CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Resource-Aware and Load-Balanced Scheduling for Cloud Computing Environments

by

Altaf Hussain

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

in the

Faculty of Computing

Department of Computer Science

2020

Resource-Aware and Load-Balanced Scheduling for Cloud Computing Environments

By

Altaf Hussain

(PC 141006)

DEPARTMENT OF COMPUTER SCIENCE
CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY
ISLAMABAD
2020

Dedicated to my beloved parents and wife

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 "**Resource-Aware and Load-Balanced Scheduling for Cloud Computing Environments**" was conducted under the supervision of **Dr. Muhammad Aleem**. 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 **23 July, 2020.**

Student Name :	Mr. Altaf Hussain (PC141006)	[illegible]

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

Examination Committee :

(a)	External Examiner 1:	Dr. Sajjad A. Madani, Professor CUI, Wah Cantt Campus	[illegible]
(b)	External Examiner 2:	Dr. Ghulam Abbas, Associate Professor GIKI, Topi, Swabi	[illegible]
(c)	Internal Examiner :	Dr. Nadeem Anjum Assistant Professor CUST, Islamabad	[illegible]

Supervisor Name :	Dr. Muhammad Aleem Professor FAST-NUCES, Islamabad	[illegible]
Name of HoD :	Dr. Nayyer Masood Professor CUST, Islamabad	[illegible]
Name of Dean :	Dr. Muhammad Abdul Qadir Professor CUST, Islamabad	[illegible]

AUTHOR'S DECLARATION

I, **Mr. Altaf Hussain (Registration No. PC141006)**, hereby state that my PhD thesis titled, '**Resource-Aware and Load-Balanced Scheduling for Cloud Computing Environments**' 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.

[illegible]

(Mr. Altaf Hussain)

Dated: 23 July, 2020

Registration No : PC141006

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled "**Resource-Aware and Load-Balanced Scheduling for Cloud Computing Environments**" 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.

[illegible]

(Mr. Altaf Hussain)

Registration No : PC141006

Dated: 23 July, 2020

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. **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.

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

3. **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.

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

5. **A. Hussain**, M. Aleem, M. A. Iqbal, and M. A. Islam. ”Sla-ralba: cost-efficient and resource-aware load balancing algorithm for cloud computing,” *The Journal of Supercomputing*, vol. 75, no. 10, pp. 6777-6803, 2019.

Altaf Hussain
(PC 141006)

Acknowledgements

First of all, I would like to thank the Almighty Allah, for His divine guidance and providence. His support, blessings, goodness, and kindness were always with me. He blessed me with motivation, passion, and hard work. It was his blessings which made me able to plan, visualize, and execute my dreams into the reality. I would like to dedicate this achievement to Him, to my parents and to my wife whose immeasurable sacrifices have led to what I am today.

Every PhD scholar dreams for a visionary supervisor, and I am extremely thankful to almighty Allah, for connecting me up with Dr. Muhammad Aleem, being the best possible supervisor. Dr. Muhammad Aleem's guidance, inspiration, and motivations enabled me to bring out my best. I learned too much and the list is quite long but to be short, Dr. Muhammad Aleem introduced me to the scientific writing, supervised me to the art of critical thinking, encouraged me to stay focus on my research goals and work hard, gave me freedom to explore different ideas, and provided me the best working environment. I pay my deep regards to him and his greatness.

I would like to express my gratitude and special thanks to Dr. Abid Khan from COMSATS University Islamabad and Dr. Syed Nasir Mehmood Shah from Universiti Teknologi PETRONAS, Malaysia for helping me to improve my research skills, for his invaluable guidance, and suggestions on improving my work. In particular, I thank Dr. Muhammad Abdul Qadir, Dr. Nayyer Masood, Dr. Muhammad Arshad Islam, Dr. Muhammad Azhar Iqbal, Dr. Aamer Nadeem and Dr. Muhammad Tanvir Afzal for providing me valuable guidance, suggestions and support during my research. I would also like to thank all the past and current members of our *Parallel Computing Network* (PCN) research lab. at Capital University of Science and Technology, Islamabad for their friendship and valuable suggestions. I wish them best of luck in their research. The knowledge and skills I gained working here will greatly benefit my future career.

I am especially grateful to Faiza Qayyum for her helpful advice and suggestions in reviewing my research articles for grammatical correction prior to submitting for review process in journal publication.

I am very obliged to the *Higher Education Commission* (HEC) of Pakistan for awarding me with a fully funded scholarship to pursue my doctoral studies.

With my deep heart, I would like to thank my beloved parents, wife, brothers, sisters and other family members for their indispensable encouragement, unconditional love, and prayers throughout my PhD studies. I thank my beloved late father for always believing in me, for encouraging me to achieve every step of this difficult journey, and for his words of praise when they were most needed. Indeed, I will remember my father, whenever someone will call me as Dr. in future. I thank my mother for being an exceptional role model, for teaching me the meaning of perseverance and hard work, and for her words of prayers for my success and caring love. I would like to thank my wife for her selflessness, unconditional love, endless support, and inspiration. My heartiest gratitude to my wife, who took care of my children while I was busy in my research activities. Her encouragement, motivation, support, and understanding made me able to work on my research activities and to write PhD thesis. I would like to acknowledge my children: Ommama Altaf, Ashtafeela Altaf, Muhammad Shehbaz Khan, and Kashmala Altaf for their patience while I was away and busy in making efforts in this noble cause. Finally, I am obliged to thank and acknowledge my brothers and sisters for being a loving, caring and supportive family.

Lastly, I want to thank and offer my regards to all those who supported me in any respect during the completion of this thesis.

Altaf Hussain

Islamabad, Pakistan

July, 2020

Abstract

High Performance Computing (HPC) is becoming ever more significant research tool in modelling complex scientific problems. Being large-scale nature, scheduling techniques are key to effectuate the execution in Cloud environment at the lowest possible cost without the need of owing any physical infrastructure. Meanwhile, the provision of high-performance computing by employing fewer resources with a cap on financial constraints is also emerging as an appealing research area in Cloud. While constructing Cloud datacenters, we should not only adopt to the best suitable software and hardware practices, but also have to keep a close eye on datacenter operations, energy consumption, scalability, security, technological innovation, fault tolerance, and many other issues. In addition, the resources should be utilized in an efficient manner along with considering the wisdom of general framework of Cloud scheduling.

In this thesis, a computation-aware load balancing scheme known as *Resource-Aware Load Balancing Algorithm* (RALBA) is proposed for scheduling compute-intensive, independent, and non-preemptive jobs on a compute Cloud. Using RALBA, the HPC workload can be distributed in a load-balanced manner guaranteeing improved resource utilization. RALBA scheme is designed with the intention to contemplate the total computing capabilities (i.e., computing powers) of *Virtual Machines* (VMs) and the computing requirements of the submitted HPC workload, which provides a load-balanced schedule with improved utilization of virtual computing resources and ultimately that of underlying physical machines in Cloud datacenters. RALBA scheme is evaluated on varying workload compositions (i.e., using several synthetic and benchmark scientific datasets) by employing different computing environments (i.e., heterogenous computing machines). The performance outcomes of RALBA have revealed that it provides substantial improvement against traditional and state-of-the-art scheduling algorithms in terms of makespan, throughput, and resource utilization. Moreover, the potential of eight state-of-the-art algorithms against RALBA is investigated in terms of makespan and resource utilization in amalgamation with machine-level load balancing using nine different instances of two benchmark scientific datasets (i.e., employing jobs heterogeneity) on seven different computing architectures The empirical examination endorsed that RALBA produces a better load-balanced schedule ensuring to release all the computing resources within almost similar quantum of time, which effectuate the *Cloud Service Providers* (CSPs) to reduce the execution delays effecting the next batch of Cloud jobs.

In addition, the HPC datasets are increasingly becoming more pertinent when executing resource allocation, and load balancing techniques for an eagle-eyed examination of efficacy and performance on Cloud. However, a real Cloud dataset is hard to acquire for such purpose due to users' data confidentiality and policies maintained in SLAs by Cloud service providers. Therefore, a new dataset known as *Google Cloud Jobs* (GoCJ) is also proposed in this dissertation as an alternative to benchmark workloads for scheduling and resource provisioning in a compute Cloud. The GoCJ dataset is realistic dataset generated using Monte Carlo simulation considering the real workload as perceived in Google cluster traces.

SLA standardizes the assumptions and responsibilities of both the Cloud users and CSPs that acts as a roadmap to the successful implementation of Cloud services. Therefore, an extension of RALBA is also presented in the form of *SLA and Resource-Aware Load Balancing Algorithm* (SLA-RALBA) for heterogeneous Cloud, which is a cost-efficient and computation-aware load balancing technique. The empirical results evidently revealed that SLA-RALBA provides improved balance between execution time and cost by guaranteeing a drastic improvement in resource utilization on Cloud as compared to existing cost-efficient SLA-techniques.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

Aliquem	Active List Queue Method
AORDCT	Average Ordinary Task Completion Time
AOT	Average Offloading Time
ARUR	Avergae Resource Utilization Ratio
ATRTR	Average Task Response Time Ratio
ATPT	Average Task Pending Time
AVIPCT	Average VIP Task Completion Time
AWT	Average Waiting Time
CDC	Cloud Data Centers
CSP	Cloud Service Provider
DRR	Deficit Round-Robin
ECMM	Elastic Cloud Max-Min
EFT	Earliest Finish Time
ETC	Expected Time to Compute
GoCJ	Google Cloud Jobs
GWA-T	Grid Workload Archiver TuDelft
HCSP	Heterogeneous Computing Scheduling Problems
HDC	Heterogeneous Distributed Computing
HPC	High Performance Computing
IaaS	Infrastructure-as-a-Service
LBIMM	Load Balance Improved Min-Min
MC	Monte Carlo
MCT	Minimum Completion Time
MET	Minimum Execution Time

MI	Million Instructions
MIPS	Million Instructions Per Second
MWAP	Mixed Workload-Aware Policy
NIST	National Institute of Standards and Technology
oi-Factor	Overall Improvement Factor
OLB	Opportunistic Load Balancing
PALBIMM	User-Priority-Aware Load Balance Improved Min-Min
PaaS	Platform-as-a-Service
PM	Physical Machine
PSSLB	Proactive Simulation-based Scheduling and Load Balancing
QoS	Quality of Service
QWMTM	QoS Guided Weighted Mean Time-Min
QWMTS	QoS Weighted Mean Time Min-Min Max-Min Selective
RALBA	Resource-Aware Load Balancing Algorithm
RASA	Resource-Aware Scheduling Algorithm
RNG	Random Number Generation
RR	Round Robin
RS	Random Selection
SaaS	Software-as-a-Service
SLA	Service Level Agreements
SLA-MCT	Service Level Agreement - Minimum Completion Time
SLA-Min-Min	Service Level Agreement - Min-Min
SLA-RALBA	SLA and Resource-Aware Load Balancing Algorithm
SBM^2	Skewness-Based Min-Min Max-Min
TASA	Task-Aware Scheduling Algorithm
TETM	Task Execution Time Modeling
VMs	Virtual Machines

Symbols

VMS	Set of VMs in a Cloud datacenter
$vmCrMap$	Set of VMs with with its computing ratio
$vmCrMap_j$	Computing ration of VM_j
CLS	Set of Cloudlets (to be scheduled)
$Cloudlet_i.MI$	Size of $Cloudlet_i$ in MI [1]
$VM_j.MIPS$	Computing power of VM_j in MIPS [1]
$RPCloudlet_j$	Remaining possible Cloudlets (can be assigned to VM_j)
$maxPCloudletVM_j$	Largest Cloudlet in $RPCloudlet_j$ that is assigned to VM_j
$minCloudlet$	Smallest sized Cloudlet (to be scheduled)
$maxCloudlet$	Largest sized Cloudlet (to be scheduled)
$VMShare$	Set of sorted VMs with computing share
$VMShare_j$	Computing share of VM_j (in MI)
$Largest\_VMShare$	Current largest computing share for any VM in VMS
$Cloudlet\_EFT_i$	Earliest finish time of Cloudlet_i
$Fill\_Scheduler$	Schedules $Clouldet_i$ to VM_j based on $VMShare_j$
$Spill\_Scheduler$	Schedules $Clouldet_i$ to VM_j based on $Cloudlet\_EFTi$
$vmList$	List of VMs in Cloud datacenter
$cloudletList$	List of Cloudlets (to scheduled)
$cloudletVmMap$	Allocation Map(vm,cloudlet)
$totalLength$	Sum of Cloudlet sizes in cloudletList
$vShareMap$	Computing share Map(vm,share)
$vShareMap_v$	Computing sare of VM v in vShareMap
$newVShare$	Modified computing share of a VM

Chapter 1

Introduction

This chapter presents the background and general architecture of Cloud computing. Further, it presents the motivation, summarizes the key contributions, and the organization of dissertation.

