + %ageTR' + AvgRT')/4;
```

4.2.5.2 S-Scheduler

The OG-RADL scheduler (Algorithm 3) calls *S-Scheduler* (Algorithm 4) if the task deadline is less than the $minCT_{ij}$. The input parameters of *S-Scheduler* consists of VM_j, $minCT_{ij}$, and task T_i with the deadline (dT_i).

Algorithm 4: S-Scheduler
Input : T_i with deadline dT_i, VM_j and $minCT_{ij}$
Output: true or false

```
1  T_c = null, dT_k = null;
2  newCT_c = 0, CT_i = 0;
3  ExecTimeT_i =0, flag = 1;
4  execTimeT_c = 0;
5  T_c = findMindT_i (tList) > dT_i;
6  if T_c == null then
7      return false;
8  else
9      repeat
10         execTimeT_c = TEMap(T_c);
11         newCT_c = CT_c + minCT_ij;
12         CT_i = newCT_c - execTimeT_c ;
13         if (newCT_c < dT_c && CT_i < dT_i) then
14             break:
15         else
16             T_c = findNextMindT_i(tList) > dT_i
17         end
18         ;
19     until tList.size();
20     PosT_c = getPosT_c(TEMap);
21     for k = PosT_c to vmQ.length do
22         dT_k = getDeadline(T_k);
23         CT_k = getCT_k (TEMap);
24         newCT_k = CT_k + minCT_ij;
25         if (newCT_k > dT_k) then
26             flag = 0;
27             break:
28         end
29     end
30     if (flag == 1) then
31         updateTaskPos();
32         return true;
33     else
34         return false;
35     end
36 end
```

As output, a true or false value is returned. Lines 1-4 show the necessary initialization of Algorithm 4. The first part of algorithm 4 (lines 5-19) tries to identify the suitable position for the newly arrived task in the task queue of VM_j and is discussed in detail in algorithm 2, Section 3.2.5.2. This part of algorithm 4 tries

to identify the suitable position for the newly arrived task in the task queue of VM_j.

The position of candidate task T_c in the queue of VM_j is identified (line 20, Algorithm 4). The for loop (lines 20-29, Algorithm 4) scan all the remaining tasks in the queue of VM_j to check their execution within their deadline. The deadline and completion time of task T_k are computed (lines 22-23, Algorithm 4).

The updated completion time (after shifting) of T_k ($newCT_k$) is computed and compared with deadline dT_k (lines 24-25, Algorithm 4). If $newCT_c$ is greater than dT_k (line 25, Algorithm 4) then flag is set zero (line 26, Algorithm 4). The zero value of the flag indicates that T_k cannot be executed within their deadline. Moreover, the break statement is executed (line 27, Algorithm 4) and the control moves out of for loop. In case, if condition at line 25 is false, the next task in the queue of VM_j is selected and the for loop (lines 20-29, Algorithm 4) is executed again. In line 30 (Algorithm 4) flag value is checked and if the flag value is 1 then the task position is updated at the queue of VM_j and the true value is returned. if the flag value is not equal to 1 (line 30, Algorithm 4) then else part (lines 33-35, Algorithm 4) is executed and a false value is returned (line 34, Algorithm 4). The repeat-until and for loop in Algorithm 4 is executed repeatedly until a suitable position in the tasks queue of VM_j is determined or all the tasks in the queue are checked. In case, task T_c is equal to null (the suitable position for task T_i not found in the task queue of VM_i), the control is transferred back to the Algorithm 3 for finding next VM with next completion time.

The complexity of the OG-RADL scheduling technique is the same as of the RADL scheme which is discussed in section 3.2.6 of chapter 3.

4.3 Experimental Evaluation and Discussions

This Section shows the experimental evaluation of the proposed scheduling approach (OG-RADL) and state-of-the-art scheduling techniques. The OG-RADL scheduling approach is an extension of the RADL Scheduler, therefore, the same experimental setup and workload generation criteria has been used which are discussed in section 3.3.1 and section 3.3.2 respectively.

4.3.1 Normalization of Evaluation Parameters

In this Section a novel normalization technique has been proposed which overcomes the limitations of existing normalization approaches like Min-Max, Z-score, *Advanced on Min-Max Z-score Decimal scaling* (AMZD).

The Overall Gain (OG) (which combines metrics like ARUR, makespan, task rejection ratio, and task response time) has been used to evaluate the performance of the proposed approach. To compute OG, the values of these evaluation metrics need to be normalized. Normalization [88, 89] is a process that transform different values to a range like 0 and 1 [88]. It is the pre-processing, mapping or scaling technique used to convert highly skewed values into a new range of values [88, 90]. There are several normalization techniques used to normalize these values for different domains and objectives. These technique includes Min-Max use by [91, 92], z-score [90, 93], Decimal Scaling [94] and *Advanced on Min-Max Z-score Decimal scaling* (AMZD) proposed by [88].

Min-Max [92] is the most commonly used normalization technique in data mining. The Min-Max approach is suited for scenarios where maximum and minimum bounds are already known [93, 95]. This technique transforms minimum value to zero, maximum value to 1, and values that lie between the minimum and maximum are transformed into ranges between zero and one (as shown in Table 4.1). Eq. (4.3) shows formula for applying normalization using Min-Max normalization technique [92].

$$V_n = \left(\frac{V_o - minValue}{maxValue - minValue}\right) * (nMaxValue - nMinValue) + nMinValue \tag{4.3}$$

, where V_n is a new/normalized value, V_o represents the original value that needs to be transformed, minValue and maxValue shows the minimum values and maximum values in the list of value that need to be normalized. nMaxValue (normalized maxValue) shows upper range (i.e 1) and nMinValue (normalized minValue) shows minimum range (i.e., 0).

Z-score [93] is another commonly used normalization technique. Z-score is computed using standard deviation and arithmetic means of the given data. This

approach overcomes the issue of handling outliers, however, the Z-score is unable to produce data with the same scale. Moreover, Z-score values can be less than 0 i.e., negative, and can be greater than 1 (as presented in 4.1). Z-score is computed as shown in Eq. (4.4).

$$V_n = (\frac{V_o - meanValue}{std\_Dev}) \tag{4.4}$$

Advanced on Min-Max Z-score Decimal scaling (AMZD) based normalization technique has been proposed in [88]. AMZD is used to scale up integer values between a range of 0 and 1 value. This technique solves the issue of handling outliers, however, the normalized value does not have consistency in scaling values and favors the smallest value in the given data (as depicted in 4.1).

$$V_n = (\frac{V_o - (10^n - 1) * d_1}{10^n - 1}) \tag{4.5}$$

where n is the number of digits in the data element(i.e., the value that needs to be normalized), and d_1 shows the first digit of the data element. Decimal scaling [94] is a normalization technique that converts the given values among the range between -1 and 1. However, there is a need for such normalization techniques that can transform values into a range of values between 0 and 1 and should have the capability to resolve the issue of handling outliers. To overcome the issues of state-of-the-art normalization, a novel normalization technique has been developed (as shown in Eq. (4.6)).

$$V_n = \frac{V_o}{\sum_{i=0}^{k}(V_i)} \tag{4.6}$$

, where V_o is the original value that needs to be normalized and V_i value of i$^{\text{th}}$ parameter. The normalized value is subtracted from 1 to transform them to a uniform scale.

The proposed normalization technique has been compared with state-of-art techniques by using NNGC [89, 96] benchmark dataset.

Table 4.1 shows a comparison of the results of state-of-the-art normalization techniques with the proposed approach. The results of Min-Max and AMZD are taken from [96]. However, results of the z-score and proposed technique are gen-

erated using the formula discussed in Eq. 4.4 and 4.6. The value highlighted (in bold) show the highest and lowest normalized value using different normalization techniques. These results show that Min-Max normalization provides consistent results, however, it favors the highest value and penalizes the smallest value which affects normalized results in case outliers.

Z-score also provides consistency in their values, however, the value of the Z-score can be greater than 1 and less than 0, which is not applicable in our case.

The normalized results of AMZD are within the range of 0 and 1, however, their results not consistent with the change in value size. AMZD approach favors the smallest value and penalizes the largest one. However, the results of the proposed normalization approach are consistent, do not favor in particular value or set of values, and the results also lie between 0 and 1. The results generated using the proposed normalization approach reveals that the behavior of the normalized results is more similar to the actual values as compared to state-of-the-art approaches.

TABLE 4.1: Normalization results for NNGC Dataset

Values	Min-Max	Z-Score	AMZD	Proposed method (OG)
2677	**0**	**-1.222**	0.677	**0.052**
3083	0.062	-1.025	0.083	0.059
3539	0.132	-0.802	0.539	0.068
4032	0.208	-0.562	**0.032**	0.078
4452	0.273	-0.358	0.452	0.086
5100	0.372	-0.042	0.1	0.098
5944	0.502	0.369	**0.944**	0.115
6913	0.651	0.841	0.913	0.133
6936	0.654	0.853	0.936	0.134
9185	**1**	**1.948**	0.185	**0.177**

In some scenarios single step normalization is not enough like the makespan needs

TABLE 4.2: Two step normalization results for NNGC Dataset [89, 96]

Values	Min-Max	Z-Score	AMZD	Proposed method (OG)
2677	**1**	**2.222**	0.677	**0.948**
3083	0.938	2.025	0.917	0.940
3539	0.868	1.802	0.461	0.932
4032	0.792	1.562	**0.968**	0.923
4452	0.727	1.358	0.54821	0.914
5100	0.628	1.042	0.9	0.901
5944	0.498	0.631	**0.056**	0.885
6913	0.349	0.159	0.087	0.867
6936	0.346	0.147	0.064	0.866
9185	**0**	**-0.948**	0.815	**0.823**

to be transformed in the range of 0 and 1. In this case, the makespan value near 0 represents better performance while the value near 1 shows poor performance. However, there is a need to normalize data in the form that a higher value needs to represent good performance while a smaller value needs to represent poor performance. Table 4.2 depicts two-step normalization results: 1) In the first step, actual values are transformed in the range of 0 and 1. In the second step, the results of the first step are subtracted from 1. Table 4.2 shows that the proposed normalization technique performs better than the state-of-the-art approaches.

4.3.2 Simulation Results

This Section shows comparison of simulation results of the proposed approach OG-RADL and that of state-of-the-art task scheduling heuristics like Dynamic Max-Min (Dy-MaxMin) [33], DLBA [35], PSSELB [19], DC-DLBA [34], RALBA [1], and MODE [76].

Overall Gain (OG) is used as a performance evaluation metric of state-of-the-art and proposed task scheduling techniques. OG combines metrics like Mean ARUR [1, 19, 33], %age of task rejection [34], makespan [1, 19, 34, 35], and average response time [19, 33, 35] (as shown in Eq. (4.2)).

Experimental results presented in Figure 4.2 shows the comparison of the proposed approach in terms of an overall performance gain (OG) as compared to the

state-of-the-art tasks scheduling heuristics using Synthetic workload benchmark dataset. Figure 4.2 reveals that the proposed approach OG-RADL has gained 6.44, 8.21%, 16.6%, 11.45%, 27.86%, and 32.2% higher performance in terms OG on Synthetic workload dataset as compared to Dy-MaxMin [33], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and MODE [76] respectively. It's because the proposed approach is resource and deadline-aware. Moreover, the proposed scheduling scheme updates VMs status at run time that helps in mapping tasks according to the latest available information of VMs.

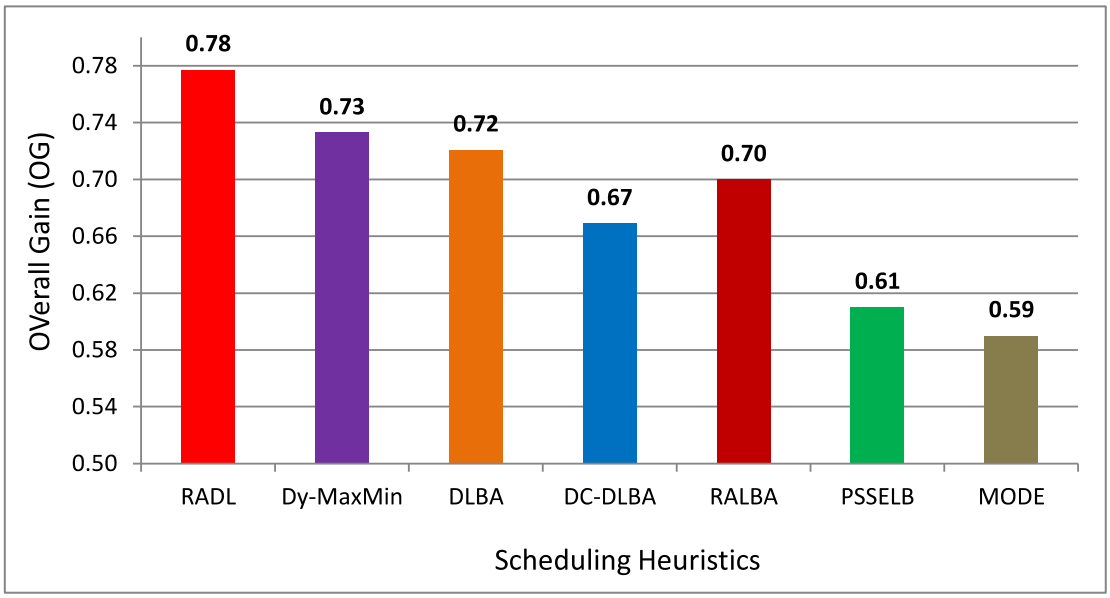


FIGURE 4.2: OG using Synthetic Benchmark Dataset

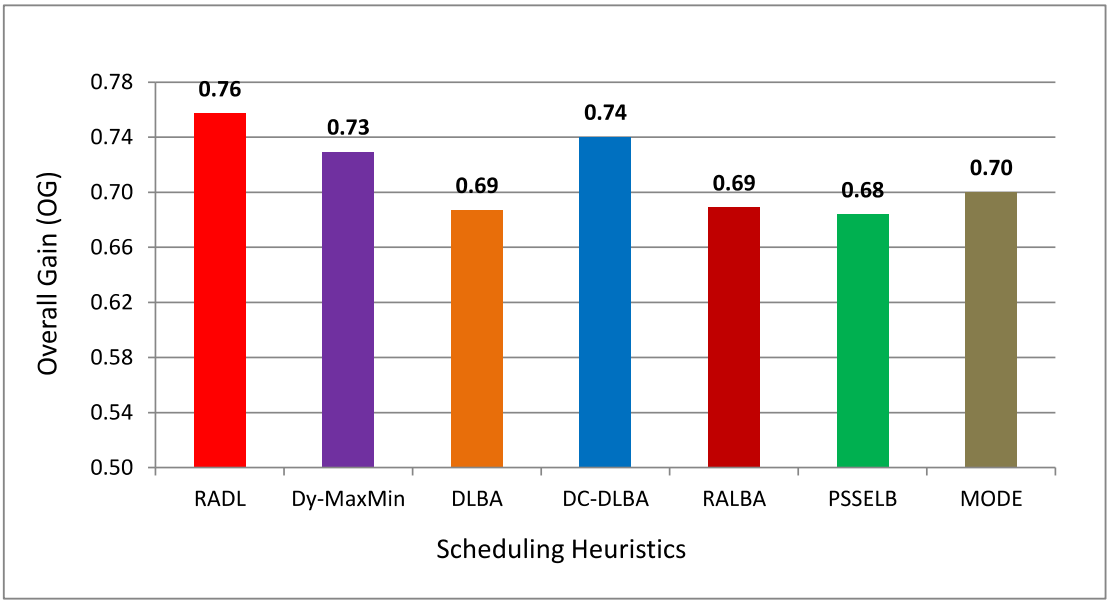


FIGURE 4.3: OG using GoCJ Benchmark Dataset

Figure 4.3 presents the overall performance results of the proposed approach as compared to state-of-the-art task scheduling heuristics using the GoCJ benchmark dataset. The results shown in Figure 4.3 shows that the proposed technique OG-RADL has attained 4.21%, 10.63%, 2.69%, 10.33%, 11.15%, and 8.57% better overall performance using GoCJ dataset against Dy-MaxMin, DLBA, DC-DLBA, RALBA, PSSELB, and MODE respectively. It's because the proposed algorithm is more scalable and map tasks to VMs based on updated VMs status.

Figure 4.4 depicts the comparison of overall performance achieved by the proposed approach OG-RADL using the HCSP benchmark dataset. Results presented in Figure 4.4 reveal that OG-RADL has gained 1.32%, 13.24%, 5.48%, 11.59%, 28.33%, and 10% higher overall performance as compared to Dy-MaxMin, DLBA, DC-DLBA, RALBA, PSSELB, and MODE respectively.