1.1 Cloud Computing

Cloud computing envisions computing in the form of services provided as a utility computing over the Internet [2]. In 2006, Eric Schmidt (CEO of Google) envisioned the term Cloud as a business model for service provisioning to the customers over the Internet. Luis M. Vaquero et al. defined Cloud as [3] "Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized *Service Level Agreements* (SLAs)".

Further in September 2011, *National Institute of Standards and Technology* (NIST) provided a more implicit definition [4] as "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction". The Cloud model is comprised of five essential characteristics [4] as follows:

1. **On-demand self-service:** the user can unilaterally provision the computing capabilities as needed without human interaction with service providers;
2. **Broad network access:** the computing capabilities are made available over the network through a standard mechanism;
3. **Resource pooling:** the computing resources are pooled to serve in a multi-tenant model with different physical and virtual resources dynamically assigned as per user demand,
4. **Rapid elasticity:** the computing capabilities can be elastically provisioned and released to scale up and down rapidly as required by the user, and
5. **Measured service:** Cloud automatically control the resource usage by leveraging a metering capability providing transparency both for provider and consumer of the utilized services.

Based on capabilities and services being offered to the users, the Cloud model comprises three service offerings into a hierarchy of as-a-service terms [5, 6]. Fig. 1.1 presents the architecture of Cloud computing showing the resources managed at each layer of the Cloud model and with relevant examples. These three well-known models of Cloud services are:

1. **Software-as-a-Service:** It provides access to online software applications over the web enabling users instead of installing and maintaining locally on the computer system. i.e., Salesforce.com [7], DropBox [8], Google Drive [9] and facebook [10] etc.
2. **Platform-as-a-Service:** It provides services in the form of an environment in which the developers and programmers can easily code and deploy the

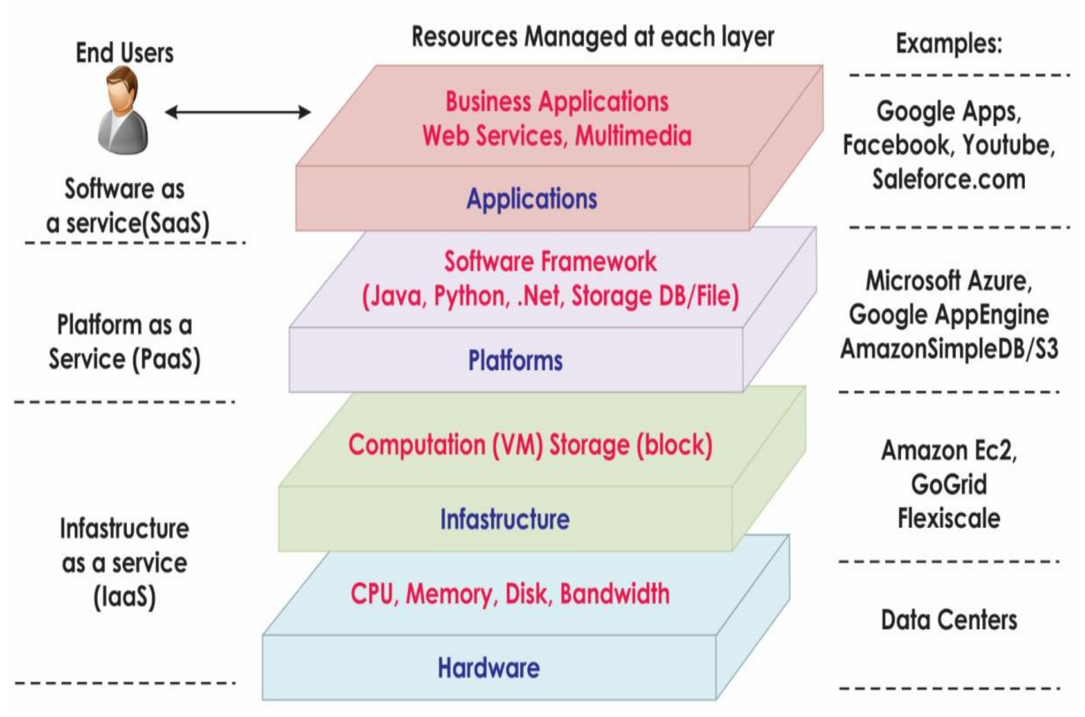

End Users
Resources Managed at each layer
Examples:
Business Applications
Web Services, Multimedia
Applications
Google Apps, Facebook, Youtube, Saleforce.com
Software as a service(SaaS)
Software Framework
(Java, Python, .Net, Storage DB/File)
Platforms
Platform as a Service (PaaS)
Microsoft Azure, Google AppEngine AmazonSimpleDB/S3
Computation (VM) Storage (block)
Infastructure
Amazon Ec2, GoGrid Flexiscale
Infastructure as a service (IaaS)
CPU, Memory, Disk, Bandwidth
Hardware
Data Centers

FIGURE 1.1: Cloud Computing Architecture [6].

Cloud applications with the ability to scale automatically without estimating the resource capacity of the applications. i.e., Google App Engine [11], and Microsoft Azure [12] etc.

3. **Infrastructure-as-a-Service:** It provides computing resources in the form of virtual machine that comprises predefined CPU, storage, bandwidth, and memory capacity with a varying prices for different application requirements [5]. In addition to virtual machines, this model provides the network and storage infrastructures to users. Amazon EC2 [13], OpenStack [14], GoGrid [15], and Flexiscale [16] are some of the major Infrastructure-as-a-Service Cloud providers.

Cloud is composed of four deployment models [4]; 1) Private Cloud infrastructure is exclusively used by a single organization comprising multiple users, 2) Community Cloud is exclusively used by a specified community of users from different organizations who share common concerns and objectives, 3) Public Cloud is provisioned for open use by general public, and 4) Hybrid Cloud is comprised of two or more Cloud infrastructures (i.e., private, public, or community).

1.1.1 Virtualization on Cloud

Clouds computing is comprised of an additional layer of virtualization as compared to Grid computing [1]. The virtualization provides an execution, management, and hosting environment for application services offered by Cloud. The virtualization technology is an effective platform for the in-time need-based provisioning, administration, and management of resources. The physical resources are provided to the user through virtualization in the form of virtual machines. The services on a compute Cloud are a combination of several registered computing, storage, memory, and bandwidth resources collected in the form of VMs.

Fig. 1.2 shows the abstraction layer of Cloud computing that comprises system accessibility, virtual instance, and physical instance layers of Cloud. The system accessibility layer provides a user-friendly interface to Cloud users for interaction and submission of the HPC jobs on Cloud. The physical instance layer is comprised of host machines, storage servers, and other networking infrastructure in the Cloud datacenters. These resources are provisioned to Cloud users through virtualization. The VMs are hosted on physical computing machines in the datacenters and the virtualization made it possible to reallocate the VMs to physical machines based on the changing workload in a dynamic manner. Datacenters have a highly dynamic and versatile number of resources; therefore, the VMs may dynamically be adapted as per need to achieve better performance with improved resource utilization. However, the load balancing should be ensured by proper mapping of physical resources to virtual machines for better resource usage [17].

1.2 Scheduling on Compute Cloud

Scheduling is a mapping mechanism that comprises of tasks submission to computing resources for execution and the selection of appropriate resources. Scheduling in Cloud computing is generally performed at two levels; one is the mapping of VM to physical machines and the second is mapping of jobs to VMs. The objective of job scheduling is to accomplish high throughput with minimal completion

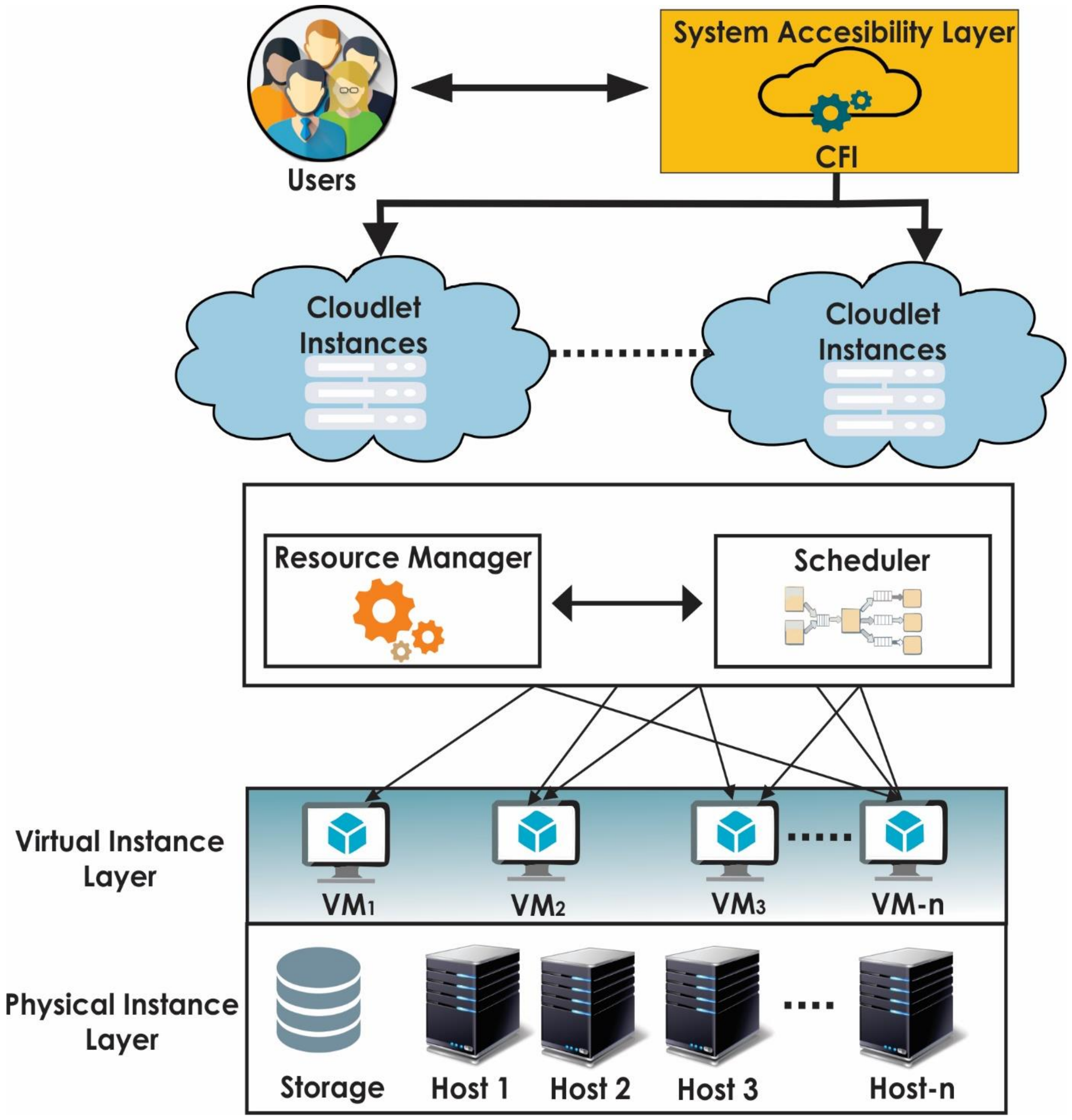


FIGURE 1.2: Cloud Computing System Architecture.

time and high resource utilization. One important aspect in the job scheduling is to maintain the load balancing to maximize the resource utilization and minimize the execution time without affecting the Cloud services. Load balancing in Cloud is a way to distribute workloads among computing resources to achieve optimal resource utilization. It is the distribution of workload among resources by avoiding the resources to be loaded with more workload than the other being idle or assigned with lesser jobs for execution. Load balancing ensures the consistent load distribution by reducing the total average time needed for the completion of workload execution. There are many metrics used to determine the performance of load balancing and job scheduling algorithms. These are scalability, resource utilization, response time, throughput, migration time, makespan, and scheduling overhead associated. In the Cloud environment, scheduling is employed at two levels, as shown in Fig. 1.2:

1. **VM scheduling** is concerned with the mapping of virtual machines to physical hosts in a Cloud data-center (i.e., virtual instance layer to physical instance layer mapping); and

2. **Job scheduling** is concerned with the assignment of Cloud jobs to virtual machines (i.e., Cloudlet instances to virtual instance layer mapping, where Cloudlet represents a user job in Cloud environment [1]).

Cloud scheduling strategies could be classified as either static or dynamic.

1.2.1 Dynamic Scheduling

The dynamic strategies consider the current state of computing resources while dispatching the workload. These strategies work according to dynamic changes in the state of computing resources. Generally, a status table is maintained to store the states of the Cloud resources.