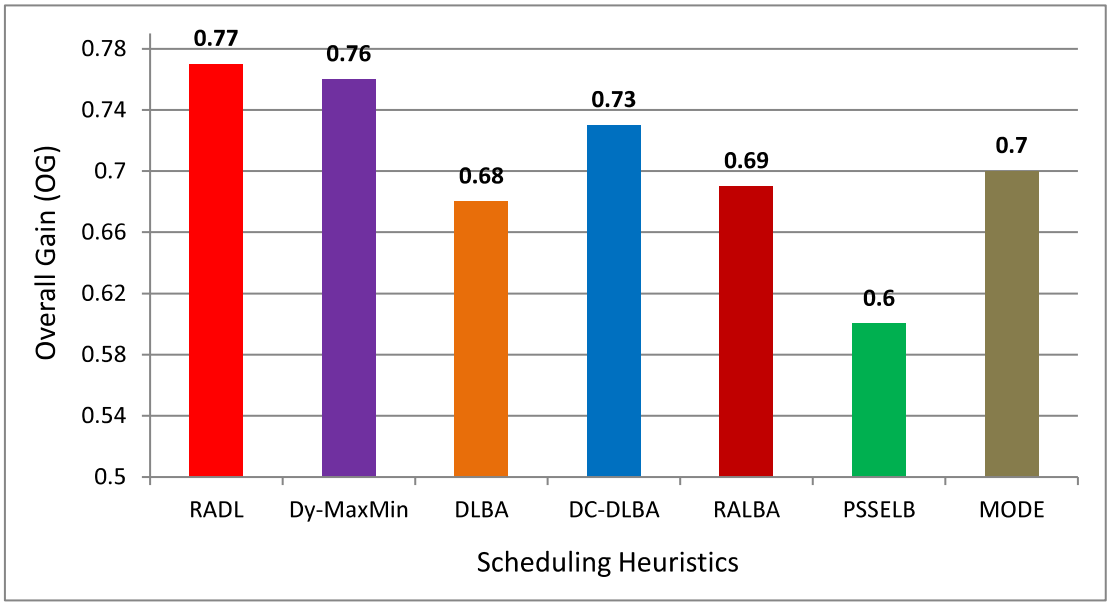


FIGURE 4.4: OG using HCSP Benchmark Dataset

4.4 Results and Discussion

A comprehensive investigation of overall performance-based experimental results shows that the proposed task scheduling approach i.e., OG-RADL outperforms in terms of overall gain as compared to state-of-the-art scheduling techniques for all three datasets. It is because the OG-RADL is deadline and resource-aware, and has comparatively less scheduling overhead. To thoroughly evaluate

the proposed approach, three different benchmark datasets (synthetic workload, GoCJ, and HCSP) have been used for experimentation. The synthetic workload-based benchmark dataset comparatively comprises a large number of smaller tasks and only a few larger size tasks. Such a dataset is called a positively skewed dataset. Moreover, the dataset that has a high number of large size tasks is termed as a negatively skewed dataset.

As compared to synthetic workload based dataset, the GoCJ dataset used in experimentation comprises a large number of large size tasks. Moreover, four (4) commonly used instances of the HCSP dataset has been used for experimentation which include i_hilo, i_lohi, c_lohi, and c_hilo instances where i and c show the consistency level of the workload. The i represents inconsistent workload and c is used for consistent workload, hi shows high heterogeneity, and lo represents low heterogeneity of the resource and workload.
DC-DLBA attained lower overall performance for synthetic workload as compared to GoCJ and HCSP datasets. It is due to the higher makespan and response time on synthetic workload and comparatively lower makespan for GoCJ and HCSP datasets. It's because of a large number of smaller size tasks in the synthetic workload where task rejection is comparatively smaller and only a few new VMs has created. OG-RADL has achieved 16.6% higher overall performance on synthetic workload as compared to DC-DLBA.
Similarly, MODE gained lower overall performance for synthetic workload as compared to GoCJ and HCSP datasets. It is due to the higher makespan and task response time on synthetic workload and comparatively lower makespan and task response time for GoCJ and HCSP datasets. It's because the number and sizes of tasks in the synthetic workload dataset are comparatively small where most of the tasks can be mapped within the specified threshold and few new VMs are created. OG-RADL has attained 32.2% higher overall performance on synthetic workload as compared to MODE.

The scrutinized results show that Dy-MaxMin achieved comparatively better overall performance than DLBA, DC-DLBA, RALBA, PSSELB, and MODE due to lower response time because of run time updates of VM and tasks status table which enable to map tasks on a more realistic expected completion time. How-

ever, OG-RADL attained 6.44% and 4.21% higher overall performance than Dy-MaxMin on synthetic, and GoCJ datasets respectively.

PSSELB attained lower overall performance for HCSP and synthetic workload-based datasets due to a large number of small size tasks. It's because PSSELB selects the largest tasks and can assign some large size tasks to the slower machine. The proposed approach has achieved 27.86%, 11.15%, and 28.33% improved overall performance as compared to PSSELB for synthetic workload, GoCJ, and HCSP dataset respectively. OG-RADL has gained 8.21%, 10.63%, and 13.24% higher overall performance against DLBA on GoCJ, HCSP, and synthetic workload datasets respectively.

The scrutinized results reveal that RALBA achieved higher overall performance as compared to PSSELB and DLBA. Experimental results show that OG-RADL achieved 11.45%, 10.33%, and 11.59% improved overall performance than RALBA on HCSP, GoCJ, and synthetic workload datasets respectively.

The overall comprehensive investigation of the experimental results shows that OG-RADL outperforms in terms of overall performance on all three benchmark datasets as compared Dy-MaxMin [33], DLBA [35], PSSELB [19], DC-DLBA [34], RALBA [1], and MODE [76]. However, OG-RADL cannot support SLA-aware and workflow-based dynamic task scheduling. Moreover, this chapter works on the assumption that every task is independent and compute-intensive. Therefore, this chapter does not consider memory, bandwidth, and communication latency.

4.5 Chapter Summary

Cloud computing has emerged as an attractive computing mode for industry and academia. The cloud computing environment provides services to the customer in a pay-as-you-go manner. To get the full benefit of the cloud and efficiently utilize cloud resources, load balancing plays an important role. To achieve maximum load balancing, there is a need to choose an efficient task scheduling and load balancing algorithm. To achieve higher resource utilization and load balancing, minimize tasks execution, and response time, a number of task scheduling algo-

rithms have been proposed. The majority of these approaches focus on improving one or two parameters like makespan or resource utilization. However, there is a need to improve the overall performance of the cloud datacenter. In this chapter, a *Overall Performance-based Resource-Aware Dynamic Load-balancer* (OG-RADL) for deadline constrained cloud tasks scheduling approach has been proposed. The proposed approach has the capability to map the workload of independent and compute-intensive tasks in a balanced manner according to the computation power of resources. The proposed approach is evaluated against state-of-the-art scheduling heuristics like DC-DLBA [34], Dy-MaxMin [33], DLBA [35], RALBA [1], PS-SELB [19], and MODE [76] using three benchmark datasets. Experimental results reveal that the proposed approach outperforms in terms of the overall performance gain as compared to state-of-the-art task scheduling algorithms.

In the future, we intend to extend the proposed approach by considering workflow-based dependent tasks where the execution sequence of tasks will matter. In the workload of workflow-based dependent tasks, memory, bandwidth, and communication latency are some of the essential factors that can hurdle cloud performance. Therefore, in the future, it is intended to use these factors as performance metrics as well.

Chapter 5

PSO-RADL: Particle Swarm Optimization based Resource and Deadline Aware dynamic Load-balancer for Deadline Constrained Cloud Tasks

5.1 Introduction

This chapter focus on Particle Swarm Optimization (PSO) based resource and deadline-aware dynamic task scheduling and load-balancing for deadline constrained cloud tasks. In chapters 3 and 4, heuristics-based tasks scheduling algorithm has been proposed. Heuristic-based algorithms provide near-optimal solutions for a specific problem and are considered problem-dependent approaches. These approaches are considered suitable for single objective or multiple objectives with non-conflicting parameters where improving one parameter can not directly affect the performance of other parameters. Figure 1.6 shows the basic classification of scheduling algorithms.

However, meta-heuristics based algorithms like PSO among others are designed

to provide a generalized optimal solution and can be applied to multiple domains [97]. These algorithms are considered more suitable to provide optimal solutions for multi-objective based problems with conflicting evaluation parameters like execution time and cost [7]. Moreover, task execution cost and penalty cost for deadline-aware schedulers are some of the key evaluation parameters that need to be considered as evaluation parameters for task schedulers. The aim of using meta-heuristics based technique is to incorporate and evaluate task execution and penalty cost along with improving resource utilization and load balancing, reducing makespan, task response time, and task rejection ratio.

The meta-heuristic algorithms are categorized into evolutionary-based like Genetic Algorithm (GA) [52, 53], bio-meta-heuristics (swarm Intelligence-based), and non-bio-metaheuristics like Simulation-Based Optimization (SBO) and Simulated Annealing (SA) [57]. Swarm Intelligence (SI) is a sub-domain of computational intelligence and first used this concept by [98]. The aim of SI is to solve computational problems by modeling self-organized populations of agents that can interact with each other. Agents can share their experiences by exchanging information. The interactions and movements of agents represent the population performance [99].

5.1.1 Swarm Intelligence

SI was first used by [98] for robotic intelligence in cellular robotic systems. After that, the definition of SI is expanded by [100] for algorithms and solving distributed problems. Swarm intelligence algorithms are broadly categorized into two sub-categories [41] i.e., 1) Sign based SI include Ant Colony Optimization (ACO) [47, 48] and Bee Colony Optimization (BCO)algorithms, 2)Imitation based SI algorithms comprise of Raven Roosting Optimization (RRO) [49] algorithms, Improved Raven Roosting Optimization (IRRO) [41], Cat Swarm Optimization (CSO) [50], Chicken Swam optimization (CSO) [41, 51], and Particle Swarm Optimization (PSO) [7, 26, 54, 55] algorithms among others.

The most prominent meta-heuristic based task scheduling algorithms include Ant Colony Optimization (ACO) [47, 48], Genetic Algorithm (GA) [52, 53], Particle

Swarm Optimization (PSO) [7, 26, 54, 55], honeybee foraging [56, 101], and Simulated Annealing (SA) [57].

ACO based meta-heuristic algorithms have better optimization at early stages, however, the convergence rate of ACO is comparatively slower. As compared to GA, PSO has an easy implementation, fast convergence, and better optimization performance [81]. PSO variants have better performance in terms of elapse time, approximation set, and hyper volume as compared to different variants of GA algorithm like SPEA2 and NSGA [102]. PSO is more popular among these meta-heuristic based algorithms due to its effectiveness for a broad range of applications, simplicity, fast convergence, and easy implementation [103]. Moreover, PSO based algorithms have a sound natural computation background along with better performance.

Computational time is the most important factor in task scheduling in the cloud. Its because cloud is a dynamic computing environment and the cloud scheduling algorithm should be fast enough to be adopted in the real cloud environment. Moreover, these algorithms should provide an optimized solution with fast convergence. The literature study shows that PSO is the most adopted optimization algorithm [99] for cloud task scheduling. Therefore, this research focuses on optimizing tasks execution time (makespan), Average Resource Utilization Ratio (ARUR), reduce task response time, minimize tasks rejection, penalty cost [26], and total tasks execution cost [26] of the cloud [63] using PSO.

5.1.2 Particle Swarm Optimization

PSO [7] can be applied to both discrete and continuous problems and is more efficient for global search in the problem space. PSO converges globally and tries to find comparatively better fitness value. However, PSO is weak for local search and cannot pay more attention to the search in the local subspace [26, 54, 55, 103]. This increases the chances of trapping to the local optima and may have a lower convergence rate in later stages.

To overcome these limitations of PSO algorithms, inertia weights play a significant role. Inertia weight is an important control parameter for effectively adjusting the

local and global search capability of the PSO algorithm. A low value of inertia weight facilitates local search while a high value of weight facilitates the global search. The literature study shows that there are five prominent and most commonly used inertia weights strategies out of a total of 15 inertia weights [82, 103]. In this chapter, a novel and adaptive inertia weight strategy has been proposed [104] which overcome the limitations of existing inertia weight strategies that results in an improved version of PSO and is termed as adaptive PSO.

In cloud computing, profit maximization is the key objective of cloud Service Providers (CSP). CSP profit can only be maximized by higher utilization of cloud resources and balancing the workload. On the Client side, minimization of the tasks execution cost, reduction in tasks execution time, and meeting tasks deadline are the key factors for the cloud users. Most of the meta-heuristic based task scheduling algorithms are single objective or bi-objective and the majority of these algorithms consider non-conflicting parameters like makespan, throughput, and response time, etc. for performance evaluations.

The main objective of this research is to investigate and analyze the multi-objective optimization problems, proposed a task scheduling framework that maps the user workload in a balanced manner, and improve the quality of service parameters like makespan, Average Resource Utilization Ratio (ARUR), reduce task response time, minimize tasks rejection, penalty cost, and total execution cost of tasks.

In this research, a *Particle Swarm Optimization based Resource and Deadline Aware dynamic Load-balancer* (PSO-RADL) for the deadline-constrained task has been proposed. PSO-RADL is a multi-objective-based task scheduling scheme. The proposed approach considered conflicting parameters like task deadline, task execution time, and monetary cost. The PSO-RADL scheduling technique has the ability to reduce the task penalty cost and task execution cost. Moreover, a novel Inertia weight strategy i.e., *Leaner Descending and Adaptive Inertia Weight* (LDAIW) for PSO-based algorithms have been proposed. The proposed inertia weight strategy improves the performance of the PSO algorithm. Its because the proposed approach provides a better combination of local and global search. In summary, the major contributions of this research are:

- In-depth critical investigation of state-of-the-art heuristics and meta-heuristics

based task scheduling algorithms to identify the strengths and limitations of these approaches.

- A Particle Swam Optimization (PSO) based novel dynamic load-balancing scheduler for non-preemptive, independent, and compute-intensive tasks-based cloud workload that produces lower task execution time, improved resource utilization, reduced task rejection, and minimized tasks response time.
- Multi-objective based task mapping framework for conflicting parameters like task deadline, task response time, makespan, and cost.
- PSO-RADL scheduling scheme for minimizing the task penalty cost and execution cost of cloud datacenter.
- A novel inertia weight strategy named *Leaner Descending and Adaptive Inertia Weight* (LDAIW) has proposed that improves the performance of PSO based algorithms.
- Empirical investigation and performance evaluation of the proposed scheduling approach against state-of-the-art scheduling heuristics.

The rest of the chapter is organized as follows. Section 5.1 discusses the introduction part. PSO-RADL system architecture, PSO-RADL algorithm, PSO-RADL system, and performance model is presented in section 5.2. Section 5.3 presents experimental evaluation and discussion that include experimental setup, workload generation, and simulation results. Results and discussion are discussed in section 5.4. Section 5.5 concludes the chapter and presents potential future directions.

5.2 PSO-RADL System Overview and Background

The literature study reveals that most of the existing scheduling algorithms suffer from issues like under-resourced utilization, load imbalance, high makespan, and

task rejection ratio. Moreover, these algorithms are not considering conflicting parameters like execution time and cost. PSO-RADL is a Particle Swarm Optimization based resource-aware dynamic load-balancing scheduler for deadline-constrained tasks. The PSO-RADL follow a single point based encoding scheme i.e., the proposed approach generate a single best solution for every generation. PSO-RADL scheduling scheme implemented and evaluated using well-known Cloudsim [11] simulator. Cloudlet is used as a synonym for the task in the Cloudsim simulator. The datacenter entity of Cloudsim is used to simulate cloud infrastructure level services. The datacenter comprises hosts machines and storage servers, and the computation power of Host Machines (HM) and storage servers on that datacenter represents the capacity of the datacenter. *Cloud Service Provider* (CSP) creates One or more Virtual Machines (VM) on every HM based on a defined VMs allocation policy. PSO-RADL scheduling scheme assigns tasks to VMs based on multi-objective based criteria having conflicting parameters like task execution time, penalty, and total execution cost.

5.2.1 PSO-RADL System Architecture

PSO-RADL is a particle swarm optimization-based resource and deadline-aware cloud task scheduling technique that allocates the incoming task in a balanced way. The system architecture of the proposed model is shown in Figure 5.1. The Symbols section shows some of the terminologies and notations used in the PSO-RADL task scheduling scheme.

1. Input and output: PSO-RDAL Scheduler receives a list of tasks with their computation requirements in Million Instructions (MIs) along with task deadline (in Milli-Seconds (MS)). A set of Virtual Machine (VM) with their processing capacity in Million Instructions Per Second (MIPS) is received as an input parameter (shown in Figure 5.1). VMs set shows the actual capacity of cloud datacenter. PSO-RDAL will give an optimized mapping of tasks to VMs as an output.