A few of dynamic schedulers witnessed in literature review are *Elastic Cloud Max-Min* (ECMM) [18, 19] and Enhanced Bee Colony algorithm [20]. A dynamic scheduling technique generally relies on the run-time parameters to schedule jobs in a best-effort and greedy manner. The dynamic strategies can be classified into two types: online and batch [21]. Online dynamic strategies deal with the scheduling of a single job, whereas the batch dynamic scheduling strategies are concerned with the mapping of a batch of jobs. Moreover, job schedulers can be categorized as preemptive and non-preemptive algorithms. In case of preemptive algorithms, a specific job is either being processed by computing resources or being queued for execution maybe reassigned to the other compute resources for its completion. On the other hand, the non-preemptive algorithm assigns a specific job to computing resources on its arrival and is not considered for reassignment.

1.2.2 Static Scheduling

A static strategy produces jobs to VMs mapping in advance and executed at once. Therefore, the computing resources along with its properties are known in advance. The main advantage of these algorithms is their ability to generate high-quality schedules by making global pre-calculation and comparisons of different solutions before choosing the best suited one [5].

Some of the prominent static algorithms are *Opportunistic Load Balancing* (OLB) [22–24], Min-Min [25–27], Max-Min [18, 25, 28], *Resource-Aware Scheduling Algorithm* (RASA) [29, 30], Sufferage [25, 31] and *Minimum Completion Time* (MCT) [24, 25, 32] strategies.

1.2.3 Static Scheduling Challenges

Scheduling techniques are very crucial in attaining the benefits offered by Cloud. There are several prominent Cloud scheduling algorithms [22–25, 29–39] presented in the literature. However, the selection of an appropriate scheduler according to a given environment (i.e., jobs and computing machines heterogeneity) to achieve desired scheduling objectives (such as reduced makespan, higher throughput, etc.) is a challenging task. Since each heuristic contains different underlying assumptions; therefore, a precise comparison cannot be made. Therefore, the scrutiny and empirical examination of state-of-the-art scheduling algorithms need to be conducted on the same test-bed with instances of similar benchmark datasets to identify the potential strengths and weaknesses of the scheduling algorithms.

Most of the static scheduling algorithms produce a good turnaround time and irrefutable QoS, while many producing a very poor resource utilization. *Minimum Execution Time* (MET) algorithm [24, 25] assigns jobs to the resource producing shortest execution time for every job regardless of currently assigned workload to the resource. MET produces a severe load-imbalance in the computing system and directly not applicable in Cloud environment. Using *Random Selection* (RS) [31, 33], *Round Robin* (RR) [31, 33, 34] and OLB [23, 24, 33] techniques, most of the resources remain idle that results in a very poor makespan and resource utilization due to its simple scheduling without considering the resource-awareness. The MCT [25, 32, 40], Min-Min [22, 25, 38, 41] and Max-Min [18, 25, 30, 41] are comparatively better than the MET [24, 25], RR [31, 33, 34], RS [31, 33] and OLB [23, 24, 33] algorithms. Min-Min performs better for small-size jobs and may cause delays in the execution of large-size jobs. However, Min-Min overloads the faster VMs with more number of small-size jobs while the slower VMs are assigned with fewer number of large-size jobs. In contrary, Max-Min [18, 25, 30, 41] resolves the problem by favouring large-size jobs and avoiding its longer response time. Behaviour of these algorithms badly affects the resource utilization on Cloud system. Sufferage algorithm calculates the sufferage-value that results in higher scheduling overhead while producing better resource utilization. Different variants of Min-Min [22, 25, 38, 41] and Max-Min [18, 25, 30, 41] techniques are designed in the form of RASA [29, 30, 42], *Task-Aware Scheduling Algorithm* (TASA) [39, 43], *Load Balance Improved Min-Min* (LBIMM) [37], *User-Priority-Aware Load Balance Improved Min-Min* (PA-LBIMM) [37], *Skewness-Based Min-Min Max-Min* (SBM^2) [28], *Proactive Simulation-Based Scheduling and Load Balancing* (PSSLB) [44], and *Proactive Simulation-Based Scheduling and Enhanced Load Balancing* (PSSELB) [44] etc. in the literature. In addition, SLA-MCT [45], SLA-Min-Min [45], Profit-MCT [45], and Profit-Min-Min [45] are also found in the review of existing state-of-the-art. These algorithms provide the desired scheduling either with the objective of improved makespan or minimal execution cost. However, most of the techniques do not provide a commendable load balance and resource utilization in Cloud.

1.3 Scheduling Objectives

The scheduling algorithms work based on some performance measures that are known as scheduling objectives. Various scheduling objectives are harnessed to determine the performance of job scheduling algorithms. The scheduling objectives could be either CSP or Cloud user oriented. The scheduling objectives desired by CSP in different scheduling strategies are known as the CSP oriented objectives such as makespan minimization [46–48], workload maximization, throughput [49, 50], energy consumption minimization [49, 51], VM utilization maximization [50, 52], maximization of reliability [46, 48], VM migration [53, 54], and server consolidation [55, 56] etc. On the other hand, the scheduling objectives desired by Cloud users for their workload execution are known as Cloud user-oriented scheduling objectives such as execution time minimization [46, 57], cost minimization [46, 48, 57], achieving deadline constraints [21, 49, 58], and response time minimization [59–61] etc. A few scheduling objectives are briefly described as follows.

Makespan:

Time taken by a compute Cloud to finish the execution of a complete batch of Cloud user jobs is known as makespan. The workload submitted by a Cloud user could be a workflow (i.e., collection of inter-dependents jobs) or collection of independent jobs. The scheduling algorithms are designed to either minimize makespan or defining a time limit for execution of workload, mentioned in SLAs.

Execution Cost:

The Cloud users pay for leasing the VM resources on a compute Cloud, being pay-as-you-go infrastructure. Execution cost is the agreed cost paid by a Cloud user for successful completion of computing services. The most common QoS parameter in Cloud is to minimize the execution cost and meet the user-defined deadline. This objective is considered in scheduling algorithms either by minimizing its value or meeting user-defined constraint on the cost, mentioned in SLAs [5].

Execution Time:

The amount of time taken by computing resources to execute a specified job is known as the execution time of a job. The sum of the execution time of all jobs in a workload is the total execution time of that workload on the computing system.

VM Utilization:

Idle time-slots within leased VMs in a Cloud often results in waste of money (as the resources paid for are not utilized). Generally, the idle time of VMs is not desired and should be minimized to increase the VMs utilization. The higher VM utilization will benefit the Cloud users in terms of cost, and execution time, while the CSPs in terms of profit, energy consumption, and optimal resource utilization.

Server Consolidation:

It is an approach for the efficient usage of physical machines (by keeping less number of machine active) and to lessen operational cost in CDCs. The virtualization technology in Cloud allows the consolidation of multiple HPC applications into virtual machines to be hosted on a single or multiple physical server. Therefore, the scheduling strategies (i.e., for VMs to *Physical Machines* (PMs) allocations) consider the server consolidation as a part of the scheduling objectives.

Deadline Constraints:

Cloud users submit either compute, memory or bandwidth-intensive HPC jobs with or without the defined-deadline in Cloud. The scheduling algorithms are developed to address such user-defined deadlines (i.e., as per SLAs) for the workload execution as a scheduling objectives.

Workload Maximization:

The scheduling algorithms are developed to address the workload maximization as one of the scheduling objectives. Workload maximization refers to the more amount of work done (i.e., the number of dependents such as workflows or independent jobs executed) by a computing system.

1.4 Datasets

Datasets are becoming increasingly more pertinent when executing the performance assessment of Cloud-scheduling, resource-allocation, and load-balancing algorithms used for eagle-eyed examination of efficiency and performance in a real-world Cloud. A minor change in the behavior and nature of the dataset is reflected in the performance of scheduling and resource-allocation policies. Assessing the scheduling and allocation policies on Cloud infrastructures under a varying load and system size is a challenging problem. Real Cloud workload is hard to acquire for performance analysis and investigation due to the users' data confidentiality and policies maintained by CSPs [62]. Therefore, the publicly available real Cloud dataset will significantly encourage other researchers to compare and benchmark their applications using an open-source benchmark. In addition, using real testbeds limits the experiments to the scale of the testbed. Hence, testing the accuracy performance with real-world datasets is crucial in the field of research, and synthetic data does not realistically represent an actual dataset [63]. The most appropriate alternative is to make the investigation in a simulation environment with a load of varying behavior in the Cloud environment. For Cloud computing research, it is valuable to formulate and ensure a widespread availability of realistic datasets that show how resourcefully the Cloud addresses the user requirements.

Based on the analysis of large-scale Google cluster traces of 29-days, a new benchmark GoCJ dataset [62] is proposed and presented in Chapter 3.

1.5 Motivation

Distributed computing comprises a plethora of heterogeneous resources interconnected over network to meet the requirements of diversified *High-Performance Computing* (HPC) applications [64]. Cloud computing is a business-oriented concept and a paradigm of scalable, heterogeneous, and distributed high performance computing vowed to deliver on-demand utility computing to users over the Internet [65, 66]. The computing resources are allocated in the form of virtual machines deployed on physical host machines-based *Cloud Data Centers* (CDC). The CDCs are generally over-provisioned to guarantee high service-availability and *Quality of Service* (QoS) computing [67]. These QoS contracts are formally negotiated and self-proclaimed in the form of SLAs to guarantee the user requirements related to execution time and cost constraints [68, 69]. However, inefficient resource allocation adversely affects resource utilization in Cloud and causes load imbalance that originates the problems of data and computing skewness, resulting in the straggler [70]. The stragglers are slow running jobs that can run up to 8 times slower than the median job on a production cluster that may increase the execution time of the other jobs up to 47% [71, 72]. In the case of stragglers in workload distribution, some of the computing resources remain idle or underutilized while waiting time increases for the jobs mapped on the busy machines. An increasing amount of stragglers in CDC diminishes the revenue of *Cloud Service Providers* (CSPs) in terms of execution gain [45]. A CDC provides virtualization for pragmatic resource management to enhance the system throughput and to reduce the computational cost and energy consumption [73].

Virtualization endures the *Virtual Machine* (VM) migration to meet the increasing demands in Cloud computing by relocating VMs on host machines in CDC. The VM migration accomplishes diverse objectives such as load-balancing on host machines, energy-aware scheduling, and fault tolerance etc. Typically, live VM migration technique [74] is employed to reduce the downtime of the overloaded VMs. However, it is a memory and cost-intensive process [75]. Consequently, different techniques [26, 76–78] have been proposed to mitigate the memory and energy consumption issues for VM migration. However, the VM migration persistently invades computing performance unless amalgamated with efficient techniques [67]. To mitigate these apprehensions and eludes the VM migration, the load balance in workload distribution should be incorporated in Cloud scheduling [26, 75–78]. Clouds are capable to provide an excellent solution to address the ever-increasing computation and storage demands of large scientific HPC applications. To attain good computing performances, mapping of Cloud jobs to the compute resources is a very crucial process. Today, several efficient Cloud scheduling heuristics [18, 24–26, 28, 31, 37, 38, 41, 44, 79] are available; however, selecting an appropriate scheduler for the given environment (i.e., considering the jobs and machines heterogeneity) and scheduling objectives (such as minimized makespan, higher throughput, increased resource utilization, load balanced mapping, and minimal execution cost etc.) is a challenging task. According to a survey [80], approximately 1.6 million tons of additional CO_2 emission are caused only due to the idle computing resources within the CDC. Moreover, the idle computing resources cost additional 19 billion dollars in terms of the consumed resources and energy cost. An idle and underutilized host machine in Cloud still consumes up to 70% of the energy required by an active machine [80].

In general, scheduling is deemed as a significant, challenging, and quintessential part of Cloud known as an NP-complete problem [24, 81–84] and particularly it must be addressed to achieve efficient, fair, and starvation-free performance in HPC environment. In general, the low scheduling overhead of static strategies adheres them to outperform over the dynamic scheduling strategies [40, 85, 86]. In addition, static scheduling algorithms avoid VM migration to prevent communication overheads to reduce the execution delays [87]. In addition, most of the static techniques produce admirable turnaround time and irrefutable QoS because of the pre-ensured availability of computing resources for the workload execution [33]. On the other hand, static algorithms usually suffer from poor resource utilization in case of resource oblivious mapping of workload [85]. Improper workload distribution in Cloud computing causes longer execution and more energy consumption due to which most of the scheduling heuristics are not efficient enough to harness the full capacity of available computing resources.