2. Initialization step: In this step, population agents (particles) are initialized to random positions in the search space. At the initialization phase of the search process, tasks are randomly mapped to VMs and local and global best values are

computed based on this mapping which will be used in the future for comparison. The velocity and position of particles are initialized at random values. Table 5.1 presents some important parameters that need to be initialized before starting the execution of the proposed approach. The c1 and c2 are the acceleration factors and based on the literature study the values of these factors are initialized to 2 and 1.49455 respectively [82]. Moreover, the initial value of inertia weight (w1) is set 0.4 [82, 103] and the final value of inertia weight (w2) is set 0.9 based on literatures [7, 82, 103]. The upper and lower bound is set to the number-of-VMs -1 and zero respectively.

TABLE 5.1: Initialization parameters

Parameters	**Values**
Number of iterations	200
Number of particles	20
Minimum position	0
Maximum position	No of VMs -1
w1	0.4 [82, 103]
w2	0.9 [7, 82, 103]
Acceleration factor c1	2 [82]
Acceleration factor c2	1.49455 [82]
Stopping Criteria	MaxItr

3. The number of iterations also known as generations is represented by Itr and the maxItr shows the upper bound i.e., the maximum number of iterations. The maximum number of iterations maxItr is set to 200 after comprehensive fine-tuning.

4. The working procedure of the swarm intelligence-based algorithm comprises two phases i.e., global and local search. To find an optimized solution, a good balance between local and global search plays a vital role. In an ideal scenario, the adoption of a global search operator should be greater than the local at the beginning of the search procedure [103]. Inertia weight strategy is considered

a strong control parameter for maintaining a balance between global and local search [82]. A number of inertia weight strategies have been proposed by various researchers. However, in this chapter, we are using the novel and adaptive inertia weight strategy that is proposed in [104].

5. The number of iterations (discussed in step 3) represents swarm generations and each generation has a number of particles. Each particle represents a single solution i.e., mapping of tasks to VMs.

6. Every particle shows a complete solution that comprises a number of tasks and VMs. Velocity and position are computed for each cloudlet to be mapped on VM. In case the position value becomes negative or greater than VMCount-1 then Position (P) is re-initialized to a random value between zero VMCount - 1. In case the value of P is within the range then VM is selected and task completion time (TRT) is computed. VM ready time and particle map are updated and this process repeats for every cloudlet.

7. When all tasks are mapped to VM then local best is computed and is updated if needed. Similarly, the global best value is compared with the recent local best and is replaced if the new value is better than the current global best value and this process repeats for each particle.

In case all particles are executed then a new iteration is started. The mapping process is completed which Itr reached to maxItr.

5.2.2 PSO-RADL Algorithm

This section discusses the proposed scheduling algorithm PSO-RADL. PSO-RADL algorithm (Algorithm 1) receives a list of cloudlets (tasks) with their sizes in Million Instruction (MI) and task deadlines (in seconds) and a list of virtual machines with their computation power in Million Instructions Per Second (MIPS) as input. The output of the PSO-RADL (Algorithm 1) consists of a global best value-based final task for VM mapping.

Lines 1-8 comprise of necessary initialization of the PSO-RADL algorithm. VMMap and CloudletMap are converted into the VMList and CloudletList (Lines 9-10, Algorithm 5). The number of VMs and Cloudlets are identified using VMList and

CloudletList (Lines 11-12, Algorithm 5). The initialization of the particles is performed at line 13 (Algorithm 5) and the result of the initialization step is stored in hashMap named pbMap.

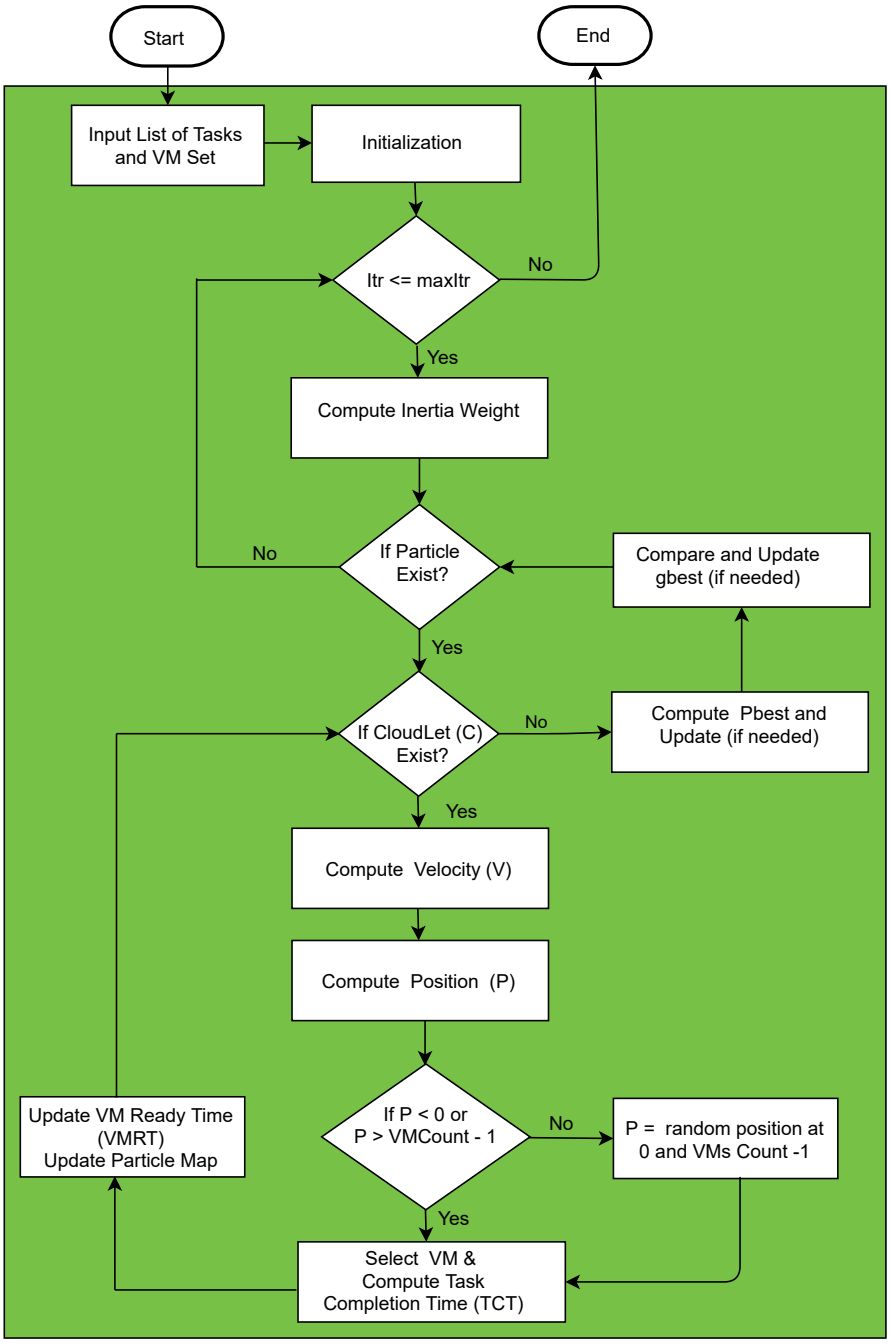


FIGURE 5.1: PSO-RADL Scheduler

The while loop (Lines 14-41, Algorithm 5) is executed until the number of iterations (Itr) reaches their maximum level i.e., MaxItr and Itr are incremented by one at the end of each iteration (Line 40, Algorithm 5). Inertia weight is computed

for every iteration (Line 15, Algorithm 5) using Eq. (5.3). The for loop (Lines 16-39, Algorithm 5) is repeated for all particles. The size of the particle varies from case to case however, the number 20 is fixed for the number of particles after comprehensive fine-tuning. The nested for loop (Lines 17-30, Algorithm 5) iterates as many times as the number of cloudlets in the received cloudlet list.
For every iteration of the nested loop, particle velocity (v) and position (pos) is updated (Lines 20-21, Algorithm 5) using Eq. (5.4) and 5.5. *r1* and *r2* are two random numbers ($0 \leq$ r1,r2 ≤ 1) which are regenerated for every update of velocity (Lines 18, Algorithm 5). Constant acceleration factors *c1* and *c2* are known as cognitive and social components of particle velocity (Lines 19, Algorithm 5). The cognitive coefficient *c1* limits the length of the swarm step that particle takes toward their personal best and the social coefficient factor limits the size of the step taken by the particle towards the global best. The values of *c1* and *c2* are set to 2 and 1.49455 respectively [82]. Condition at line 22 restricts the position value within their boundaries. if the position value is less than zero or greater than the total number of VMs, a random value position is generated between 0 and VMCout-1 (Lines 22-24, Algorithm 5).
At line 25 (Algorithm 5), VM is identified and execution time (i.e., exectime) is computed for the current cloudlet on the selected VM (Line 26, Algorithm 5) using Eq. (3.3). Task completion time (TCT) is computed by adding task execution time with VM ready time (Line 27, Algorithm 5) using Eq. (5.1). VM ready time and particle hashmap pMap are updated (Lines 28-29, Algorithm 5). Personal best value (i.e., pBestValue) is computed and compared with the previous pBestValue (Line 31-32, Algorithm 5). if the condition at line 32 (Algorithm 5) is true then the pBestValue is updated in the pbMap i.e., the personal best hashMap that stores particles' best value (Line 33, Algorithm 5).

The nested if statement (Lines 34-37, Algorithm 5) is executed when the condition at line 34 is true i.e. when the updated pBestValue is greater (improved) than the global best value (gBestValue). The new improved value of pBestValue is assigned to gBestValue (Line 35, Algorithm 5) and pMap is updated in gbFMap which stores the global best based mapping of tasks to VMs (Line 36, Algorithm 5).

Algorithm 5: Proposed PSO-RADL scheduler

Input : CloudletMap: List of tasks (cloudlets) with their length in MI and VMMap: List of VMs with their processing capability in MIPS

Output: gbFMap: global best based final mapping of tasks to VMs

```
pos = 0, w = 0.0, pBestValue =0.0
v = RandomNo(0, 1), gBestValue =0.0
noParticles = 20, w1 = 0.4, w2 = 0.9
Itr = 1, MaxItr = 200, c1 = 2, c2 = 1.49455
pbMap<Integer, Double> = Null
VMRTMap <Integer, Double> = Null
pMap<Cloudlet, Vm> = Null
particlesMap<Integer, pMap<Cloudlet, Vm>> = Null
VMList = getVMs(VMMap)
CloudletList = getCloudlets(CloudletMap)
CloudletCount = CloudletList.size()
VMCount= VMList.size()
pbMap = initializeParticles(CloudletCount, VMCount, pbMap, pMap,
  gbFMap, noParticles)
while (Itr <= MaxItr) do
    w = ((w1 - w2)/Pbs)+((MaxItr - Itr)/Itr)*(w1 - (w1 - w2)/Pbs)
    for (p = 1 to noParticles) do
        for (c = 1 to CloudletCount) do
            r1 = RandNo(0, 1)
            r2 = RandNo(0, 1)
            v = (w * v) + (c1 * r1 * (pbMap.get(p)- pos)) + (c2 * r2 *
              (gBValue - pos))
            pos = pos + v
            if pos >= VMCount || pos < 0 then
                pos = RandPosition(0, VMCount-1)
            end
            Vm VM = getVM(pMap);
            execTime = getCETime(CloudletList, clt, VM)
            TCT = execTime + getVMReadyTime(VMRTMap, VM, clt)
            updateVMRTMap(VMRTMap, VM, clt, execTime)
            updatepMap(VMRTMap, pMap, pos, clt);
        end
        pBestValue = getpbMap(VMRTMap)
        if (pBestValue > pbMap.get(p)) then
            pbMap.put(p, pBestValue)
            if pBestValue > gBestValue then
                gBestValue = pBestValue
                gbFMap.put(0, pMap)
            end
        end
    end
    Itr++
end
```

5.2.3 PSO-RADL System Model

This section presents the System model of the PSO-RADL scheduling technique. Some basics definitions, notations, terminologies used in the system architecture, and mathematical expressions of the proposed approach are shown in the Abbreviations and Symbols sections. The Cloud datacenter consists of a list of VMs (shown in Eq. (3.1)) that represents the size of the cloud. Task Completion Time (TCT) is the expected completion time of the task on a particular virtual machine (shown in Eq. (5.1)) and mathematically expressed as:

$$\text{TCT} = \text{execTime} + \text{VMRT} \tag{5.1}$$

Task Completion time of task on VM is the sum of the execution time of the current task and the completion time of already assigned tasks to that VM. The *execTime* is the execution time of a task on the VM and VMRT is the VM ready time. The *execTime* is computed using Equation (3.3)
The size of the task is shown in million instructions (MI) and MIPS shows the computation capacity of VM. The VM ready time is computed using Eq. (5.2)

$$\text{VMRT} = \sum_{i=1}^{k} \text{execTime}_i \tag{5.2}$$

where, k is the total number of tasks assigned to the VM. Inertia weight plays an important role in swarm intelligence based meta-heuristic algorithm. W represents inertia weight and is computed using Eq. (5.3) [104].

$$\text{w} = \frac{((\text{w}_1 \text{ - } \text{w}_2)}{\text{Pb}_s} + (\frac{(\text{MaxItr-Itr})}{\text{MaxItr}})) * (\frac{(\text{w}_2 \text{ - } (\text{w}_2 \text{ - } \text{w}_1)}{\text{Pb}_s}) \tag{5.3}$$

Where w_1 is the maximum value and w_2 is the minimum value which is set to 0.9 and 0.4 respectively. MaxItr represents the total number of iterations and Itr shows the current iteration. The Pb_s shows personal best status which is used as feedback to adjust inertia weight. Maintaining velocity is one of the major

activities of PSO based algorithms. PSO algorithm update particle velocity using Eq. (5.4) [82, 103].

$$v_{pn}^{g+1} = w * v_{pn}^{g} + c_1 r_1 (Pb_{pn}^{g} - x_{pn}^{g} + c_2 r_2 (Gb_{pn}^{g} - x_{pn}^{g}) \tag{5.4}$$

Where p shows the number of particles and p = 1, 2, 3, ..., P, g represents the number of iterations g = 1,2,3, ..., MaxItr, and n represent the number of dimensions i.e., the number of tasks that need to be mapped on VMs in an optimal manner. V_{pn}^{g} represents current velocity of the particle and X_{pn}^{g} represent the current position of p^{th} particle with g^{th} iteration in n dimensional space. W shows the inertia weight, *c1* and *c2* are the constant acceleration factors, and *r1* and *r2* are randomly generated values between 0 and 1.
The position of particle is updated using Eq. (5.5) [82, 103].

$$x_{pn}^{g+1} = (v_{pn}^{g+1} + x_{pn}^{g}) \tag{5.5}$$

To identify the fitness of the particle, every solution of PSO-based algorithms is evaluated through its objective function. The objective function can be a maximization or minimization problem. The objective function of the PSO-RADL scheduler is based on the maximization problem and mathematically expressed using Eq. (5.6).

$$\text{Objective Function} = \max((\text{ARUR} + 1/\text{makespan} + 1/\text{rtList})/3) \tag{5.6}$$

VM completion time VM_CT is the sum of completion time of all cloudlets which are allocated to the VM for execution and is computed by using Eq. (3.7) [1]. Task Response time (RspTime) is the difference of task execution start time and task arrival time and is computed using Eq. (3.8).

Cost is one of the most important evaluation parameters for cloud scheduling algorithm. Penalty cost (penalityCost) is the multiple of delay time units and penalty rate (shown in Eq. (5.7))[76].

$$\text{penaltyCost} = \text{delayTime} \times \text{penaltyRate} \tag{5.7}$$

The penalty rate is the cost per unit of delay time. Delay time is task execution time (in seconds) that exceeds the task deadline. Deadline violation occurs when the execution time of a task exceeds its deadline and is measured in Seconds.

$$\text{TaskExecutionCost} = \sum_{j=1}^{m} \text{VMCost}_j \tag{5.8}$$

Task execution cost is the sum of costs for all VMs. The Eq. (5.8) shows task execution cost where m represents the total number of VMs and VMCost_j represents the cost of VM_j [76].

5.2.4 PSO-RADL Performance Model

Performance evaluation of the proposed technique PSO-RADL is performed against the state-of-the-art task scheduling approaches in terms of Makespan (mathematically represented using Eq. (3.9) [1, 23, 34, 59, 62]), Average Resource Utilization Ratio (ARUR) which is computed using Eq. (3.10) [1, 19], tasks response time (computed using Eq. (3.12), percentage of task rejection which computed using Eq. (3.11), penalty cost (computed using (5.9), and total cost (computed using (5.10) (penalty plus task execution cost).

Total penalty cost is the sum of penalty cost of all tasks whose deadline violated [76] and is computed using Eq. (5.9)

$$\text{TotalPCost} = \sum_{i=1}^{n_{dm}} \text{penaltyCost}_i \tag{5.9}$$

where n_{dm} represent the number of tasks whose deadline missed. Reduced total penalty cost represents higher performance in terms of maximum tasks meeting their deadlines. The total cost of cloud is the sum of the total penalty cost and task execution cost and is computed using Eq. (5.10) [76]. Minimized total cost represents better performance in terms of reduced execution expense of user job.

$$\text{TotalCost} = \text{TotalPCost} + \text{TaskExecutionCost} \tag{5.10}$$

5.3 Experimental Evaluation and Discussions

This section describes the experimental evaluation and discussion of the proposed task scheduling technique PSO-RADL and state-of-the-art task scheduling technique in cloud computing. The PSO-RADL scheduling scheme is an extension of the RADL Scheduler and the same experimental setup and workload generation criteria has been used which are discussed in section 3.3.1 and section 3.3.2 respectively.

5.3.1 Simulation Results

This Section discuss the simulation results of the proposed task scheduling technique PSO-RADL and comparison of these results against state-of-the-art scheduling heuristics. These heuristics include PSSELB [19] RALBA [1], Dynamic MaxMin (D-MaxMin)[33], DC-DLBA [34], DLBA [35], and MODE [76].

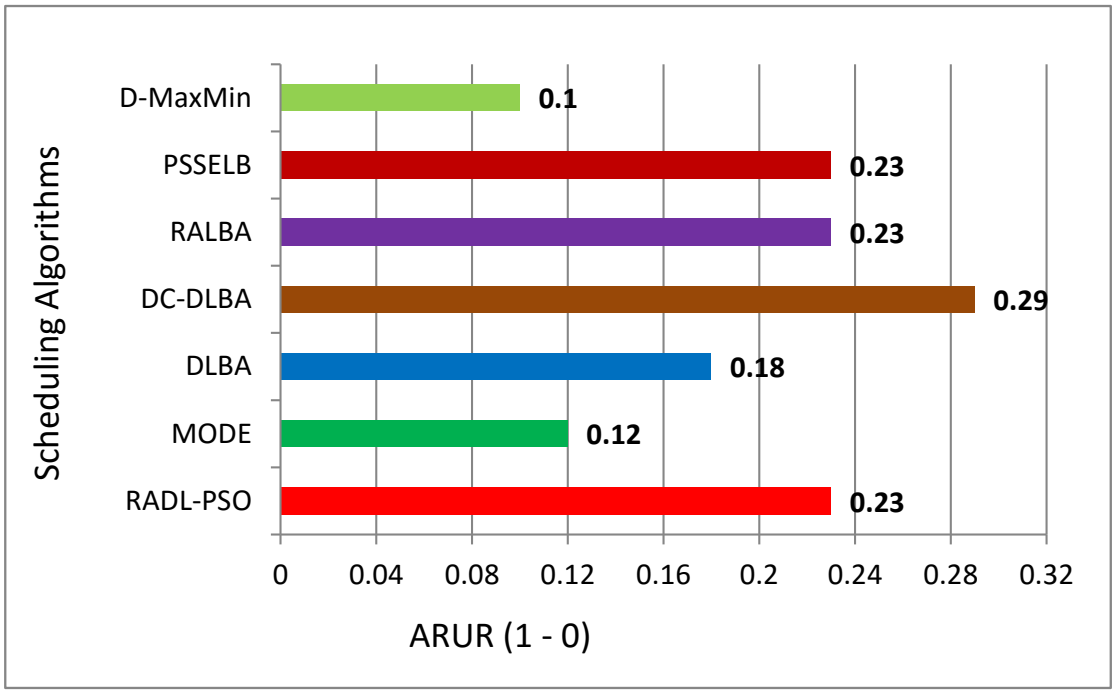


FIGURE 5.2: ARUR results for Synthetic dataset

The performance of the proposed technique and state-of-the-art task scheduling heuristics is evaluated using metrics like Mean ARUR [1, 19, 33], makespan [1, 19,

34, 35], average response time [19, 33, 35], %age of task rejection[34], penalty cost [76], total cost[76] of the cloud.

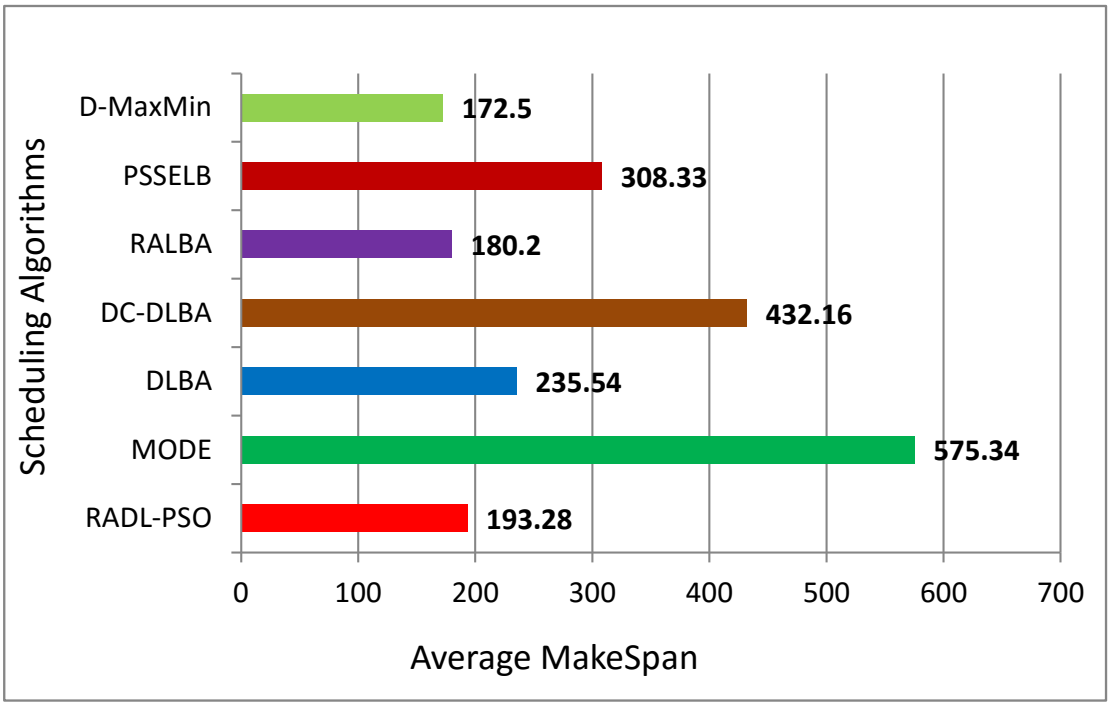


FIGURE 5.3: Makespan results for Synthetic dataset

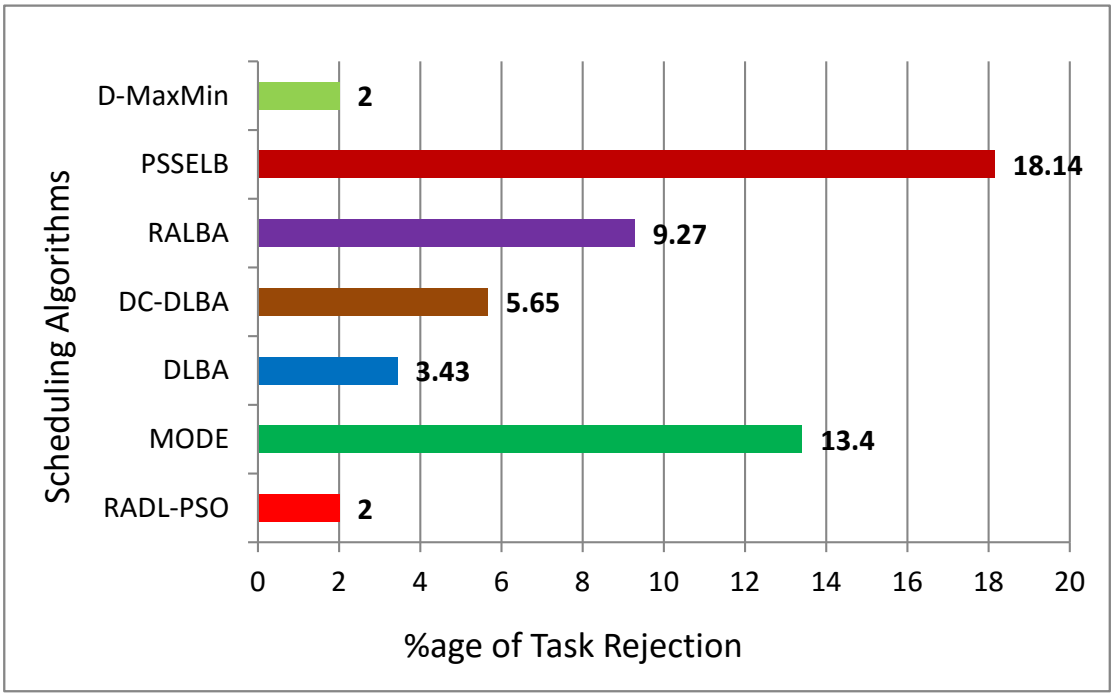


FIGURE 5.4: Task rejection results for Synthetic dataset based executions

Figure 5.2 shows ARUR based experimental results using Synthetic workload [1]

dataset. Synthetic workload-based benchmark datasets comprise a large number of smaller size tasks and only a few larger size tasks which are termed as a positively skewed dataset.

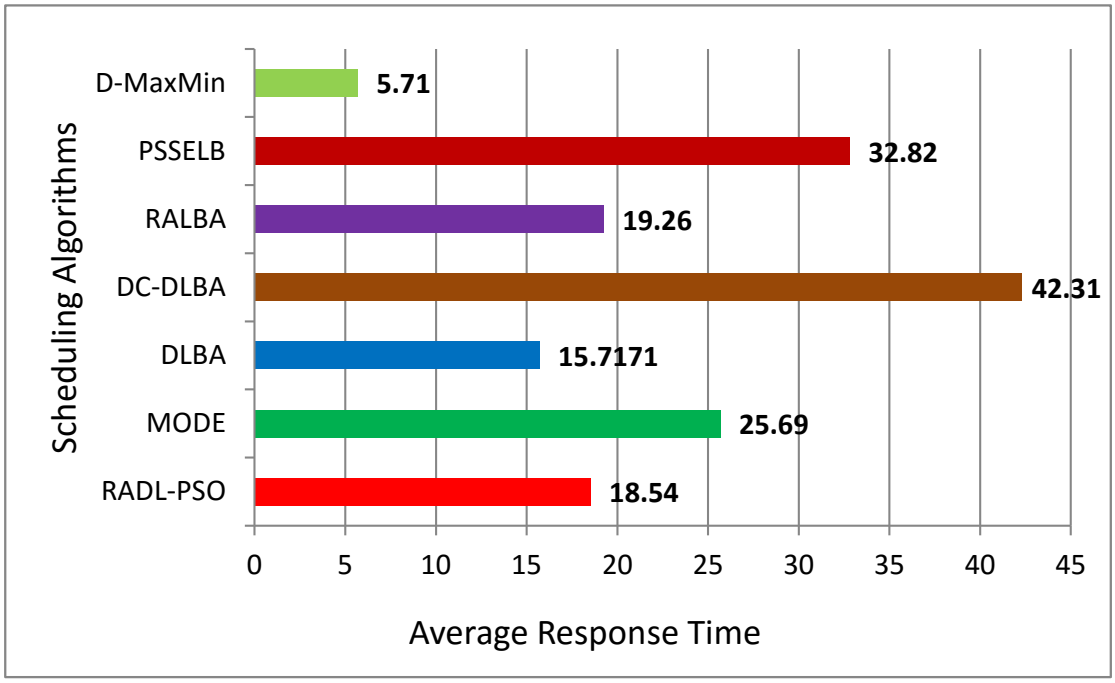


FIGURE 5.5: Task Response Time results for Synthetic dataset based executions

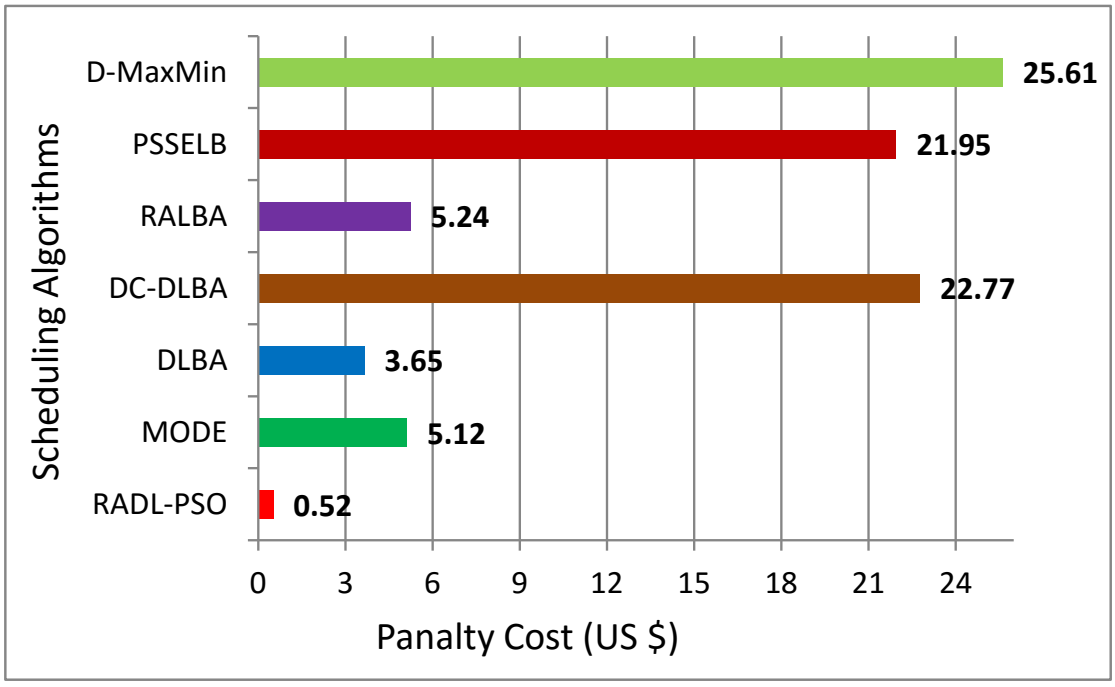


FIGURE 5.6: Penalty cost results for Synthetic dataset based executions

The experimental results presented in figure 5.2 shows that the proposed PSO-

RADL approach attained 92%, 28%, -21%, 0%, 0%, and 130% higher resource utilization as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1] and PSSELB [19], and D-MaxMin [33], respectively. Figure 5.3 reveals that the proposed technique PSO-RADL has attained 66%, 18%, 55%, -7%, 37%, and -12% reduced makespan using synthetic workload benchmark dataset against MODE [76], DLBA [35], DC-DLBA [34], RALBA [1] and PSSELB [19], and D-MaxMin [33], respectively.