The users usually desire to find an immediate response time, the shortest possible execution time, and the lowest possible cost for the HPC applications on Cloud. On the other hand, CSPs require resources utilization on its maximum possible capacity with more revenue generation. Therefore, a crucial aspect of scheduling is to map Cloud jobs in a load balanced manner to reduce the makespan of a job pool (batch of submitted Cloud jobs). The load balanced mapping refers to a distribution of jobs (among VMs) so that all the VMs accomplish the execution of the assigned workload approximately in same time duration. Importantly, a balanced load ensures higher system throughput, lower execution time, and improved resource utilization for a job pool. Efficient job scheduling mostly increases the user's satisfaction, and improves the system utilization. Since each algorithm contains different underlying assumptions; therefore, a precise comparison is a tedious task. In this regard, we empirically scrutinize and experimentally compared state-of-the-art static scheduling heuristics employing the similar computing setups using both the synthetic and other Cloud benchmark workloads.

In this thesis, the workloads used for empirical-based investigation is referred to a collection of independent, non-preemptive, and compute-intensive jobs that are submitted on Cloud. Unlike workflows, the jobs in these workloads are without inter-job data dependencies.

1.5.1 Research Questions

State-of-the-art scheduling techniques and strategies are empirically investigated to provide the scheduling of jobs on VMs in Cloud computing. Primarily, this section describes four broad research questions addressed in this thesis:

1. How to formulate and make a benchmark dataset publicly available for Cloud computing that reflects the jobs-size behavior of a real Cloud?
2. How to ensure efficient provisioning of computing resources and job execution with improved throughput and minimal makespan?
3. How to minimize the load imbalance in workload distribution to guarantee an improved resource utilization on a compute Cloud?
4. How to ensure improved resource utilization in SLA-aware (cost-efficient) scheduling on a compute Cloud?

1.6 Research Contributions

In light of the motivation and challenges outlined in the previous sections, the major research contributions of this PhD dissertation are summarized and presented as follows:

1. A critical analysis and synthesis of several existing static Cloud scheduling algorithms to identify the pros and cons of these techniques. In-depth performance analysis (in terms of turnaround time, resource utilization, and throughput) and empirical assessment of the existing static scheduling algorithms using two synthetic and one benchmark GoCJ realistic workloads is provided using CloudSim simulator [1]. This empirical investigation identify the potential research directions that could assist the scientific community to cope with the challenges pertaining to the static scheduling algorithms for Cloud computing (It has been discussed in detail in Section 2.5.4.);

2. A benchmark GoCJ dataset is proposed that provides a reflection of real workload behaviour as perceived in Google cluster traces [88–93] and MapReduce logs [92] from the M45 supercomputing cluster (Section 3.1 elaborates this in detail.);

3. A novel load balancing scheduler (named RALBA) to schedule pool of non-preemptive, compute-intensive, and independent Cloud jobs that produces reduced makespan, increased resources utilization, and higher throughput (The detail of RALBA can be found in Section 3.2 and Section 5.1.);

4. In-depth empirical investigation (using CloudSim simulator) to critically contemplates the performance analysis of 9 prominent static scheduling algorithms (including the proposed RALBA technique) using 9 instances of two benchmark datasets (i.e., GoCJ [62, 94], and *Heterogeneous Computing Scheduling Problem* (HCSP) instances [24, 95]) using different computing architectures [1] (Section 5.2 elaborates this in detail.);

5. Cost-efficient and resource-aware scheduler for SLA-based compute-intensive, non-preemptive, and independent jobs that produces an improved balance among execution time, execution cost, and resource utilization. The SLA-RALBA performance is evaluated compared to the state-of-the-art SLA-aware scheduling algorithms using two benchmark GoCJ [62, 94] and HCSP [24, 95] datasets (The detail of SLA-RALBA can be found in Section 3.3 and Section 5.3.).

Table 1.1: Research Contributions.

Problem	**Contribution**	**Features**
Research Question - 1	GoCJ Dataset [62]	- Benchmark dataset for Cloud Computing. - Workload behavior based on the real Google cluster traces of 29-days. - Publically available on Mendeley Repository and MDPI Data journal in the Supplementary Materials.
Research Question - 2 and Research Question - 3	RALBA Scheduler [50]	- Resource-aware scheduling. - Provides Load-balanced scheduling with improved resource utilization. - Provides improved load balancing at machine-level during workload execution. - Provides job scheduling with improved makespan and increased throughput.
Research Question - 4	SLA-RALBA Scheduler [96]	- Resource-aware and SLA-aware scheduling. - Provides cost-efficient job scheduling with improved resource utilization. - Provides scheduling with improved balance among resource utilization, execution cost and execution time.

Table 1.1 shows the association of research questions and contributions in this thesis along with the features of each research contribution. While, Table 1.2 presents the mapping of publications (made out of research work carried out for this thesis) against research contributions.

1.7 Thesis Organization

The core chapters of this thesis are structured as shown in Fig. 1.3. The chapters of the thesis are organized as follows:

Chapter 2 presents the literature review by providing a critical analysis of the state-of-the-art Cloud scheduling algorithms, SLA-aware Cloud scheduling algorithms, and the existing datasets in the literature. The chapter presents a detailed empirical investigation to highlight the research issues in Cloud computing.

TABLE 1.2: Mapping Publications on Research Contributions.

Journal	**Research Contribution**
Computing and Informatics [97]	This publication presents the empirical investigation conducted to evaluate ten existing scheduling algorithms and to highlight the research gaps in Cloud Computing.
MDPI Data [62]	This publication proposes and presents a new GoCJ dataset based on real workload behavior (Google cluster traces [88–91, 93]).
Cluster Computing [50]	This publication proposes and presents a novel resource-aware scheduler known as RALBA that provides improved resource utilization.
IEEE Access [52]	This publication presents a rigorous investigation of existing eight scheduling algorithms against RALBA in terms of machine-level load balancing and resource utilization in Cloud Computing.
The Journal of Supercomputing [96]	This publication proposes and presents a novel SLA and resource-aware scheduler known as SLA-RALBA that provides a drastic increased in resource utilization, while providing improved balance between execution time and cost.

Chapter 3 is dedicated to propose a new benchmark dataset in the form of a GoCJ dataset for distributed and Cloud computing. A novel resource-aware scheduling mechanism (i.e., RALBA scheduler) that is capable of mapping users jobs in Cloud environment in a balanced manner is proposed and presented. A second novel cost-efficient and resource-aware scheduling mechanism (i.e., SLA-RALBA scheduler) that provides jobs to virtual machines mapping with an improved balance among execution time, cost and resource utilization is proposed and described. In addition, the system architecture algorithms and complexity of proposed algorithms are presented.

Chapter 4 presents the experimental environments (simulation setup and VMs composition in computing environment) and the datasets employed in conducting the experimentation of RALBA, VM-level investigation of RALBA as compared

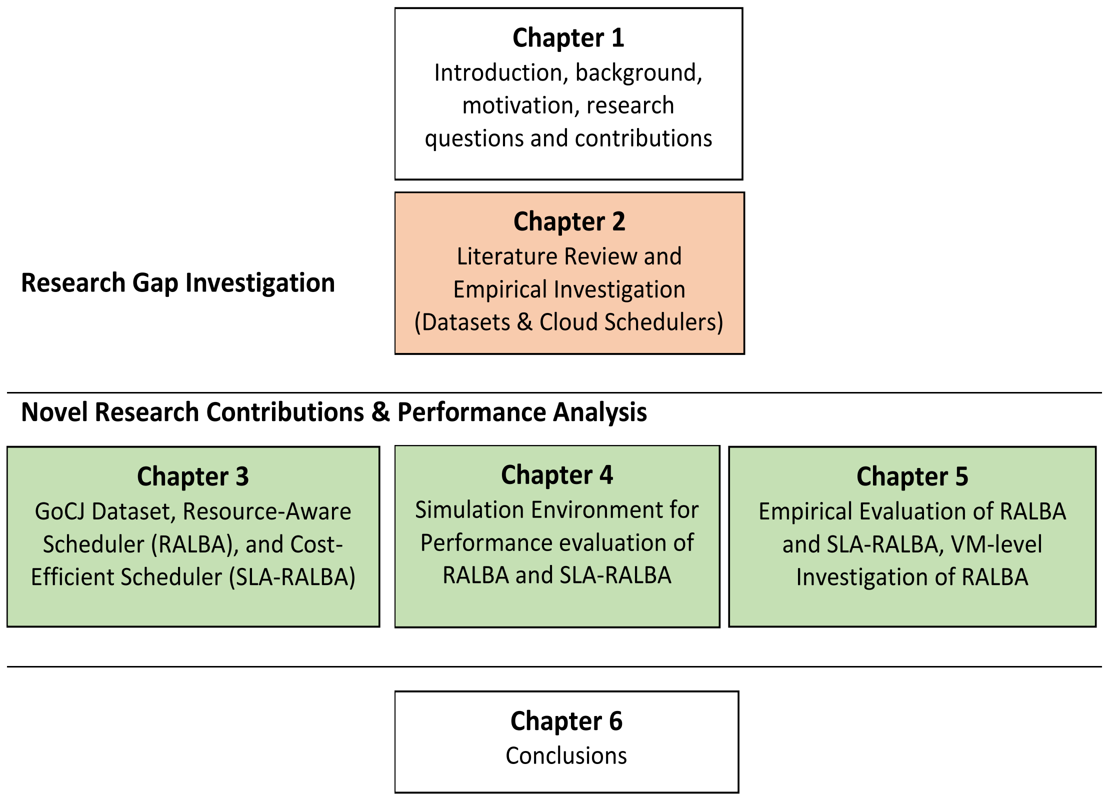

Chapter 1
Introduction, background, motivation, research questions and contributions
Chapter 2
Literature Review and Empirical Investigation (Datasets & Cloud Schedulers)
Research Gap Investigation
Novel Research Contributions & Performance Analysis
Chapter 3
GoCJ Dataset, Resource-Aware Scheduler (RALBA), and Cost-Efficient Scheduler (SLA-RALBA)
Chapter 4
Simulation Environment for Performance evaluation of RALBA and SLA-RALBA
Chapter 5
Empirical Evaluation of RALBA and SLA-RALBA, VM-level Investigation of RALBA
Chapter 6
Conclusions

FIGURE 1.3: Thesis Organization.

to the existing stat-of-the-art scheduling algorithms, and in the experimental evaluation of SLA-RALBA against state-of-the-art SLA-aware scheduling.

Chapter 5 is dedicated to present the performance analysis of RALBA in terms of makespan, throughput, and resource utilization. The Chapter presents the empirical investigation of 09 scheduling algorithms including RALBA (i.e., using the instances of two benchmark datasets) in terms of resource utilization, in addition to a new performance metric i.e., VM-level load balancing. The performance evaluation of SLA-RALBA is presented in terms of makespan, throughput, and resource utilization using 15 instances of two benchmark datasets with additional consideration of execution cost.

Chapter 6 concludes the thesis, summarizes the findings, and provides directions for future work.

Chapter 2

Literature Review and Empirical Investigation

This chapter presents the literature review and provides a comprehensive understanding of the inherited mechanism of state-of-the-art algorithms by conducting an empirical investigation of ten renowned Cloud scheduling techniques. The experiments are conducted using three datasets: 2 synthetics and 1 benchmark datasets.

2.1 Overview

Compute Cloud comprises a distributed set of HPC machines to stipulate on-demand computing services to remote users over the Internet. Clouds are capable enough to provide an optimal solution to address the ever-increasing computation and storage demands of large scientific HPC applications. To attain good computing performances, mapping of Cloud jobs to the compute resources is a very crucial process. In the current are, several efficient Cloud scheduling algorithms are available, however, selecting an appropriate scheduler for the given environment (i.e., jobs and machines heterogeneity) and scheduling objectives (such as minimized makespan, higher throughput, increased resource utilization, load balanced mapping, etc.) is still a difficult task. In this chapter, we consider ten important scheduling algorithms (i.e., OLB, PSSLB, PSSELB, MCT, Min-Min, LBIMM, Max-Min, RASA, RASA, and Sufferage) to perform an extensive empirical study to insight the scheduling mechanisms and the attainment of the major scheduling objectives. This study assumes that the Cloud job pool consists of a collection of independent and compute-intensive tasks that are statically scheduled to minimize the total execution time of a workload. The experiments are performed on CloudSim simulator using two synthetic and one benchmark GoCJ [62, 94] workloads. This empirical study presents a detailed analysis and insights into the circumstances requiring a load balanced scheduling mechanism to improve overall execution performance in terms of makespan, throughput, and resource utilization. The outcomes have revealed that the Sufferage and TASA algorithms produce minimum makespan for the Cloud jobs. However, these two scheduling algorithms are not efficient enough to exploit the full computing capabilities of Cloud virtual machines.