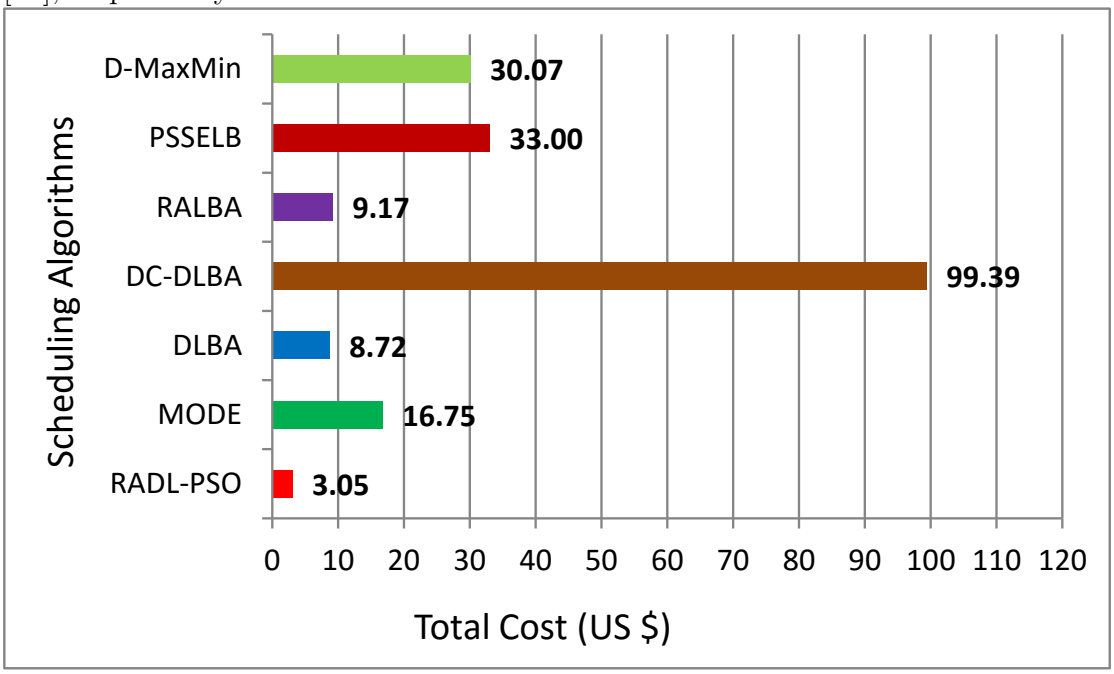


FIGURE 5.7: Total Cost results for Synthetic dataset based executions

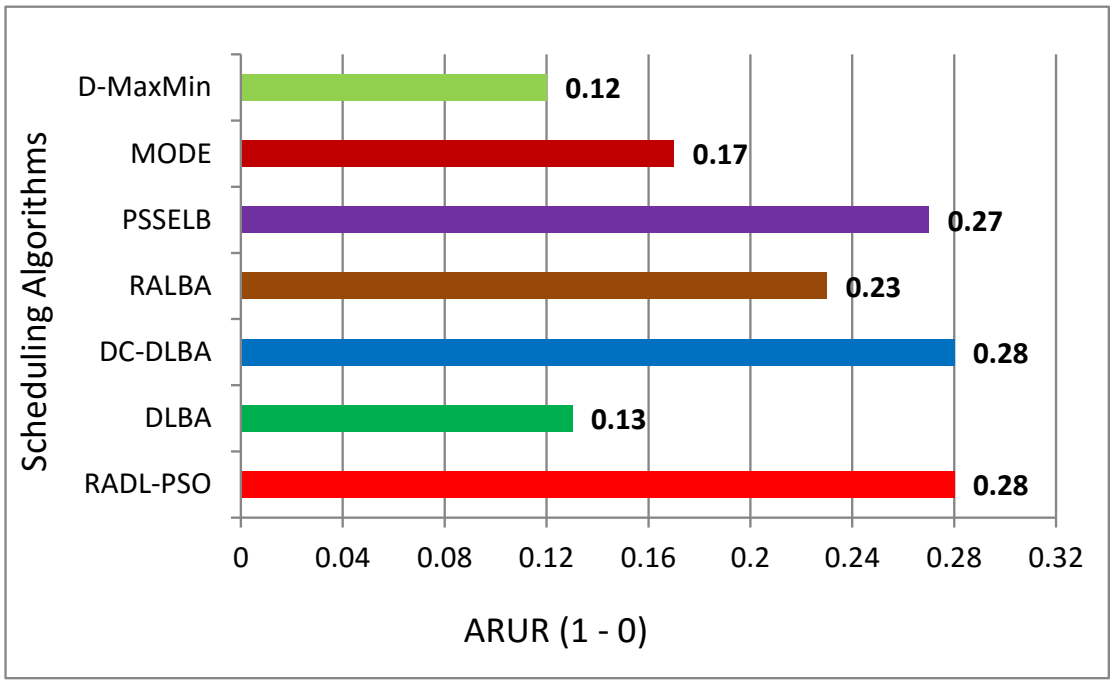


FIGURE 5.8: ARUR results for GoCJ workload-based dataset

Experimental results presented in figure 5.4 shows that the proposed technique has attained 85%, 42%, 65%, 78%, 89% lower task rejection and 28%, -18%, 56%, 4%, 44% minimized response time (as shown in Figure 5.5) using Synthetic workload benchmark dataset as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1] and PSSELB [19], respectively. Figure 5.6 shows total penalty cost based results on Synthetic workload dataset.

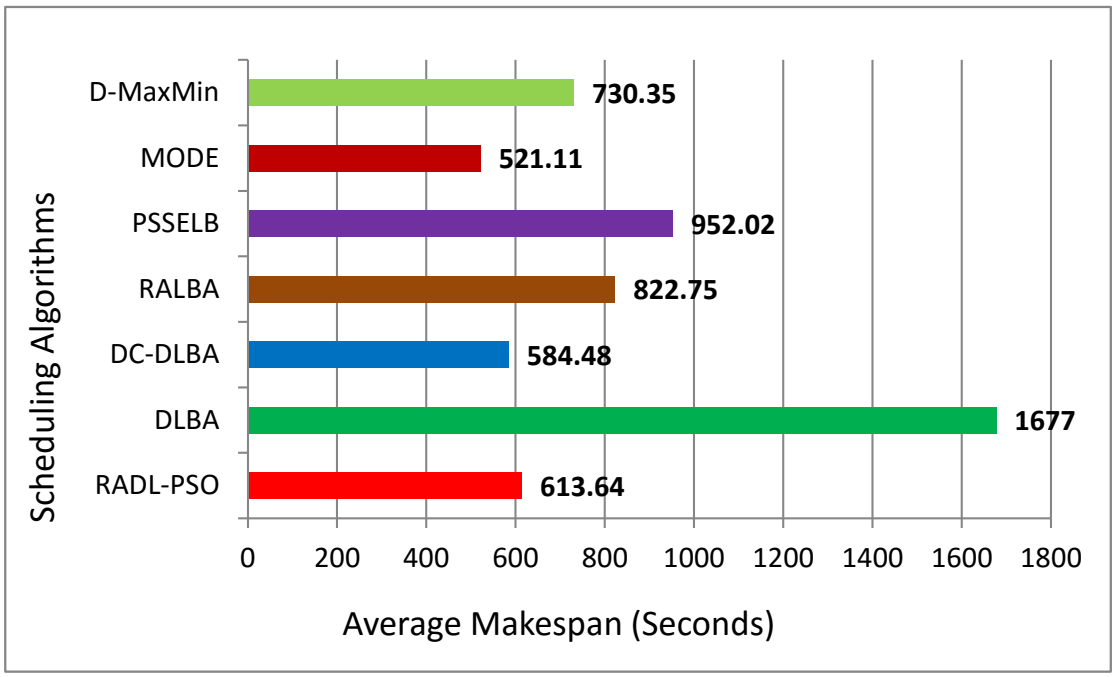


FIGURE 5.9: Makespan results for GoCJ dataset based executions

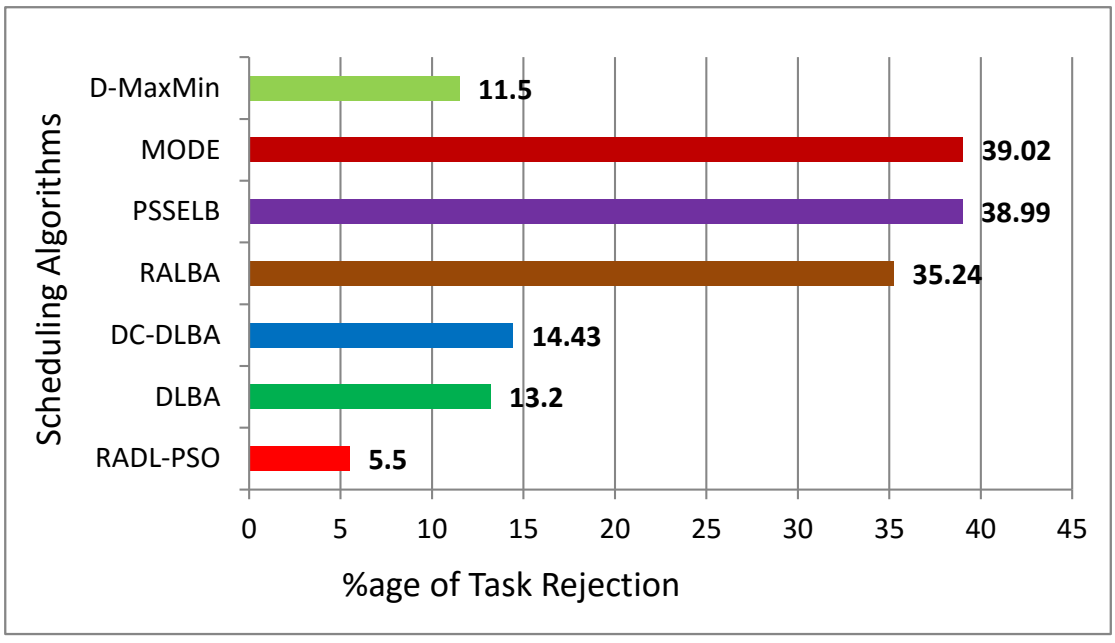


FIGURE 5.10: Task rejection results for GoCJ workload-based dataset

These results reveals that the proposed approach PSO-RADL has gained 90%, 86%, 98%, 90%, 98%, and 98% lower total penalty cost and 82%, 65%, 97%, 67%, 91%, and 90% Lower total cost of executing user tasks as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1] and PSSELB [19], and D-MaxMin [33], respectively.

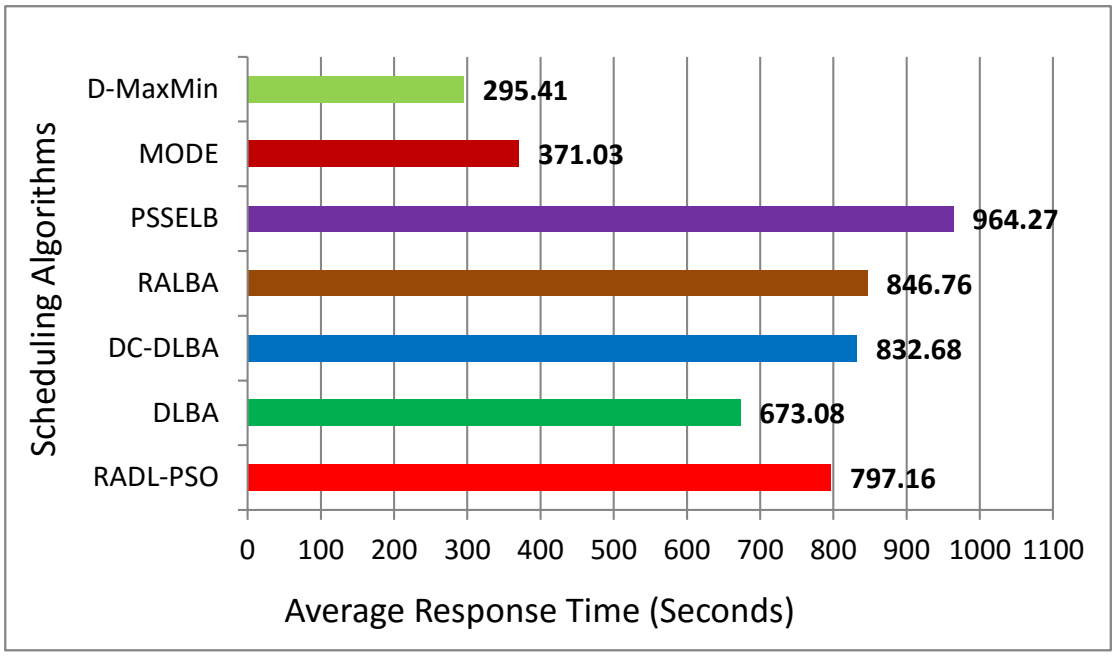


FIGURE 5.11: Task Response Time results for GoCJ dataset

GOCJ [59] is the second benchmark dataset used for performance evaluation and comparison of the proposed approach PSO-RADL. In the GOCJ dataset most of the tasks are of larger size than smaller size tasks. Experimental results evince that the PSO-ADL has attained 77%,-18%, 0%,-15%, 35%, 92% higher ARUR (as presented in Figure 5.8), and 63%, -5%, 25%, 36%, -18%, and 16% lower makespan (as presented in figure 5.9) using GoCJ benchmark dataset as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and D-MaxMin [33], respectively.

Figure 5.10 shows that PSO-RADL has achieved 58%, 62%, 84%, 86%, 86%, and 52% reduced task rejection as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and D-MaxMin [33], respectively. The PSO-RADL has gained 4%, 6%, and 17% lower task response time (as shown in figure 5.11) as compared to DC-DLBA [34], RALBA [1], and PSSELB [19], respectively using

GoCJ benchmark dataset.

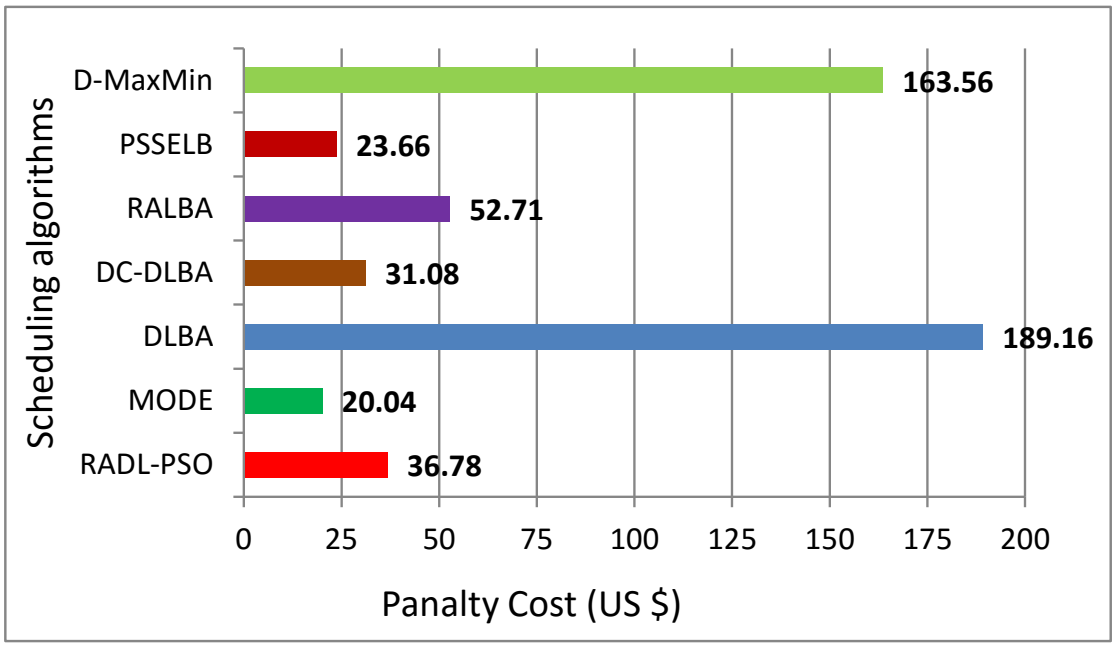


FIGURE 5.12: Penalty cost results for GoCJ dataset based executions

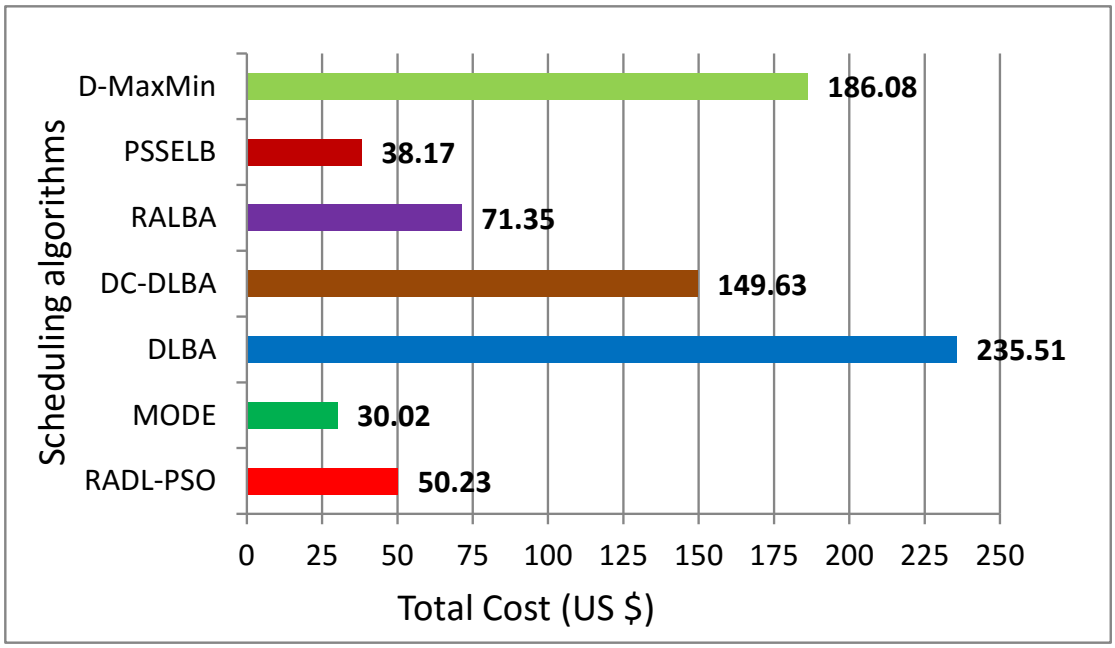


FIGURE 5.13: Total cost results for GoCJ dataset based executions

Figure 5.12 presents that the proposed approach PSO-RADL has achieved 81%, -18%, 30%, and 78% lower penalty cost and 79%, 66%, 30%, and 73% reduced total cost as compared to DLBA [35], DC-DLBA [34], RALBA [1], D-MaxMin

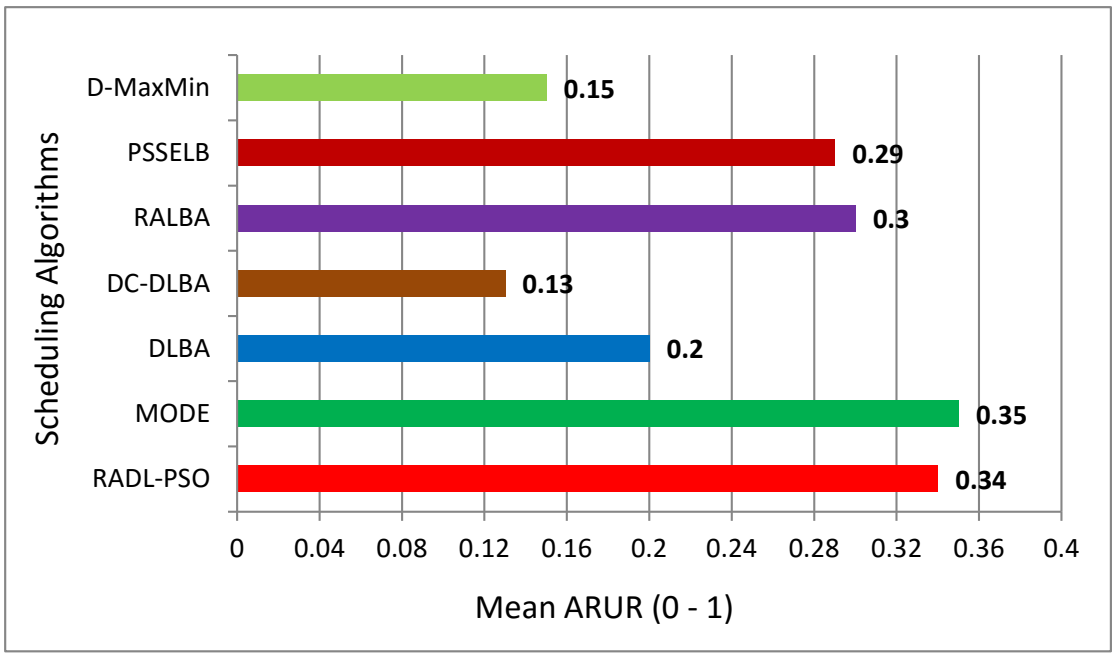


FIGURE 5.14: ARUR results for HCSP instances based dataset

[33], respectively.

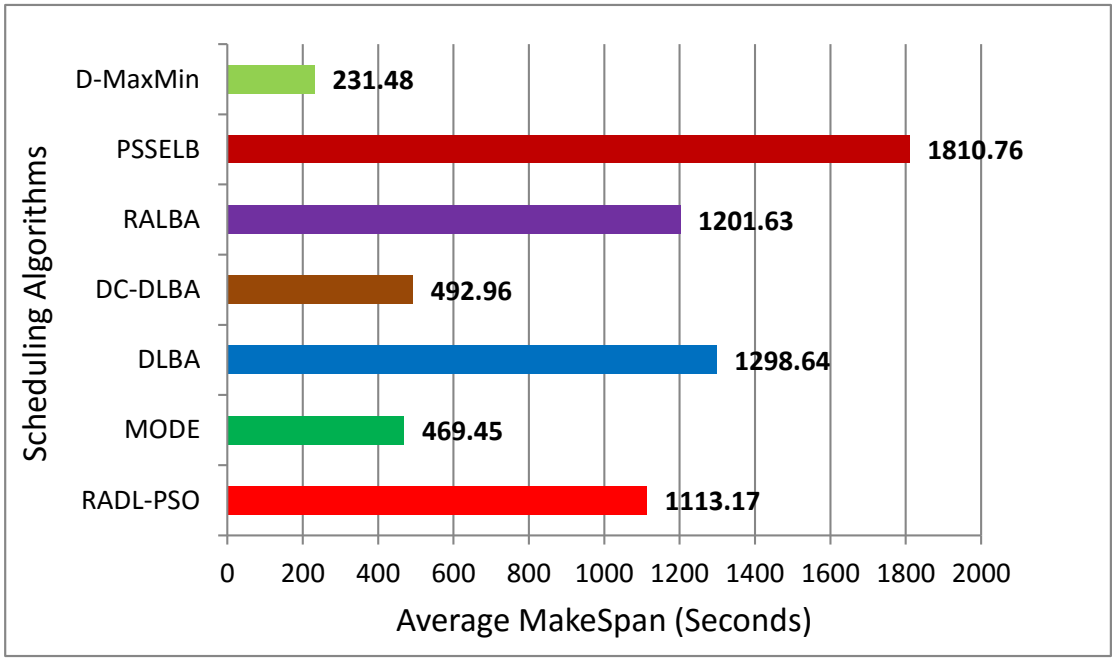


FIGURE 5.15: Makespan results for HCSP instances based dataset

Heterogeneous Computing Scheduling Problem (HCSP) [59, 62] is based on *Expected Time to Compute*(ETC) model and used as a benchmark dataset that comprise of HCSP instances. The experimental results shown in figure 5.14 re-

veals that the proposed technique PSO-RADL has gained -3%, 70%, 162%, 13%, 17%, and 127%, -137%, 14%, -126%, 7%, 39%, and -381% lower makespan as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and D-MaxMin [33], respectively using HCSP benchmark dataset.

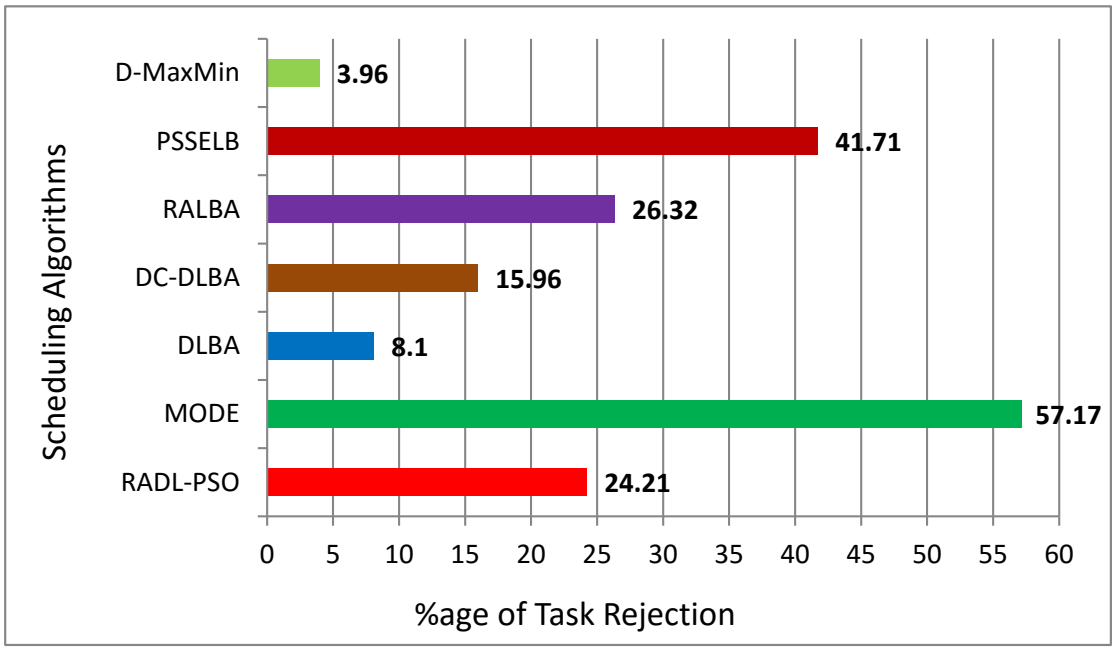


FIGURE 5.16: Task rejection results for HCSP instances based dataset

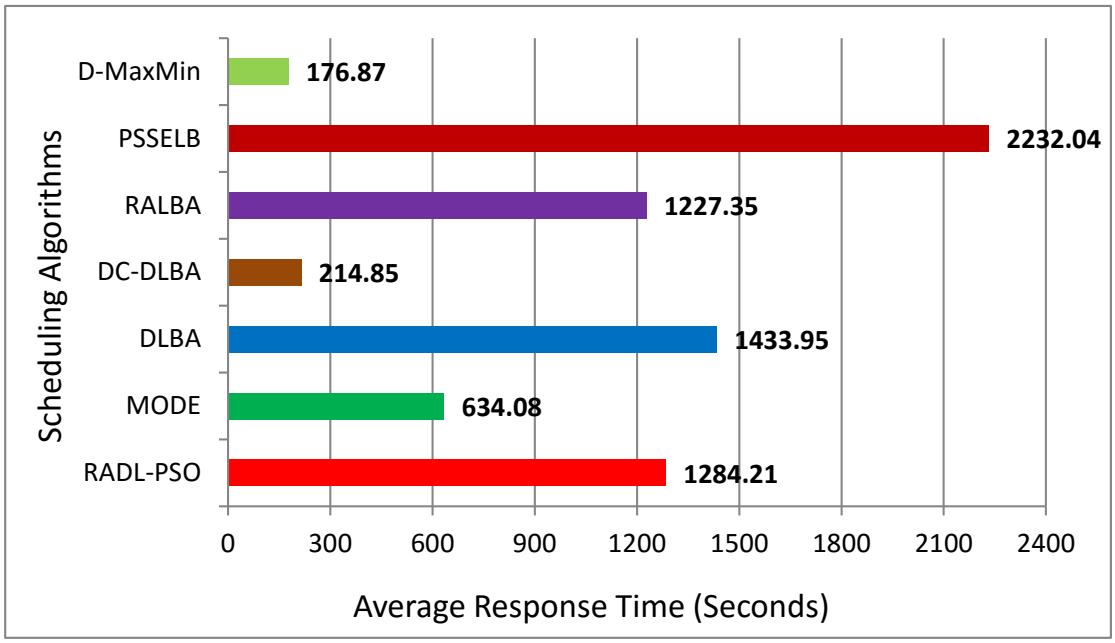


FIGURE 5.17: Task Response Time results for HCSP instances based dataset

Figure 5.16 shows that the proposed approach PSO-RADL has gained 58%, -

199%, -52%, 8%, 42%, and -511% minimized task rejection and -76%, 22%, -418%, 9%, 50%, and -529% lower response time (presented in Figure 5.17) as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and D-MaxMin [33], respectively using HCSP benchmark dataset.

Figure 5.18 shows penalty cost based experimental results using HCSP benchmark dataset. Figure 5.18 reveals the proposed approach PSO-RADL has achieved -257%, 39%, 0%, 25%, -11%, and 25% lower penalty cost and -247%, 40%, 53%, 27%, 20%, and 26% reduced total cost as compared to MODE [76], DLBA [35], DC-DLBA [34], RALBA [1], PSSELB [19], and D-MaxMin [33], respectively using HCSP benchmark dataset.

5.4 Results and Discussion

The comprehensive investigation of experimental results shows that PSO-RADL (proposed approach) has achieved significant improvements in terms of the total time of executing user job (makespan), cloud resource utilization, meeting task deadline, penalty cost, and total execution cost. Its because the proposed approach is deadline and resource-aware, and using PSO based meta-heuristic algorithm.

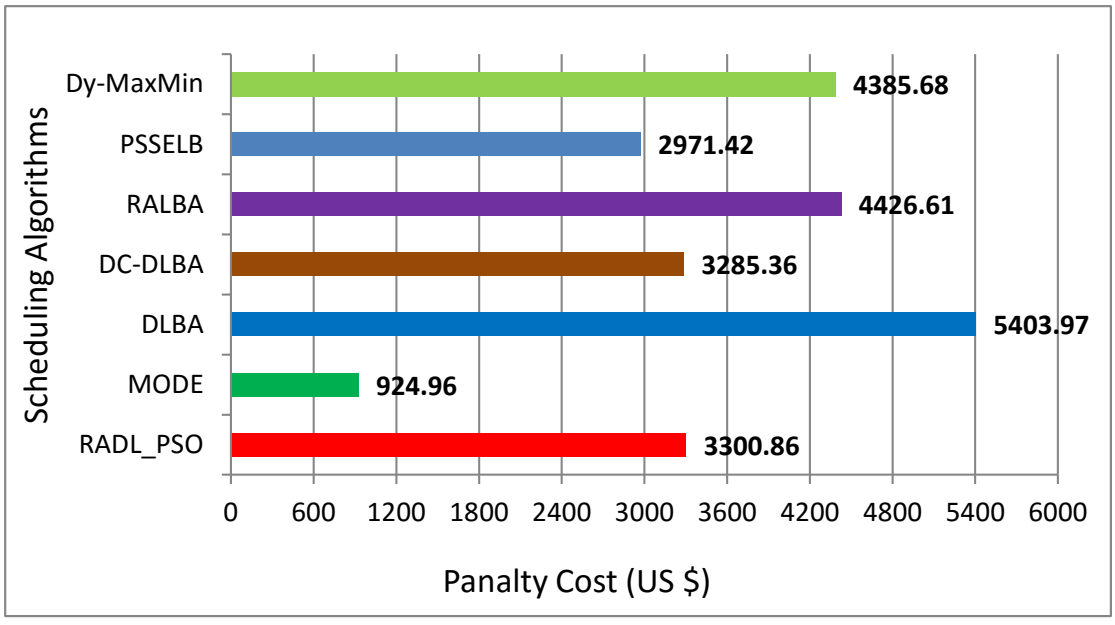


FIGURE 5.18: Penalty cost results for HCSP dataset based executions

To perform experiments and evaluate the performance of the proposed approach, three benchmark datasets have been used. The synthetic workload-based dataset is generated in [1] using a well know Monte-Carlo method. This dataset comprises a large number of small tasks as compared to large-size tasks and is termed a positively skewed dataset. The GoCJ benchmarks dataset comparatively large dataset and comprises a large number of larger size tasks. Moreover, the HCSP instances-based dataset is the largest in terms of the number of tasks. The HCSP instances used for experimentation comprise i_hilo, i_lohi, c_hilo, and c_lohi.

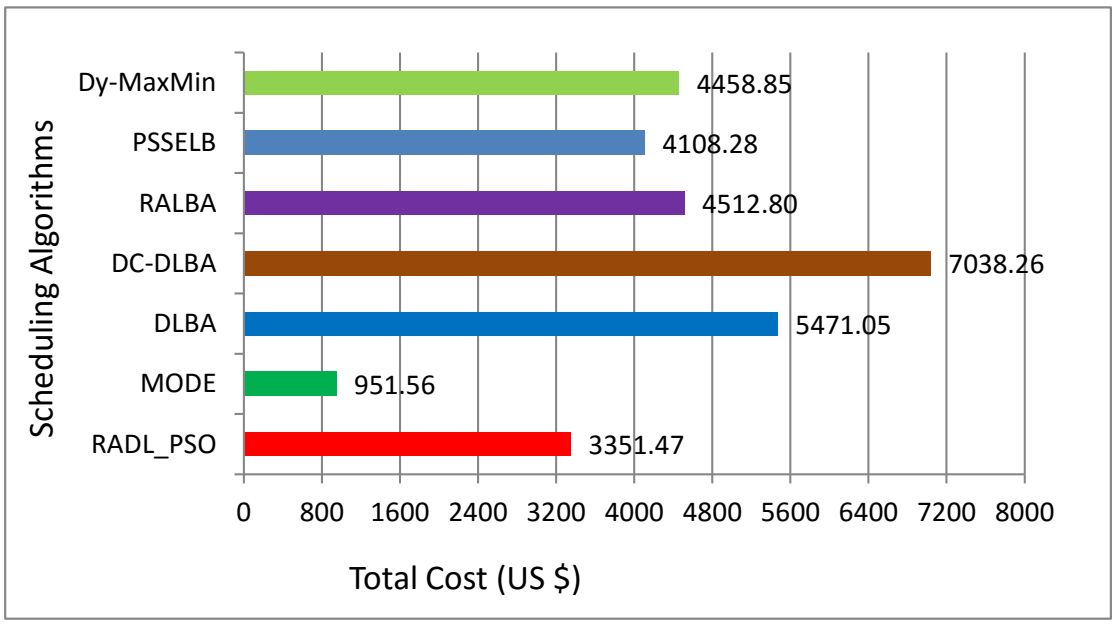


FIGURE 5.19: Total Cost results for HCSP dataset based executions

D-MaxMin has attained a lower response time on all three benchmark datasets. This is because the D-MaxMin updates task and VM status table at run time which helps to allocate tasks to VMs based on a more realistic expected completion time. However, D-MaxMin has gained lower resource utilization, high penalty, and total cost because this technique neither leases new VMs nor considers task deadline for the task to VM mapping and high violation of task deadline respectively. PSO-RADL has achieved 130%,127%, 92% higher ARUR, 98%, 78%, 25% lower penalty, and 90%, 73%, 26% reduced total cost for synthetic, GoCJ, and HCSP benchmark dataset respectively.