In an empirical analysis, Syed Hamid Hussain Madni et al. provides the investigation of *First Come First Serve* (FCFS), MET, MCT, Min-Min, Max-Min, and Sufferage scheduling algorithms [25]. Based on their analysis, Syed Hamid Hussain Madni et al. concluded that the hybridization of these techniques may result in more improved results and overcome the limitations of each other to achieve the optimization of task scheduling in Cloud computing [25]. Therefore, in this research work, some hybridized scheduling techniques (i.e. RASA [29, 30], TASA [39], LBIMM [37, 44], PSSLB [44], and PSSELB [44] algorithms) are also considered for performance investigation of resource utilization in cloud computing. Fig. 2.1 shows ten scheduling algorithms; where the techniques on the tail of each arrow are the modified and hybridized techniques based on the scheduling techniques directed by the arrow symbols (i.e., the mechanism of RASA is based on Max-Min and Min-min, the mechanism of TASA is based on Sufferage and Min-Min, LBIMM is the modified version of Min-Min, PSSLB is the modified versions of Max-Min, and PSSELB is the modified version of PSSLB technique). In this

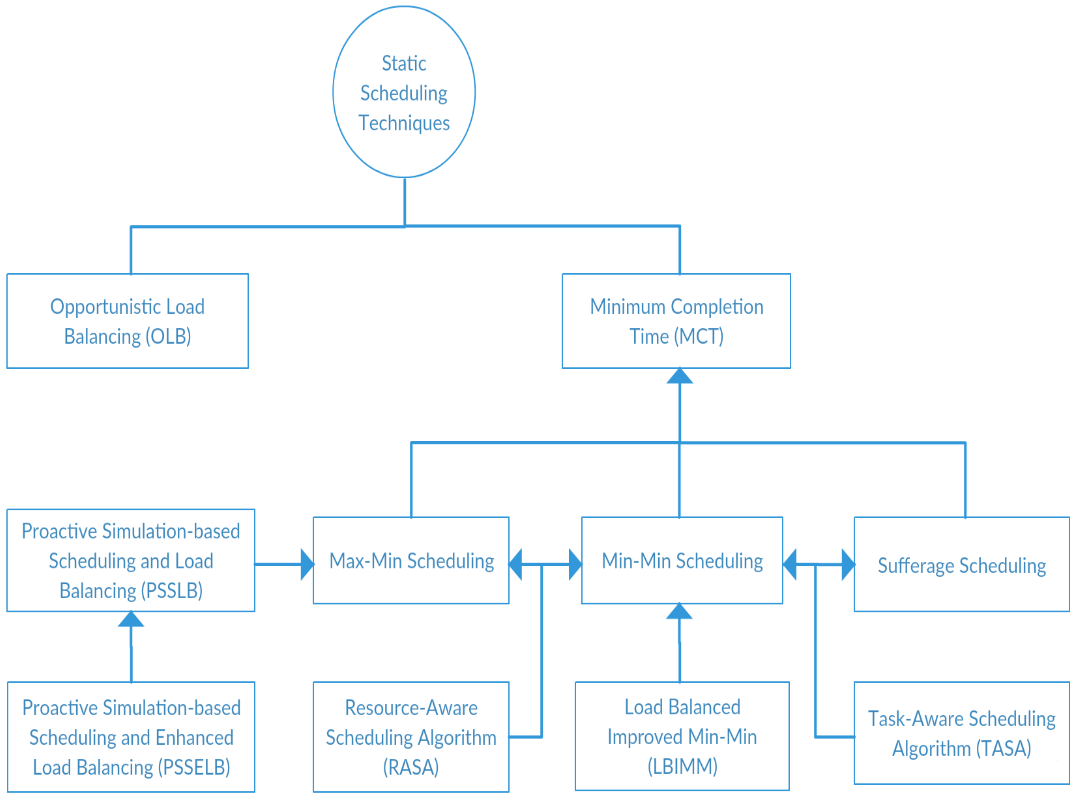

Static Scheduling Techniques
Opportunistic Load Balancing (OLB)
Minimum Completion Time (MCT)
Proactive Simulation-based Scheduling and Load Balancing (PSSLB)
Max-Min Scheduling
Min-Min Scheduling
Sufferage Scheduling
Proactive Simulation-based Scheduling and Enhanced Load Balancing (PSSELB)
Resource-Aware Scheduling Algorithm (RASA)
Load Balanced Improved Min-Min (LBIMM)
Task-Aware Scheduling Algorithm (TASA)

FIGURE 2.1: Hybridization of Cloud Scheduling Algorithms.

study, we consider the following assumptions for the empirical-based comparison of the employed scheduling heuristics. One such assumption is that a workload is referred to as a collection of independent and compute-intensive tasks (without inter-task data dependencies). The mapping of these tasks is performed statically to minimize the scheduling overhead and to evade job migrations [24]. This empirical study provides a detailed analysis of the scheduling algorithms and insights of the scheduling mechanisms where a higher throughput and reduced execution time is attained; however, a considerable room of improvement is observed in terms of resource utilization. We argue that the existing state-of-the-art static scheduling heuristics should address the load-balancing issue to attain exquisite resource utilization. Near-Optimal resource utilization will produce higher throughput, reduced execution time (for the Cloud job pool), and energy efficient execution.

In summary, we present an analysis of the resource utilization of virtual resources in terms of workload distribution among all the VMs. The empirical investigation reveals that most of the scheduling algorithms are not efficient enough to exploit the full computing capabilities of Cloud virtual machines. This empirical study has highlighted various pressing research gaps that must be overcome to improve the scheduling performance and to reduce cost at Cloud service provider level. Major contributions presented in this chapter are:

1. A critical analysis and synthesis of existing datasets in the literature to identify the the pros and cons of each dataset and realize the need for a new benchmark dataset that should publicly be available for Cloud research community;
2. A critical analysis and synthesis of existing state-of-the-art static Cloud scheduling algorithms to identify the pros and cons of each algorithm;
3. A critical analysis and synthesis of existing SLA-aware static Cloud scheduling algorithms to identify the pros and cons of each algorithm;
4. In-depth performance analysis (in terms of turnaround time, resource utilization, and throughput) and empirical assessment of existing static scheduling algorithms using two synthetic datasets and one realistic benchmark dataset for Cloud and distributed computing [62];

2.2 Existing Datasets

The developers of resource-allocation and scheduling algorithms share test datasets (i.e., benchmarks) to enable each other to compare the performance of newly developed algorithms. However, mostly it is hard to acquire real Cloud datasets due to the users' data confidentiality issues and policies maintained by CSP [62]. Accessibility of large-scale test datasets, depicting the realistic high-performance computing requirements of Cloud users, is very limited. The existing datasets (i.e., HCSP [24, 95], and *Task Execution Time Modeling* (TETM) [98] Instances) are publicly available for research purposes and these datasets are based on varying task and machine heterogeneity. However, the compositions of the job sizes

Table 2.1: Summary of Existing Datasets.

Feature	**Description**
HCSP Instances [24]	HCSP instances is a standardized benchmark-based on a range-based method to produce ETC matrices with a variation of heterogeneity in the tasks to be executed and machines in the system. Likewise, small size HCSP instances (up to 1024 tasks and 32 machines) are provided to direct download, while for the larger instances a generator program along with the seeds used for the random number generator (execute the generator program using the correspondent seeds to replicate the instances) is provided.
TETM Instances [98]	To simulate a heterogeneous computing environment, a coefficient-of-variation-based technique to produce ETC matrices with a variation of heterogeneity in the tasks to be executed, and computing machines in the distributed system is presented. The model is capable of generating the dataset to evaluate the performance of scheduling heuristics. It generates a dataset reflecting the required task and machine heterogeneity; however, the TETM dataset does not reflect the realistic workload behavior i.e., the workload behavior on Google cluster traces or real compute Cloud etc. Similar to GoCJ, any number of tasks can be created with the TETM method.
GWA-T traces [99]	The dataset contains the performance metrics of 1750 VMs from distributed datacenters of Bitbrains, which is a service provider of hosting and business computation. The clients of Bitbrains are many major banks, credit card operators, and insurers etc. GWA-T traces focus on performance metrics of VMs; however, the GoCJ exhibits the jobs sizes and behaviors in a workload.
Facebook Hadoop Workload [100]	The workload is based on Hadoop traces on 600 machine cluster on Facebook spans over 6 months duration from May 2009 to October 2009 containing 1 million jobs. The jobs in the workload are recorded with submit time and inter-job_submit_gap parameters.

in these datasets are not derived from any real cluster or Cloud workload. Furthermore, there are some publicly available real workload traces (such as Google cluster traces [101], Yahoo cluster traces [102], Facebook Hadoop workload [100], OpenCloud Hadoop workload [103], Eucalyptus IaaS Cloud Workload [104], and GWA-T traces [99], etc.); however, most of these require pre-processing and in-depth low-level details to be regenerated and used for experimentation. The HCSP instances and TETM dataset are used in research work [91–93]. However, HCSP and TETM datasets are not based on compute-traces of any real Grid or Cloud system. Similarly, GWA-T traces exhibit the performance metrics of VMs in the datacenters instead of jobs behavior. Some existing datasets (i.e., HCSP [95], TETM [98], and GWA-T instances [99]) are tabulated in Table 4.2.

2.3 Cloud Scheduling Algorithms

This section presents the state-of-the-art Cloud scheduling and SLA-aware Cloud scheduling heuristics which are found in the existing literature.

2.3.1 State-of-the-art Cloud Scheduling Algorithms

In this section, the working of state-of-the-art static Cloud scheduling heuristics (i.e., RS [31–33], RR [31, 34], MCT [25, 35], OLB [23, 24, 33], Min-Min [23, 32, 36, 37], LBIMM [37, 44], Max-Min [22–24], Sufferage [23, 31, 38], RASA [29, 42], PSSLB [44], PSSELB [44], and TASA [39]) are is delineated below.

RS scheduling allocates jobs to VMs without considering the current load of VMs [31–33]. RS has a minimal scheduling overhead, simple implementation, and low complexity compared to other static Cloud scheduling algorithms [34]. However, RS arbitrarily assigns a job to a randomly selected VM. However, the unfair scheduling mechanism employed by RS may lead to the selection of an overloaded VM that could cause the load-imbalance, low resource utilization, and a long waiting time for the submitted jobs [31, 32].

RR scheduling mechanism distributes Cloud jobs over the available VMs in a circular order [31, 34]. The equitable scheduling of RR results in a mapping of an almost equal number of jobs to VMs regardless of the job sizes [31, 34]. RR has minimal scheduling overhead as compared to the other scheduling techniques [31, 32, 79]. However, the assignment of small jobs to the faster resources and large jobs to the slower resources originates the longer makespan, poor resource utilization, and load imbalance [32, 79].

OLB algorithm [23, 24, 33, 34] assigns each job, in an arbitrary order, to the next available machine regardless of considering the job's execution time on that particular machine. OLB is a simple scheduling scheme having low scheduling overhead and complexity. A major scheduling objective of the OLB scheme is to make all the Cloud machines as busy as possible [24]. However, OLB scheduling algorithm mostly results in poor makespan because it is not resource-aware.

MCT technique assigns the candidate job to a VM producing minimum completion time for it [22, 24, 33, 38]. The current load of a VM is employed to identify appropriate VM for assignment of the job [24, 32]. At each scheduling step, MCT algorithm has to scan all the available VMs to find the machine producing minimum completion time for a candidate job that causes significant scheduling overhead. On the other hand, the MCT algorithm produces improved makespan and resource utilization compared to RR and RS techniques [35]. Another concern of the MCT is that it assigns more jobs to faster VMs that leads to load imbalance [22, 79, 105].

Min-Min algorithm is based on the MCT mechanism [24, 29, 35, 38]. The functionality of this algorithm follows two steps; firstly, the earliest finish time of all jobs is determined by considering all VMs. In the second step, the job with the minimum earliest finish time is selected and assigned to the concerned VM. On each scheduling decision, the ready time of the candidate VM is updated and this process continues until all the jobs have been scheduled on the VMs. Min-Min heuristic favors small-sized jobs (i.e., schedule first) and penalizes (i.e., resulting in delayed response time) large-sized jobs [32, 105, 106]. Min-Min often results in low resource utilization for a job pool based on fewer large-sized jobs [22, 31, 35]. Moreover, Min-Min overloads faster VMs with small-sized jobs, while the slower VMs are allocated with fewer but large-sized jobs that lead to load imbalance. Therefore, the large-sized jobs mapped on slower VMs often results in longer makespan [26].

Max-Min has a resemblance to Min-Min. Both algorithms have a divergent job selection policy but the functionality of both algorithms is almost identical. In

Max-Min, the first step is identical to Min-Min but in the second step, the job with the maximum earliest finish time is selected and assigned to the concerned VM [23, 38]. On each scheduling decision, the ready time of the VM is updated and the process is repeated until all the jobs have been scheduled. Max-Min favors larger jobs by producing minimum completion time; on the other hand, Max-Min penalizes smaller size jobs [23, 27, 33, 106]. Moreover, Max-Min produces load imbalance for a job pool with larger size jobs [27, 84]. To avoid longer response time (for the larger jobs), Max-Min scheduling heuristic selects larger jobs to be executed early [31, 35]. Max-Min heuristic mostly performs better in the scenario when there are a large number of small-sized jobs with a few larger size jobs [37, 38]. The inherent mechanism of both Max-Min and Min-Min heuristics adversely affects the resource utilization in Cloud (as evident in our experimental results presented in Section 2.5).