DC-DLBA has attained comparatively better ARUR for synthetic workload than

PSO-RADL. The reason is that DC-DLBA creates few new VMs for synthetic workload as the sizes of tasks in the synthetic workload are smaller. However, DC-DLBA shows poor performance for all other parameters like makespan, task response time, and task execution cost among others. DC-DLBA gained higher task response time, makespan, task rejection, and higher resource utilization for the execution of synthetic workload and comparatively reduced makespan for HCSP and GoCJ benchmark datasets. It is because of lower task rejection due to a large number of large-size tasks and as a result, only a few new VMs are created. The proposed approach PSO-RADL has attained 55% lower makespan, 56% reduced task response time, 65% reduced task rejection of synthetic workload dataset, and 165% higher ARUR on HCSP dataset.

MODE has attained higher makespan, task rejection, and cost for synthetic workload and GoCJ datasets and comparatively lower makespan and task rejection for the HCSP dataset. Its because this approach creates only a few new VMs for synthetic and GoCJ datasets than HCSP instances-based datasets. Moreover, MODE has achieved a higher penalty and total execution cost for synthetic workload and GoCJ datasets as compared to the HCSP dataset due to increased violation of task deadlines. PSO-RADL has achieved 66% lower makespan, 92% higher ARUR, 85% reduced task rejection, 90% lower penalty, 82% minimized total cost on synthetic workload. MODE has gained lower costs for GoCJ and HCSP datasets because of their cost awareness approach, however, MODE has a higher task rejection ratio for all three datasets. It's because MODE is not deadline and resource-aware.

The scrutinized results reveal that DLBA has a higher makespan, cost, and lower resource utilization for GoCJ and HCSP datasets as compared to synthetic workload. It is due to the high scheduling overhead for larger datasets which limits the scalability of the DLBA technique. PSO-RADL has attained 77% and 70% higher ARUR, 63%, 14% reduced makespan, 81%, and 39% lower penalty cost, and 79% and 40% reduced total cost on GoCJ and HCSP datasets.

PSSELB has attained lower penalty and total execution cost for GoCJ and HCSP datasets, however, PSSELB has poor performance in terms of makespan, task response time, and task rejection for all three datasets as compared to PSO-RADL. This is because PSSELB does not lease any VM at the run time nor does it sup-

ports task deadline. Moreover, PSSELB has achieved lower resource utilization for synthetic workload than GoCJ and HCSP datasets. This is due to the small number of large-size tasks which results in assigning some larger tasks to the slower VMs.

Analysis of the experimental results shows that RALBA has attained better performance in terms of ARUR and makespan for the execution of synthetic workload based dataset. This is because RALBA is a resource-aware task scheduling technique and is more suitable for the positively skewed dataset. However, PSO-RADL has outperformed as compared to RALBA in terms of ARUR, makespan for GoCJ, and HCSP datasets. Moreover, PSO-RADL outperforms in terms of task response time, task rejection, penalty, and total cost for the execution of all three datasets. The overall comprehensive analysis of the experimental results reveals that the majority of the cases PSO-RADL perform better in terms of makespan, meeting task deadline, ARUR, task response time, penalty cost, and total execution cost. Moreover, the proposed approach outperform in terms of ARUR, makespan, and meeting task deadlines in almost all cases and showed comparable performance in terms of task response time, penalty, and total execution cost as compared to their counterpart. Its because the PSO-RADL is resource and deadline aware. However, due to the inherent nature of being a meta-heuristics-based algorithm, PSO-RADL performs better for small and medium-size workload-based datasets as compared to larger datasets.

5.5 Chapter Summary

Cloud computing has two key actors, Cloud Service Providers (CSP) and end-users of the cloud. The ultimate objective of CSPs is to maximize cloud profit and cost minimization is the key objective of cloud end users. To meet these objectives, a balanced distribution of workload plays an important role. Many heuristics and meta-heuristics-based task scheduling has been proposed. Most of the existing state-of-the-art task scheduling heuristics either consider a single evaluation parameter or multiple non-conflicting parameters. This leads to a need for efficient schedulers that can provide an optimized solution for multiple and con-

flicting task scheduling objectives. This chapter presents an adaptive PSO-based resource and deadline-aware dynamic load balancer for task scheduling in cloud computing. The performance of the proposed technique PSO-RADL is evaluated and compared with state-of-the-art task scheduling heuristics like MODE, DLBA, DC-DLBA, D-MaxMin, PSSELB, and RALBA. Three benchmark datasets have been used for performance evaluation and experimentations. Experimental results reveal that the proposed tasks scheduling technique outperforms state-of-the-art task scheduling heuristics in terms of average makespan, resource utilization, task response time, meeting tasks deadline, penalty cost, and total execution cost. In the future, it is intended to extend the proposed approach for combining heuristics and meta-heuristic-based task scheduling approaches. Moreover, PSO-RADL can be extended for workflow-based (dependent tasks) task scheduling.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

Cloud computing has emerged as the most favorite computing platform for researchers and industry. To get the full benefit of the cloud and achieve high user satisfaction, cloud service providers demand load balancing in terms of higher resource utilization and executing user's tasks within their deadlines. This leads to needing for efficient task scheduling algorithms. Cloud task scheduling and load balancing have emerged as important and challenging research issues in cloud computing. To achieve higher resource utilization and meeting task deadlines, a number of heuristics and meta-heuristic based task scheduling algorithms have been proposed.

These algorithms can be static, dynamic, and batch dynamic. The static task scheduling algorithms suffer from issues like poor-utilization of cloud resources, higher makespan, task response time, and high task rejection ratio. This because tasks once mapped to VMs can not be re-adjusted at run time and the newly arrived tasks with a shorter deadline will have to wait till the execution of already mapped tasks. The batch dynamic task scheduling algorithm provided dynamism at the batch level. However, these algorithms are unable to re-adjust the already mapped to tasks at runtime, have issues like new batch formation based response time delay, and inter-batch under-utilization cloud resources. This

results in under-utilization of cloud resources, high makespan and task response time, and are unable to meet the deadline of tasks with a shorter deadline.
Dynamic task scheduling algorithms can alter the tasks mapping and execution strategy at runtime. However, the majority of the dynamic scheduling algorithms do not consider the deadlines of the tasks and accept input tasks in the form of batch-based. Therefore, the majority of these approaches are unable to achieve high utilization of cloud resources, minimized task execution and response time, and reduced tasks rejection ratio.
To achieve higher resource utilization, meeting task deadlines, minimize task rejection ratio with reduced makespan, *A Resource-Aware Dynamic Load-balancer for Deadline Constrained cloud Tasks* (RADL) technique has been proposed. The RADL scheduler map user's tasks based on the expected minimum completion time and considering the task deadline into account. This approach has the ability to update VM and tasks status and re-adjust the mapped tasks at runtime which helps in allocating tasks based on more realistic expected completion time. The RADL task scheduler is evaluated and compared with the state-of-the-art scheduling algorithms using three benchmark datasets. Experimental results shows that the proposed technique has attained up to 67.74%, 303.57%, 259.2%, 146.13%, 405.06%, and 259.14% improvement in terms of average resource utilization, meeting tasks deadlines, lower makespan, task response time, penalty cost, and task execution cost respectively as compared to the state-of-the-art tasks scheduling heuristics.

To evaluate the performance of task scheduling algorithms, most of the existing state-of-the-art scheduling techniques either considering a single metric or multiple evaluation metrics individually. However, improving a single parameter or multiple parameters individually may not improve the overall performance of the cloud. In the second contribution of this thesis *An Overall performance gain based dynamic load-balancer for deadline constrained task* (OG-RADL) has been proposed. OG-RADL computes and investigates the overall performance of the scheduling algorithm by combining multiple evaluation parameters like ARUR, makespan, task response time, and task rejection ratio. A novel normalization technique is proposed that attempts to overcome the limitations of the existing

normalization techniques. The performance of the OG-RADL has evaluated using three benchmark datasets. The evaluation results reveal that the OG-RADL has gained 32.2%, 11.15%, and 28.33% improved overall performance as compared to the contemporary approaches for the execution of Synthetic, GoCJ, and HCSP dataset respectively.

Heuristic-based task scheduling algorithms find near-optimal solutions in a fast manner, having simple implementation, lower scheduling overhead, and are suitable for considering scheduling objectives with non-conflicting parameters. However, in the real cloud scenario, users can have the quality of service requirements with conflicting parameters like tasks execution time and cost. A number of meta-heuristic based task scheduling algorithms have been proposed in the literature.

In this thesis, a Particle Swarm Optimization based resource-aware dynamic load-balancing scheduler (PSO-RADL) is proposed. To balance the local and global search of particles, a novel and adaptive inertia weight strategy have been proposed. Experimental results reveal that the PSO-RDAL has gained up to 66%, 162%, 56%, 89%, 98%, and 97% enhancement in terms of makespan, average resource utilization, task response time, meeting task deadline, penalty cost, and total execution cost respectively as compared to existing state-of-the-art tasks scheduling heuristics.

6.2 Limitations

The proposed approach gives equal importance to different evaluation parameters like ARUR, Task Rejection Ratio, and makespan. However, sometimes users can assign high weightage to one parameter than others. In the usage scenario of the proposed scheduling technique, all the input tasks should be independent and compute-intensive and not considering parameters like priority, memory, bandwidth, and communication latency cloud resources. However, in workflow-based dependent tasks, the execution sequence of tasks matters, and memory, bandwidth, priority, and communication latency can be the essential factors that can hurdle cloud performance. The proposed approach tries to accommodate the newly arrived deadline-based tasks and reduce tasks rejection. However, this can lead to

starvation for tasks with no deadline if all the newly arrived tasks are deadline-based. The proposed scheduling technique also not considering the SLA awareness and machine learning technique for cloud task scheduling.

6.3 Future Directions

1. The key challenges in the cloud tasks scheduling are dispersion, uncertainty, heterogeneity of the resources, and user applications. These challenges are not resolved by the traditional cloud scheduling approaches and there is a need to design and develop cloud-oriented applications and services by taking care of these challenges in the cloud computing environment.

2. Applying online (dynamic) task scheduling to assign and release resources based on consumer budget and needs. Heuristics and Meta-heuristics algorithm needs to incorporate SLA-based Quality of Service (QoS) requirements between consumers and providers.

3. Dynamic scalability is the ability to acquire and release resources dynamically according to the incoming workload. Dynamic scalability helps efficient utilization of resources when the resource demand decreases and increases end-user satisfaction by reducing SLA violation in high resource demand hours. However, the current dynamic task scheduling approaches are not able to estimate the user demand and respond accordingly. The Machine Learning-based workload prediction and resource management can help to lease and release resources at runtime. Moreover, a machine learning-based cloud scheduler also helps in the selection of suitable task scheduling algorithms.

4. Mapping of tasks on resources located on multiple data centers and exchange information among datacenters. However, most of the researchers have designed algorithms for mapping tasks to VMs located on the same datacenter where time and cost of data transfer are not considered. In many real cloud scenarios, data transfer cost may affect the cloud performance and need to be considered for performance evaluation in cloud tasks scheduling algorithms.

Moreover, novel techniques should be investigated that apply the sharing of information and effective communication between data centers

5. Energy consumption is one of the most important factors to reduce the task execution cost. Improving energy efficiency is a big challenge because an idle machine can consume up to 70% of energy as compared to a working machine. To efficiently utilize cloud resources and maximize the CSP profit in a situation when cloud resources demand fluctuates. It is important to correctly estimate the dynamic resource demands as the accuracy of forecasting directly affects the results of resource allocations. Regression-based and machine learning-based resource demand prediction techniques need to be explored and evaluated in cloud Computing.

6. Cloud computing is an important platform to execute users' web-based services as a pay-per-use mode. However, due to high fluctuations in user demands, it becomes difficult to effectively utilize cloud resources. Autonomic management systems (systems that adapt according to the situation) play a vital role to cope with these challenges. However, the current autonomic systems need to be improved and further investigated for dynamic provisioning of cloud resources to fulfill users' QoS requirements and enhance cloud system efficiency.

Bibliography

[1] A. Hussain, M. Aleem, A. Khan, M. A. Iqbal, and M. A. Islam, "Ralba: a computation-aware load balancing scheduler for cloud computing," *Cluster Computing*, vol. 21, no. 3, pp. 1667–1680, 2018.

[2] S. Nabi and M. Khan, "An analysis of application level security in service oriented architecture," *International Journal of Modern Education and Computer Science*, vol. 6, no. 2, p. 27, 2014.

[3] S. Nabi, S. U. Rehman, S. Fong, and K. Aziz, "A model for implementing security at application level in service oriented architecture," *Journal of Emerging Technologies in Web Intelligence*, vol. 6, no. 1, pp. 157–163, 2014.

[4] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *2008 grid computing environments workshop*, pp. 1–10, Ieee, 2008.

[5] N. Sadashiv and S. D. Kumar, "Cluster, grid and cloud computing: A detailed comparison," in *2011 6th International Conference on Computer Science & Education (ICCSE)*, pp. 477–482, IEEE, 2011.

[6] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 50–55, 2008.

[7] M. Kumar and S. Sharma, "Pso-based novel resource scheduling technique to improve qos parameters in cloud computing," *Neural Computing and Applications*, pp. 1–24, 2019.

[8] M. Aruna, D. Bhanu, and S. Karthik, "An improved load balanced meta-heuristic scheduling in cloud," *Cluster Computing*, pp. 1–9, 2017.

[9] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente, "From infrastructure delivery to service management in clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1226–1240, 2010.

[10] G. Sharma and P. Banga, "Task aware switcher scheduling for batch mode mapping in computational grid environment," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, 2013.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[12] D. K. Barry and D. Dick, *Web Services, Service-Oriented Architectures, and Cloud Computing: The Savvy Manager's Guide*. 2013.

[13] D. Hazra, A. Roy, S. Midya, and K. Majumder, "Distributed task scheduling in cloud platform: A survey," in *Smart Computing and Informatics*, pp. 183–191, Springer, 2018.

[14] A. Sajjad, A. A. Khan, and M. Aleem, "Energy-aware cloud computing simulators: A state of the art survey," *International Journal of Applied Mathematics Electronics and Computers*, vol. 6, no. 2, pp. 15–20, 2018.

[15] R. v. d. M. G. S. Moore and Gartner, *Gartner Industry analyst firm. Gartner, Inc.* Gartner, Inc., 2018 (accessed Jauuary 15, 2019). https://www.gartner.com/newsroom/id/3871416.com.

[16] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of internet services and applications*, vol. 1, no. 1, pp. 7–18, 2010.

[17] G. S. Bhunia and P. K. Shit, "E-research and geocomputation in public health," in *GeoComputation and Public Health*, pp. 37–78, Springer, 2021.

[18] Y. Wang, Y. Ban, and X. Hong, "Overview of e-science research in china," in *China's e-Science Blue Book 2020*, pp. 3–12, Springer, 2021.

[19] N. Alaei and F. Safi-Esfahani, "Repro-active: a reactive–proactive scheduling method based on simulation in cloud computing," *The Journal of Supercomputing*, vol. 74, no. 2, pp. 801–829, 2018.

[20] M. Adhikari and T. Amgoth, "Heuristic-based load-balancing algorithm for iaas cloud," *Future Generation Computer Systems*, vol. 81, pp. 156–165, 2018.

[21] S. Mousavi, A. Mosavi, and A. R. Varkonyi-Koczy, "A load balancing algorithm for resource allocation in cloud computing," in *International Conference on Global Research and Education*, pp. 289–296, Springer, 2017.

[22] M. Ibrahim, S. Nabi, R. Hussain, M. S. Raza, M. Imran, S. A. Kazmi, A. Oracevic, and F. Hussain, "A comparative analysis of task scheduling approaches in cloud computing," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 681–684, IEEE, 2020.

[23] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 772–783, 2018.

[24] E. S. Alkayal, N. R. Jennings, and M. F. Abulkhair, "Survey of task scheduling in cloud computing based on particle swarm optimization," in *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pp. 1–6, IEEE, 2017.