Sufferage algorithm [28, 34, 79] computes the sufferage-value for each job by finding the difference between its MCT and the second MCT (higher than the first MCT) generated by any VM. The job with the largest sufferage-value is assigned to the VM producing minimum completion time for it. Sufferage produces minimal makespan; but it has a consequential scheduling overhead due to a large number of calculations (to compute sufferage-value) in each scheduling decision [50]. Mostly, Sufferage produces minimal makespan compared to RS, RR, MCT, Min-Min, and Max-Min [29, 34].

RASA [27, 106] contemplates Min-Min and Max-Min alternatively to utilize the merits of both algorithms. RASA was originally proposed for Grid computing. First, RASA develops a job-related resource-matrix that contains the completion time of each job on all the resources. For the mapping of a first job, Min-Min algorithm is employed if numbers of jobs are odd; otherwise, Max-Min is used for mapping the jobs. After that, all the remaining jobs are scheduled using one of the two algorithms (i.e., Min-Min and Max-Min) alternatively. RASA provides fair scheduling for both large and small size jobs [27, 29, 34] . On the other hand, RASA generates load imbalance [31] and penalizes smaller jobs if the workload contains a large number of big jobs [23, 45].

TASA favors the smaller jobs in one scheduling step using Min-Min and finds an appropriate VM for the job by using Suffrage in the second scheduling step [39]. In most of the cases, TASA generates better makespan and resource utilization as compared to other scheduling techniques such as Min-Min, Max-Min, RASA and OLB [39].

LBIMM [37, 44] allocates jobs to VMs based on Min-Min technique in the first phase. In the next phase, LBIMM finds the smallest job on the most loaded VM and determines its completion time on the other VMs. After that, the minimum completion time of that job is compared with makespan. If the minimum completion time is less than the makespan, the job is allocated to that new VM and the ready time of both VMs are modified. This procedure is repeated for the next smallest job on the most loaded VM too. The process is repeated until there is no other VM that has the completion time for the smallest job on the heavily loaded VM as compared to makespan. This technique shares a load of heavy VMs with the idle or under-utilized VMs. LBIMM produces better makespan and load balancing than Min-Min algorithm.

PSSLB and PSSELB [44] are proposed to assign the large-sized jobs to the machines that can execute them faster than the other computing machines. PSSLB finds the matrix (i.e., each row has a completion time of a specified job on all machines) of completion time of each job on each machine. The matrix is sorted in a way that the last column stores minimum completion time for each job. Therefore, the longest job on the last column is selected and assigned to a machine producing minimum completion time for it.

PSSELB [44] is the modified version of PSSLB that produces a load balanced schedule. The largest job among the unallocated jobs is assigned to the machine using PSSLB, and completion time of this job is considered as a pivot. After that, the jobs that produce equal to or minimum completion time (i.e., on other machines) than the pivot are iteratively determined and assigned to the concerned machines (i.e., producing MCT for a job equal to or minimum than the pivot value). Next, the largest job is assigned to the concerned machine using PSSLB, and the pivot is updated with the completion time of the largest job. Again, the jobs with MCT on other machines (i.e., except the machine with last largest job assigned) that is equal to or less than the pivot are determined and assigned to the concerned machine. This scheduling procedure is repeated until all the unallocated jobs are assigned to machines in the same way.

SBM^2 technique is proposed by Panda et al. [28] for scheduling of skewed size workload on Grid. If the dataset contains a large number of shorter jobs with a few longer jobs then the dataset is referred to as positively skewed. On the other hand, if the dataset is comprised of a large number of longer jobs with a few shorter jobs then the dataset is known as negatively skewed [28]. SBM^2 computes the skewness of the workload in each scheduling decision. Min-Min is employed if the workload is negatively skewed; otherwise, Max-Min is adopted for the scheduling of a positively skewed workload. **PA-LBIMM** algorithm is proposed in [37]. PA-LBIMM categorizes both the jobs and resources in VIP and normal classes. First, the VIP jobs are scheduled on VIP resources using Min-Min. After that, PA-LBIMM schedules normal jobs on all resources using Min-Min.

Table 2.2 presents the summary of strengths and weaknesses of the scheduling algorithms in the literature.

2.3.2 SLA-Aware Cloud Scheduling Algorithms

To date, various studies have been conducted that focus on budget, user priorities, execution time, response time, deadline, and SLA-violations as QoS requirements for Cloud resource provisioning and scheduling.

Garg, Saurabh Kumar, et al. proposed *Mixed Workload-Aware Policy* (MWAP) [85] that is an admission control and scheduling to enhance the resource utilization and profit of Cloud datacenters guaranteeing the QoS needs of Cloud users as per SLA with auto-scaling support. MWAP used Artificial Neural Network to forecast the future computing utilization of in the datacenters. Philipp Leitner et al. proposed SLA-aware scheduling technique to reduce the expenses on Cloud resources

Table 2.2: Summary of Cloud Scheduling Algorithms.

Algorithms	**Strengths**	**Weaknesses**
RS [31–33]	Minimal scheduling overhead [79], low complexity [34],	Poor resource utilization [31, 32], load-imbalance [31, 32]
RR [31, 34]	Fairness in scheduling [31, 79], minimal overhead [22, 34]	Generally poor makespan [33], load-imbalance [32, 79]
OLB [22–24]	Low complexity, Minimal overhead, keep machines busy [24, 25, 32]	Poor resource utilization [34], load-imbalance [34]
MCT [24, 25, 32]	Improved makespan than OLB [24], machine-aware scheduling [26]	Load-imbalance [22, 38], overloads faster VMs
MinMin [25–27]	Favors smaller jobs [29, 42], reduced makespan for smaller jobs [32, 33]	Overloads faster VMs with smaller Jobs [29], penalize larger jobs.
LBIMM [37, 44]	Improved makespan and resource utilization than Min-Min [37]	A few smaller jobs are penalized while rescheduled [37]
PA-LBIMM [37]	Improved makespan for VIP jobs [37], priority-aware scheduling	Penalize normal jobs due to VIP jobs, relatively increased makespan than Min–Min [37]
MaxMin [18, 25, 28]	Favors larger jobs [33, 107], reduced makespan for larger jobs [106].	Penalize smaller jobs [39], load-imbalance for job pool with more larger jobs [29].
RASA [29, 30]	Fair treatment of larger and smaller jobs [31].	Penalize smaller jobs in dataset with more larger jobs [26]
Sufferage [25, 31]	Improved makespan than MCT, Min-Min, and Max-Min [18], job allocation to appropriate VM [106].	High scheduling overhead due to sufferage-value calculation [23].
TASA [39]	Improved makespan than Max-Min, Min-Min and RASA [39].	Load balancing is not considered [39].
PSSLB [44]	Reduces completion and response time for larger jobs [44]	Penalizes smaller jobs [44]
PSSELB [44]	Improved makespan than PSSLB [44]	Load-imbalance compared to PSSLB [44]
SBM^2 [28]	Fair treatment for positively and negatively skewed dataset, low makespan than Min–Min and Max–Min [28]	Heap of calculation in finding positive and negative skewness in each scheduling step [28], Load-imbalance

and loss incurred as SLA-violations in an IaaS Cloud [108]. This cost-efficient technique ensures to reduce the cost consumption of CSP on resource management of the Cloud. Alrokayan et al. proposed a cost and SLA-aware *Branch and Bound for a Pruned tree* (BBPruned) [109] technique for big data analytics. The BBPruned allocates Cloud resources for scheduling of MapReduce tasks with a budget and deadline constraints without SLA-violation. Sharma et al. presented Kingfisher [106] that is a cost-efficient framework to offer the effectual provision of elasticity on Cloud to optimize the incurred cost. Kingfisher supports single application provisioning while ignoring the SLA and mixed workload. Lenzini proposed Aliquem (Active list queue method) that is an implementation of *Deficit Round-Robin* (DRR) scheduling to guarantee the significant tradeoff between latency and fairness with low complexity of DRR [36]. Aliquem proves its efficacy against the counterpart Pre-Order DRR and Smoothed DRR implementation scheduling techniques.

Execution-MCT allocates a job to VM producing minimum completion time for it [24, 25, 33, 38, 95]. On each scheduling decision, the current load of a VM is considered to identify an appropriate VM for the allocation of a candidate job [24, 32]. The VM producing minimum completion time for a job is selected and assigned to it. The execution-MCT overloads the faster VMs as compared to slower VMs that results in load imbalance [22, 79]. Execution-Min-Min heuristic operates on the scheduling mechanism of execution-MCT [35, 37, 38] that works in two scheduling steps. In the first step of scheduling decision, the earliest completion time of every job is calculated bearing in mind all VMs in the computing system. Then, the job with the minimum of these completion times is identified and allocated to the concerned VM. Execution-Min-Min favors smaller jobs and penalizes larger jobs [32, 106]. The execution-Min-Min introduces a load imbalance by overloading faster VMs with smaller jobs [24, 50]. PA-LBIMM categorized the jobs and computing machines into VIP and normal classes [37]. The jobs of the former class are allocate to VIP machines using Execution-Min-Min. Afterward, jobs of the latter class are scheduled on all computing machines using Execution-Min-Min. Profit-MCT [45, 110] and Profit-Min-Min [45, 84] work based on execution cost

Example of SLA-2 Job with w1 = 0.3 and w2 = 0.7 values for Job—VM mapping

	VM-1	**VM-2**	**VM-3**
	Execution time	Execution time	Execution time
Job →	22	8	15
Weighted value	22/45 = **0.49**	8/45 = **0.18**	15/45 = **0.34**
	Execution Cost	Execution Cost	Execution Cost
Job →	2	8	6
Weighted value	2/16 = **0.13**	8/16 = **0.5**	6/16 = **0.38**
	Weighted sum	Weighted sum	Weighted sum
Weighted Sum	**0.49***0.3 + **0.13***0.7 = 0.23	**0.18***0.3 + **0.5***0.7 = 0.40	**0.34***0.3 + **0.38***0.7 = 0.36

despite of execution time. The scheduling mechanism of Profit-MCT and Profit-Min-Min are identical to Execution-MCT and Execution-Min-Min, respectively; with a difference of its objective function that is the minimization of execution time in the former heuristics and the minimization of execution cost in the latter heuristics. On each scheduling step, the jobs are assigned to the VM that executes it with the minimum execution cost. Profit-MCT and Profit-Min-Min overload the VMs with the least execution cost and result in a load-imbalance by avoiding to utilize the other VMs. SLA-Min-Min and SLA-MCT are proposed in [45] are based on common SLA to form a balance between overall execution time and cost. SLA-Min-Min and SLA-MCT use three types of SLA-levels; minimization of execution time, cost and both for the submitted workload. SLA-1 jobs prefer to be executed with minimal execution time, SLA-3 prefer to be executed with minimal execution cost and the SLA-2 desires to be executed with a balance between both execution time and cost. Let there is SLA-2 job that has execution time 22, 8, and 15 seconds on VM1, VM2, and VM3, respectively. Similarly, job1 has execution cost 2, 8, and 6 unit cost on VM1, VM2, and VM3, respectively. For SLA-02 jobs, the weight values for both execution time and execution cost will be 0.5 (i.e., w1 = w2). But different weight values may also be given by the user of the SLA jobs. For SLA jobs with weight values for execution time and execution cost, SLA-MCT finds the sum of execution time and cost over all VMs. The execution time and cost is normalized by dividing with corresponding sum values. Then, it finds the weighted sum by multiplying the normalized execution

Table 2.3: Summary of Static SLA-Aware Cloud Scheduling Algorithms.

Algorithms	**Strengths**	**Weaknesses**	**Complexity**
SLA-MCT [45]	Cost-efficient scheduling, single phase scheduling to balance between execution time and cost [45].	Poor resource utilization, load-imbalance [45].	O($M.N$)
SLA-Min-Min [45]	Cost-efficient scheduling, two-phase scheduling to balance between execution time and cost [45].	Poor resource utilization, load-imbalance [45].	O($M.N^2$)
Profit-MCT [45, 110]	Cost-efficient scheduling [45].	Overloads VMs with least execution cost [45], poor resource utilization.	O($M.N$)
Profit-Min-Min [45, 84]	Cost-efficient scheduling [45].	Overloads VMs with least execution cost [45], poor resource utilization.	O($M.N^2$)

time and execution cost value with the corresponding weighted value given by the user for the job. The example of SLA-2 job with w1 = 0.3 and w2 = 0.7 values for Job-VM mapping is presented as above. The VM-1 has smaller weighted sum 0.23 for SLA-2 Job, so the job is scheduled on VM-1. However, SLA-1 jobs are scheduled on VM producing minimum completion time and SLA-3 jobs are scheduled on VM producing minimum cost value. Table 2.3 presents insight summary of the existing SLA-aware algorithms in the literature.