[25] H. B. Alla, S. B. Alla, A. Touhafi, and A. Ezzati, "A novel task scheduling approach based on dynamic queues and hybrid meta-heuristic algorithms for cloud computing environment," *Cluster Computing*, vol. 21, no. 4, pp. 1797–1820, 2018.

[26] C. Gogos, C. Valouxis, P. Alefragis, G. Goulas, N. Voros, and E. Housos, "Scheduling independent tasks on heterogeneous processors using heuristics and column pricing," *Future Generation Computer Systems*, vol. 60, pp. 48–66, 2016.

[27] M. Xu, W. Tian, and R. Buyya, "A survey on load balancing algorithms for virtual machines placement in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. e4123, 2017.

[28] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing," *Journal of Network and Computer Applications*, vol. 143, pp. 1–33, 2019.

[29] M. Ibrahim, S. Nabi, A. Baz, N. Naveed, and H. Alhakami, "Towards a task and resource aware task scheduling in cloud computing: An experimental comparative evaluation," *International Journal of Networked and Distributed Computing*, vol. 8, no. 3, pp. 131–138, 2020.

[30] Z. Chen, Y. Zhu, Y. Di, and S. Feng, "A dynamic resource scheduling method based on fuzzy control theory in cloud environment," *Journal of Control Science and Engineering*, vol. 2015, p. 10, 2015.

[31] E. Santos-Neto, W. Cirne, F. Brasileiro, and A. Lima, "Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids," in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 210–232, Springer, 2004.

[32] S.-L. Chen, Y.-Y. Chen, and S.-H. Kuo, "Clb: A novel load balancing architecture and algorithm for cloud services," *Computers & Electrical Engineering*, vol. 58, pp. 154–160, 2017.

[33] Y. Mao, X. Chen, and X. Li, "Max-min task scheduling algorithm for load balance in cloud computing," in *Proceedings of International Conference on Computer Science and Information Technology*, pp. 457–465, Springer, 2014.

[34] M. Kumar and S. Sharma, "Deadline constrained based dynamic load balancing algorithm with elasticity in cloud environment," *Computers & Electrical Engineering*, vol. 69, pp. 395–411, 2018.

[35] S. K. Mishra, M. A. Khan, B. Sahoo, D. Puthal, M. S. Obaidat, and K. Hsiao, "Time efficient dynamic threshold-based load balancing technique for cloud computing," in *Computer, Information and Telecommunication Systems (CITS), 2017 International Conference on*, pp. 161–165, IEEE, 2017.

[36] S. Nabi and M. Ahmed, "Og-radl: overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks," *The Journal of Supercomputing*, pp. 1–33, 2021.

[37] A. Deldari, M. Naghibzadeh, and S. Abrishami, "Cca: a deadline-constrained workflow scheduling algorithm for multicore resources on the cloud," *The journal of Supercomputing*, vol. 73, no. 2, pp. 756–781, 2017.

[38] N. Panwar, S. Negi, and M. M. S. Rauthan, "Non-live task migration approach for scheduling in cloud based applications," in *International Conference on Next Generation Computing Technologies*, pp. 124–137, Springer, 2017.

[39] M. Ibrahim, M. A. Iqbal, M. Aleem, and M. A. Islam, "Sim-cumulus: An academic cloud for the provisioning of network-simulation-as-a-service (nsaas)," *IEEE Access*, vol. 6, pp. 27313–27323, 2018.

[40] J. O. Gutierrez-Garcia and A. Ramirez-Nafarrate, "Collaborative agents for distributed load management in cloud data centers using live migration of virtual machines," *IEEE transactions on services computing*, vol. 8, no. 6, pp. 916–929, 2015.

[41] S. Torabi and F. Safi-Esfahani, "A dynamic task scheduling framework based on chicken swarm and improved raven roosting optimization methods in cloud computing," *The Journal of Supercomputing*, vol. 74, no. 6, pp. 2581–2626, 2018.

[42] P. Kaur and M. Sharma, "Diagnosis of human psychological disorders using supervised learning and nature-inspired computing techniques: a meta-analysis," *Journal of medical systems*, vol. 43, no. 7, pp. 1–30, 2019.

[43] Y. Hu, K. Liu, X. Zhang, L. Su, E. Ngai, and M. Liu, "Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review," *Applied Soft Computing*, vol. 36, pp. 534–551, 2015.

[44] Y. Ding, K. Zhou, and W. Bi, "Feature selection based on hybridization of genetic algorithm and competitive swarm optimizer," *Soft Computing*, pp. 1–10, 2020.

[45] A. Agrawal and S. Tripathi, "Particle swarm optimization with adaptive inertia weight based on cumulative binomial probability," *Evolutionary Intelligence*, pp. 1–9, 2018.

[46] S. C. Satapathy, S. Chittineni, S. M. Krishna, J. Murthy, and P. P. Reddy, "Kalman particle swarm optimized polynomials for data classification," *Applied Mathematical Modelling*, vol. 36, no. 1, pp. 115–126, 2012.

[47] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *2013 8th international conference on computer engineering & systems (ICCES)*, pp. 64–69, IEEE, 2013.

[48] D. Kaur and S. Singh, "An efficient job scheduling algorithm using min-min and ant colony concept for grid computing," *International Journal Of Engineering And Computer Science*, vol. 3, no. 07, pp. 6943–6949, 2014.

[49] A. Brabazon, W. Cui, and M. O'Neill, "The raven roosting optimisation algorithm," *Soft Computing*, vol. 20, no. 2, pp. 525–545, 2016.

[50] S.-C. Chu, P.-W. Tsai, and J.-S. Pan, "Cat swarm optimization," in *Pacific Rim international conference on artificial intelligence*, pp. 854–858, Springer, 2006.

[51] S. Deb, X.-Z. Gao, K. Tammi, K. Kalita, and P. Mahanta, "Recent studies on chicken swarm optimization algorithm: a review (2014–2018)," *Artificial Intelligence Review*, pp. 1–29, 2019.

[52] Y. Xiao and A. Konak, "A genetic algorithm with exact dynamic programming for the green vehicle routing & scheduling problem," *Journal of Cleaner Production*, vol. 167, pp. 1450–1463, 2017.

[53] P. Krishnadoss and P. Jacob, "Ocsa: task scheduling algorithm in cloud computing environment," *International Journal of Intelligent Engineering and Systems*, vol. 11, no. 3, pp. 271–279, 2018.

[54] N. Dordaie and N. J. Navimipour, "A hybrid particle swarm optimization and hill climbing algorithm for task scheduling in the cloud environments," *ICT Express*, vol. 4, no. 4, pp. 199–202, 2018.

[55] S. S. Gill, R. Buyya, I. Chana, M. Singh, and A. Abraham, "Bullet: particle swarm optimization based scheduling technique for provisioned cloud resources," *Journal of Network and Systems Management*, vol. 26, no. 2, pp. 361–400, 2018.

[56] D. B. LD and P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied soft computing*, vol. 13, no. 5, pp. 2292–2303, 2013.

[57] K. Y. China and Q. Y. China, "A task scheduling based on simulated annealing algorithm in cloud computing," *International Journal of Hybrid Information Technology*, vol. 9, no. 6, pp. 403–412, 2016.

[58] A. Gupta, H. Bhadauria, and A. Singh, "Load balancing based hyper heuristic algorithm for cloud task scheduling," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 6, pp. 5845–5852, 2021.

[59] A. Hussain and M. Aleem, "Gocj: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, p. 38, 2018.

[60] R. v. d. M. G. S. Moore and Gartner, *Gartner Industry analyst firm. Gartner, Inc.* Gartner, Inc., 2018 (accessed September

12, 2021). https://www.gartner.com/en/newsroom/press-releases/2021-08-02-gartner-says-four-trends-of-public-cloud.

[61] A. Hussain, M. Aleem, M. A. Iqbal, and M. A. Islam, "Investigation of cloud scheduling algorithms for resource utilization using cloudsim," *Computing and Informatics*, vol. 38, no. 3, pp. 525–554, 2019.

[62] A. Hussain, M. Aleem, M. A. Islam, and M. Iqbal, "A rigorous evaluation of state-of-the-art scheduling algorithms for cloud computing," *IEEE Access*, vol. 6, pp. 75033–75047, 2018.

[63] M. Ibrahim, S. Nabi, A. Baz, H. Alhakami, M. S. Raza, A. Hussain, K. Salah, and K. Djemame, "An in-depth empirical investigation of state-of-the-art scheduling approaches for cloud computing," *IEEE Access*, vol. 8, pp. 128282–128294, 2020.

[64] F. Xhafa and A. Abraham, "A compendium of heuristic methods for scheduling in computational grids," in *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 751–758, Springer, 2009.

[65] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," in *Proceedings Seventh Heterogeneous Computing Workshop (HCW'98)*, pp. 79–87, IEEE, 1998.

[66] R. V. Rasmussen and M. A. Trick, "Round robin scheduling–a survey," *European Journal of Operational Research*, vol. 188, no. 3, pp. 617–636, 2008.

[67] A. K. Bardsiri and S. M. Hashemi, "A comparative study on seven static mapping heuristics for grid scheduling problem," *International Journal of Software Engineering and Its Applications*, vol. 6, no. 4, pp. 247–256, 2012.

[68] O. Elzeki, M. Rashad, and M. Elsoud, "Overview of scheduling tasks in distributed computing systems," *International Journal of Soft Computing and Engineering*, vol. 2, no. 3, pp. 470–475, 2012.

[69] L. Kong, J. P. B. Mapetu, and Z. Chen, "Heuristic load balancing based zero imbalance mechanism in cloud computing," *Journal of Grid Computing*, vol. 18, no. 1, pp. 123–148, 2020.

[70] J. Praveenchandar and A. Tamilarasi, "Dynamic resource allocation with optimized task scheduling and improved power management in cloud computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 3, pp. 4147–4159, 2021.

[71] M. Kumar, K. Dubey, and S. Sharma, "Elastic and flexible deadline constraint load balancing algorithm for cloud computing," *Procedia Computer Science*, vol. 125, pp. 717–724, 2018.

[72] B. Wang, Y. Song, J. Cao, X. Cui, and L. Zhang, "Improving task scheduling with parallelism awareness in heterogeneous computational environments," *Future Generation Computer Systems*, vol. 94, pp. 419–429, 2019.

[73] M. Ghobaei-Arani, A. Souri, T. Baker, and A. Hussien, "Controcity: an autonomous approach for controlling elasticity using buffer management in cloud computing environment," *IEEE Access*, vol. 7, pp. 106912–106924, 2019.

[74] M. A. Alworafi and S. Mallappa, "A collaboration of deadline and budget constraints for task scheduling in cloud computing," *Cluster Computing*, vol. 23, no. 2, pp. 1073–1083, 2020.

[75] S. Wang, Z. Ding, and C. Jiang, "Elastic scheduling for microservice applications in clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 98–115, 2020.

[76] M. Yazdanbakhsh, R. K. M. Isfahani, and M. Ramezanpour, "Mode: a multi-objective strategy for dynamic task scheduling through elastic cloud resources," *Majlesi Journal of Electrical Engineering*, vol. 14, no. 2, pp. 127–141, 2020.

[77] A. Shahidinejad, M. Ghobaei-Arani, and M. Masdari, "Resource provisioning using workload clustering in cloud computing environment: a hybrid approach," *Cluster Computing*, vol. 24, no. 1, pp. 319–342, 2021.

[78] S. Nabi, M. Ibrahim, and J. M. Jimenez, "Dralba: Dynamic and resource aware load balanced scheduling approach for cloud computing," *IEEE Access*, vol. 9, pp. 61283–61297, 2021.

[79] E. K. Tabak, B. B. Cambazoglu, and C. Aykanat, "Improving the performance of independenttask assignment heuristics minmin, maxmin and sufferage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1244–1256, 2014.

[80] X. Wang, C. S. Yeo, R. Buyya, and J. Su, "Optimizing the makespan and reliability for workflow applications with reputation and a look-ahead genetic algorithm," *Future Generation Computer Systems*, vol. 27, no. 8, pp. 1124–1134, 2011.

[81] L. Zhang, Y. Chen, R. Sun, S. Jing, and B. Yang, "A task scheduling algorithm based on pso for grid computing," *International Journal of Computational Intelligence Research*, vol. 4, no. 1, pp. 37–43, 2008.

[82] A. Khalili and S. M. Babamir, "Makespan improvement of pso-based dynamic scheduling in cloud environment," in *Electrical Engineering (ICEE), 2015 23rd Iranian Conference on*, pp. 613–618, IEEE, 2015.

[83] M. Kumar and S. Sharma, "Pso-cogent: Cost and energy efficient scheduling in cloud environment with deadline constraint," *Sustainable Computing: Informatics and Systems*, vol. 19, pp. 147–164, 2018.

[84] M. P. McGarry, M. Reisslein, C. J. Colbourn, M. Maier, F. Aurzada, and M. Scheutzow, "Just-in-time scheduling for multichannel epons," *Journal of Lightwave Technology*, vol. 26, no. 10, pp. 1204–1216, 2008.

[85] A. Shahidinejad, M. Ghobaei-Arani, and L. Esmaeili, "An elastic controller using colored petri nets in cloud computing environment," *Cluster Computing*, vol. 23, no. 2, pp. 1045–1071, 2020.

[86] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95*, vol. 94, 2010.

[87] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan, "An analysis of traces from a production mapreduce cluster," in *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pp. 94–103, IEEE Computer Society, 2010.

[88] S. K. Panda and P. K. Jana, "Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment," *Information Systems Frontiers*, vol. 20, no. 2, pp. 373–399, 2018.

[89] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[90] D. Singh and B. Singh, "Investigating the impact of data normalization on classification performance," *Applied Soft Computing*, vol. 97, p. 105524, 2020.

[91] A. Pandita, P. K. Upadhyay, and N. Joshi, "Prediction of service-level agreement violation in cloud computing using bayesian regularisation," in *International Conference on Advanced Machine Learning Technologies and Applications*, pp. 231–242, Springer, 2020.

[92] V. Gajera, R. Gupta, P. K. Jana, *et al.*, "An effective multi-objective task scheduling algorithm using min-max normalization in cloud computing," in *2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pp. 812–816, IEEE, 2016.

[93] A. Jain, K. Nandakumar, and A. Ross, "Score normalization in multimodal biometric systems," *Pattern recognition*, vol. 38, no. 12, pp. 2270–2285, 2005.

[94] G. N. Reddy and S. P. Kumar, "Maco-mots: modified ant colony optimization for multi objective task scheduling in cloud environment," *International Journal of Intelligent Systems and Applications*, vol. 11, no. 1, pp. 73–79, 2019.

[95] M. A. Alsaih, R. Latip, A. Abdullah, S. K. Subramaniam, and K. Ali Alezabi, "Dynamic job scheduling strategy using jobs characteristics in cloud computing," *Symmetry*, vol. 12, no. 10, p. 1638, 2020.

[96] J. Bezos, *ANN CIFC Data set (NNG-C)*. IEEE, 2006 (accessed Jauuary 15, 2019). https://www.neural-forecastingcompetition.com/.

[97] A. Kumar and S. Bawa, "A comparative review of meta-heuristic approaches to optimize the sla violation costs for dynamic execution of cloud services," *Soft Computing*, vol. 24, no. 6, pp. 3909–3922, 2020.

[98] G. Beni and J. Wang, "Swarm intelligence in cellular robotic systems," in *Robots and biological systems: towards a new bionics*, pp. 703–712, Springer, 1993.

[99] O. Ertenlice and C. B. Kalayci, "A survey of swarm intelligence for portfolio optimization: Algorithms and applications," *Swarm and evolutionary computation*, vol. 39, pp. 36–52, 2018.

[100] E. Bonabeau, D. d. R. D. F. Marco, M. Dorigo, G. Théraulaz, G. Theraulaz, *et al.*, *Swarm intelligence: from natural to artificial systems*. No. 1, Oxford university press, 1999.

[101] S. S. Senthilkumar, K. Brindha, N. K. Agrawal, and A. Vaidya, "Dynamic load balancing using honey bee algorithm: Load balancing," in *Encyclopedia of Information Science and Technology, Fifth Edition*, pp. 98–106, IGI Global, 2021.

[102] A. Panneerselvam and B. Subbaraman, "Multi-objective optimization for scientific workflow task scheduling in iaas cloud," *International Journal of Engineering & Technology*, vol. 7, no. 4.6, pp. 174–176, 2018.

[103] X. Huang, C. Li, H. Chen, and D. An, "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies," *Cluster Computing*, pp. 1–11, 2019.

[104] S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, "Adpso: Adaptive pso-based task scheduling approach for cloud computing," *Sensors*, vol. 22, no. 3, p. 920, 2022.