2.4 Job Scheduling Objectives

Cloud computing provides feasibility to gain access to large-scale and high-speed resources without establishing their own computing infrastructure to execute HPC applications. However, from the past several years, the efficient utilization of resources on a compute Cloud has become a prime interest of the scientific community. One of the major causes behind inefficient resource utilization is the imbalance distribution of workload in a distributed computing. It is not only an optimal solution to schedule the independent jobs on machines solely based on the execution time, throughput and average resource utilization ratio; instead, the machine-level load balancing must also be considered to effectuate the usage of the full capacity of computing power in a Cloud system. The existing state-of-the-art contains various Cloud scheduling heuristics that employ load balancing and resource utilization. Alaei and Safi-Esfahani [44] proposed and designed a reactive/proactive scheduling framework that merely relies on prior knowledge pertaining to tasks for a load balanced scheduling. Chen et al. [37] introduced LBIMM and PALBIMM heuristics based on a Min-Min algorithm. LBIMM and PA-LBIMM allocate the jobs to VMs based on the priorities of users and reconsidering the heavily loaded VMs in the scheduling process. This strategy ensures an increased throughput, optimal resource allocation, and decreased average completion time of all jobs. Muhammed et al. [32] proposed Max-Average heuristic that harnesses the Max-Min as a baseline and produces a standard solution with load balanced schedule. The two heuristics are designed by Chauhan [111]: *QoS Guided Weighted Mean Time-Min* (QWMTM) and *QoS Weighted Mean Time Min-Min Max-Min Selective* (QWMTS). The QWMTM and QWMTS consider the QoS weight-index to perform scheduling on the grid based on the criterion of time average. The QWMTM scheduling is based on Min-Min and QWMTS that considers the merits and demerits of the Min-Min and Max-Min for scheduling by reflecting network bandwidth as a QoS parameter. Panda and Jana [45] presented SLA-MCT and SLA-Min-Min heuristics for load balanced scheduling for heterogeneous Cloud computing. The SLA-MCT and SLA-Min-Min are based on MCT and Min-Min techniques, respectively. Both of these techniques consider the execution time and cost as SLA parameters. The outcomes of the study reported a significant balance between makespan and gain costs of the computing services. Mao et al. [18] proposed a load balancing technique based on the Max-Min by maintaining a task status table to predict the real-time load of VMs in the Cloud.

Table 2.4 presents a few scheduling heuristics and the different performance aspects/scheduling objectives employed by the contemporary state-of-the-art. In the literature, Elzeki et al. [22], Mohialdeen [34], Hussain et al. [50], and Li,

Table 2.4: Common Scheduling Objectives in Literature.

Research studies	**Scheduling objectives**
HSGA [64]	Makespan, load balancing rate, failure factor, speedup.
RALBA [50]	Throughput, makespan, ARUR.
RePro-Active [44]	Throughput, makespan, ARUR.
Max-Average [32]	Makespan, ARUR.
QWMTM [38]	Makespan, ARUR, load balancing level.
MCT [38, 50, 52]	Throughput, makespan, ARUR.
Min-Min [37, 50]	Makespan, ARUR, AVIPCT, AORDCT.
Max-Min [18, 50]	Throughput, makespan, ARUR.
Sufferage [31, 50]	Makespan, percent load imbalance ration.
RASA [29, 50]	Throughput, makespan, ARUR.
TASA [39, 50]	Throughput, makespan, ARUR.
SLA-MCT [45]	Makespan, ARUR, gain cost, penalty cost.
SLA-Min-Min [45]	Makespan, ARUR, gain cost, penalty cost.
LBIMM [37]	Makespan, ARUR, AVIPCT, AORDCT.
PA-LBIMM [37]	Makespan, ARUR, AVIPCT, AORDCT.
ECMM [18]	ATRTR, ATPT.

Bo, et al. [38] conducted a comparative analysis of several scheduling heuristics employing different scheduling objectives (in the form of performance parameters expressed in Table 2.4). Further, it is witnessed that most of the researchers [22, 32, 37, 44, 45, 50, 111] employed ARUR as a performance factor to enhance the resource utilization attained by scheduling heuristics [37, 44, 50, 52, 75].

2.5 Job Scheduling - An Empirical Investigation

The empirical investigation of stat-of-the-art Cloud scheduling is conducted [97] using renowned CloudSim simulator with two synthetic and one GoCJ benchmark datasets and presented as follows. The evaluation parameters used in this investigation are makespan, throughput and ARUR.

2.5.1 Simulation Setup

Evaluation of scheduling and resource allocation policies on a real Cloud (with a varying load and system size) is a challenging problem. The use of real testbeds

Table 2.5: Configuration of Simulation Environment

Parameters	**Details**
Simulator/Version	CloudSim version 3.0.2.
Power of Host Machines	04 Dual-core, and 26 Quad-core
Power of each core	4000 MIPS
Total Host Machines	30 Host Machines
Total VMs	50 Virtual Machines
Total Cloudlets	100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000

restricts the experiments to the scale of the test environment. Cloud computing model is based on a pay-per-use model, thus repeatable experiments on real Cloud may incur a high monetary cost. Therefore, an ideal alternative to evaluate resource management related Cloud policies is to use a simulation environment that enables Cloud developers to conduct experiments by employing the desired and varying configurations related to computing infrastructure and dataset (i.e., Cloud jobs). In this work, we use a renowned Cloud simulator called CloudSim [1]. A user job is represented as Cloudlet in CloudSim and the job's size (computational requirement) is measured in terms of *Million Instructions* (MIs) [1]. Similarly, the computing power of a VM is measure in terms of *Million Instructions Per Second* (MIPS) in CloudSim simulator [1]. We perform the simulation-based experiments on a machine equipped with Intel Core i3-4030U Quad-core processor (having 1.9 GHz clock speed) and 04 GB of main memory. Liu and Cho characterizes the computing machines and workloads on a Google cluster and found that 93% of the machines are fairly homogeneous on Google cluster with approximately 6% of the machines are found with a greater computing capability [88]. Using the characteristics of the real computing machines (found in Liu and Cho's study [88]) we build an experimental setup for empirical evaluation. Table 2.5 illustrates the configuration details of the employed simulation environment. Fig. 2.2 presents the overall statistics of the employed VMs with computing powers in terms of *Million Instructions Per Second* (MIPS).

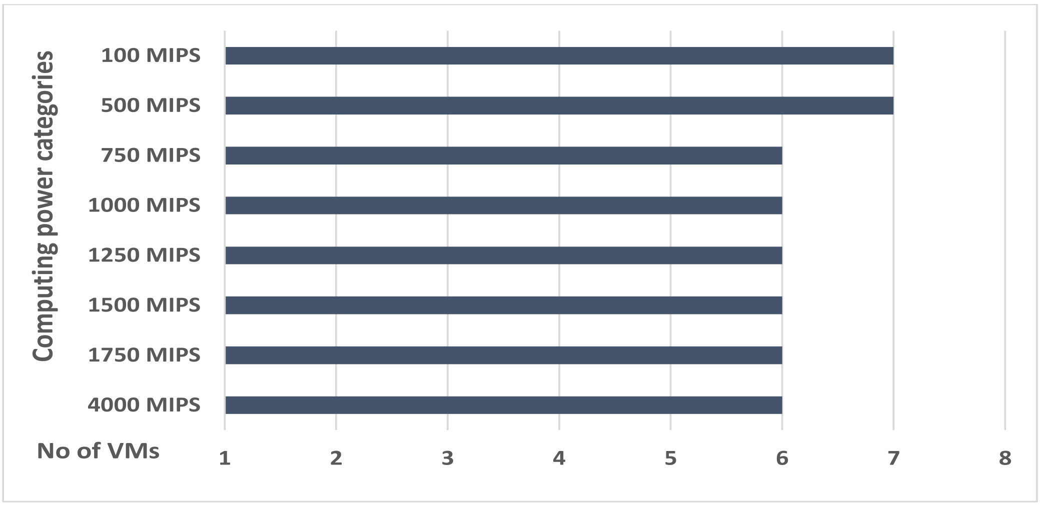


FIGURE 2.2: VMs in Experimental Datacenter.

2.5.2 Employed Datasets

In this empirical work, one GoCJ benchmark dataset [62] and two synthetic datasets are used for the performance assessment of scheduling algorithms in simulation-based experimentation. The detail of proposed benchmark GoCJ dataset is presented in Chapter 3. Fig. 2.3 presents the ratios and sizes of Cloudlets distribution in GoCJ workload in terms of percentage and MIs, respectively. We studied the literature [107, 112, 113] to generate two synthetic datasets containing fix-sized jobs. Mehdi et al. [107] conducted the experimentation of a genetic scheduler with heterogeneous VMs (1.0 GHz to 4.0 GHz speed) to execute up to 100 Cloudlets. In another work, Mehdi et al. [112] have examined the Cloud scheduling using 100 VMs (computing power of 1.0 GHz, 2.0 GHz, 2.5 GHz, and 3.0 GHz) to execute up to 500 Cloudlets. Behzad et al. [113] presented a comparative analysis of different scheduling algorithms by using 7000 and 15000 jobs with a varying number of CPUs (i.e., 04 to 64 processors). In addition, it is evident in the literature [88, 92] that workloads are usually positively skewed (having fewer number of larger jobs and large number of smaller jobs). Following two synthetic workloads are formulated for the empirical investigation existing scheduling algorithms.

The synthetic-I workload is created with five fixed-size Cloudlets (see Fig. 2.4). The majority of Cloudlets (i.e., 75% of the Cloudlets in synthetic-I workload)

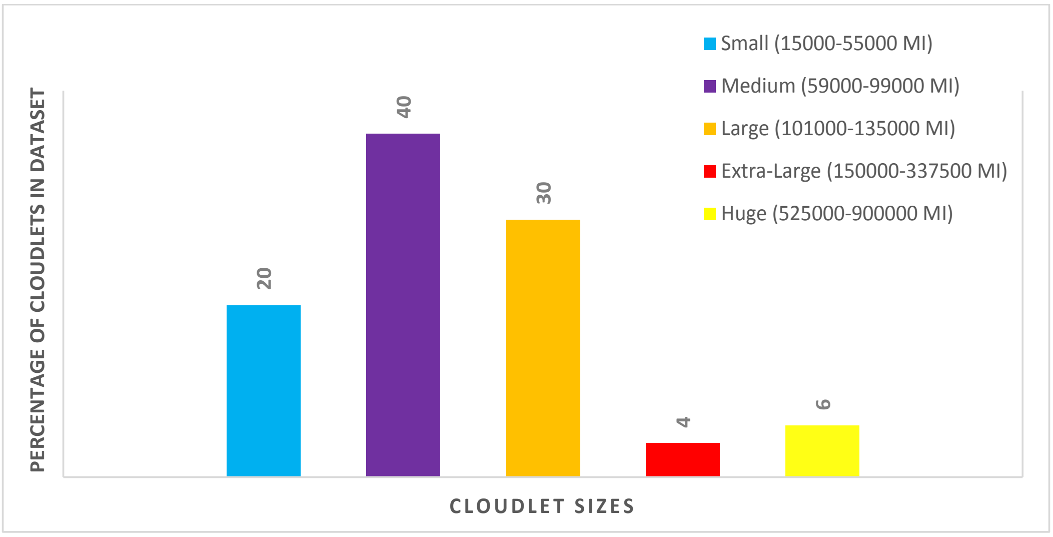


FIGURE 2.3: Composition of GoCJ Dataset.

complete execution within 02 seconds and a small tail of the Cloudlet distribution execute up to 45 seconds (i.e., 15% of the Cloudlets run for 15 seconds and 5% of the Cloudlets run for 45 seconds, respectively). The fixed-sizes of tiny, small, medium, large and extra-large cloudlets in the synthetic-I dataset are 200, 1000, 5000, 15000, and 45000 MIs, respectively. Similarly, the synthetic-II workload is generated using a random number generation mechanism by employing five Cloudlet-size ranges (see Fig. 2.4). The majority of Cloudlets (i.e., 85% in synthetic-II workload) are of short size and a small tail of the Cloudlet distribution completes execution within 45 seconds (i.e., 10% of the Cloudlets run for 10 seconds and 5% of the Cloudlets run for 45 seconds, respectively). The cloudlet-size ranges of tiny, small, medium, large, and extra-large cloudlets are 1–200, 800–1200, 1800–2500, 7000-10000, and 30000–45000 MIs, respectively.

2.5.3 Analysis and Discussion

Three performance metrics are measured for ten existing Cloud scheduling algorithms and presented for experimental evaluation i.e., makespan, ARUR, and throughput.

Makespan-based Investigation

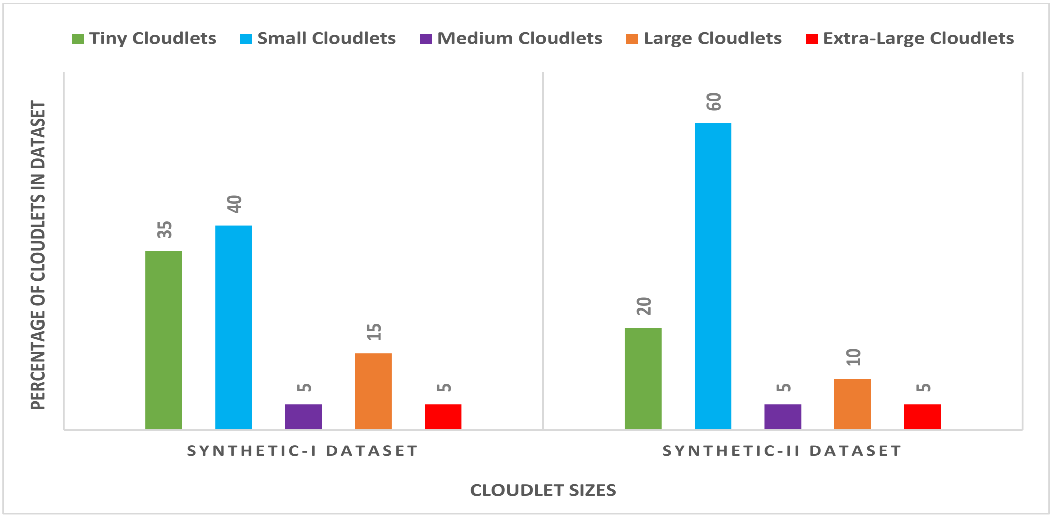


FIGURE 2.4: Composition of Synthetic Workloads.

Term makespan is used to represent the completion of all the Cloudlets execution of a workload. The smaller value of makespan represents a better execution performance. The makespan is mathematically expressed as follows:

$$Makespan = \max_{\forall j=1,2,3,\ldots,M}(VM\_CT_j) \tag{2.1}$$

where, M represents the total number of VMs (which is 50 in our experiments, shown in Table 2.5) and VM_CT_j is the completion time of a specific VM_j by executing its assigned Cloudlets. VM_CT_j is computed as:

$$VM\_CT_j = \sum_{i=1}^{N_j} \frac{Cloudlet_i.MI}{VM_j.MIPS} \tag{2.2}$$

where, $Cloudlet_i.MI$ represents the size of $Cloudlet_i$ in terms of MIs, $VM_j.MIPS$ is the computing power of VM_j in terms of MIPS and N_j represents the total number of Cloudlets assigned to VM_j. Fig. 2.5 shows the makespan results of 10 scheduling heuristics for Synthetic-I, Synthetic-II, and GoCJ workloads. For more clarity, the average makespan (i.e., separately for synthetic-I, synthetic-II, and GoCJ workloads) of all the experiments using a different number of Cloudlets is calculated as follows:

$$Avg\_Makespan = \frac{\sum_{i=1}^{NE} Makespan_i}{NE} \tag{2.3}$$

where, NE represents the number of experiments performed for each scheduling heuristics (i.e., using specified workload) and $Makespan_i$ represents the makespan of the ith experiment. Each experiment is repeated using a varying number of Cloudlets (i.e., Cloudlets 100—10000, as presented in Table 2.5).

LBIMM, TASA, and Sufferage produce the shortest makespan for Synthetic-I, Synthetic-II, and GoCJ workload, respectively. However, there is a minor difference with respect to the makespan of LBIMM, TASA and sufferage heuristics for the three workloads. On the other hand, OLB achieves the largest makespan for Synthetic-I, Synthetic-II, and GoCJ workloads. Fig. 2.6 presents the average makespan attained by the employed scheduling heuristics for the execution of Synthetic-I, Synthetic-II, and GoCJ workloads, respectively. Fig. 2.6 shows that the LBIMM consumes lower execution time (using Synthetic-I workload) as compared to TASA (5.44% lower), Sufferage (8.98% lower), Min-Min (26.35% lower), MCT (32.62% lower), Max-Min (285.69% lower), RASA (302.27% lower), PSSLB (285.73% lower), PSSELB (387.24% lower), and OLB (647.18% lower). Further, the results (using Synthetic-II workload) affirm that TASA achieves lower makespan than the Sufferage (0.88% lower), LBIMM (1.30% lower), Max-Min (252.44% lower), MCT (35.20% lower), Min-Min (38.25% lower), RASA (256.14% lower), PSSLB (251.73% lower), PSSELB (259.37% lower) and OLB (616.34% lower). Similarly, Fig. 2.6 presents that Sufferage consumes lower makespan (using GoCJ workload) as compared to LBIMM (0.38% lower), TASA (1.98% lower), MCT (17.68% lower), RASA (1.74% lower), Min-Min (25.28% lower), Max-Min (11.26% lower), PSSLB (11.26% lower), PSSELB (207.30% lower) and OLB (220.90% lower).

Throughput-based Investigation

Throughput is expressed as the number of jobs executed during the span of per unit time. In our experiments, the number of Cloudlets executed per second is

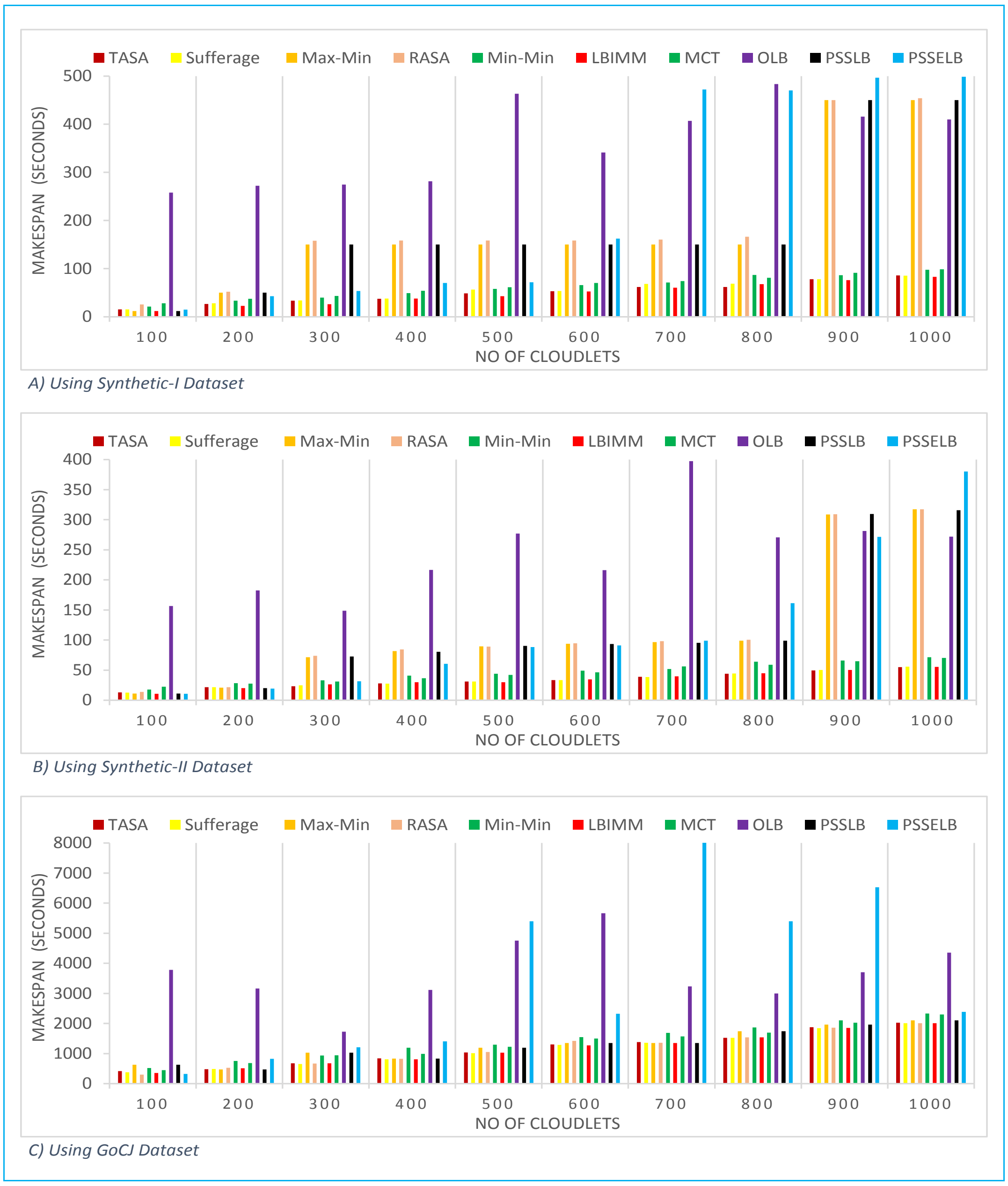


FIGURE 2.5: Makespan Results.

referred to as the throughput of a specific scheduling heuristic. Throughput can be calculated as follows:

$$Throughput = \frac{N}{Makespan} \tag{2.4}$$

where, N is the number of employed Cloudlets. A scheduling technique producing higher throughput value is assumed as a better performing algorithm. For more

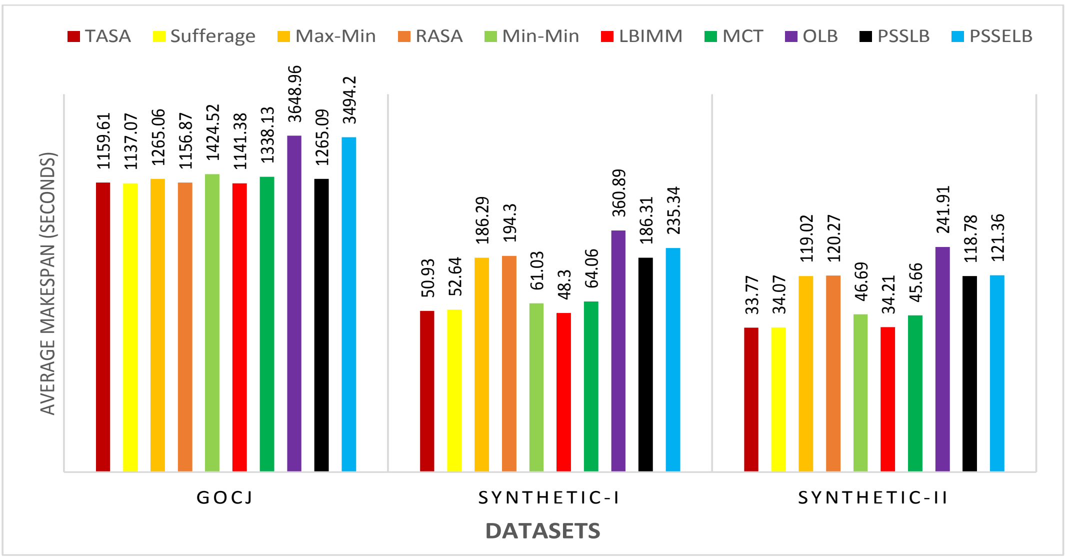


FIGURE 2.6: Average Makespan Results

clarity in results, the average throughput (i.e., for synthetic-I, synthetic-II, and GoCJ workloads) is calculated as:

$$Avg\_Throughput = \frac{\sum_{i=1}^{NE} Throughput_i}{NE} \tag{2.5}$$

Fig. 2.7 presents the throughput results for the execution of Synthetic-I, Synthetic-II, and GoCJ workloads. Likewise, average makespan results, the LBIMM, TASA, and Sufferage heuristics achieves the highest throughput for Synthetic-I, Synthetic-II, and GoCJ workloads, respectively. For more clarity of the simulation results, the average throughput of scheduling heuristics for Synthetic-I workload is shown in Fig. 2.8. LBIMM achieves a higher throughput as compared to the TASA (7.53% higher throughput), Sufferage (11.12% higher throughput), Min-Min (30.24% higher throughput), MCT (37.75% higher throughput), Max-Min (183.41% higher throughput), RASA (236.11% higher throughput), PSSLB (183.41% higher throughput), PSSELB (171% higher throughput) and OLB (560.60% higher throughput). Fig. 2.8 presents the average throughput of ten employed heuristics for Synthetic-II workload. The results affirm that the TASA achieves higher throughput as compared to the Sufferage (1.0% higher), LBIMM (1.14% higher), MCT (34.32% higher), Min-Min (38.95% higher), Max-Min (147.34%

higher), RASA (158.47% higher), PSSLB (145.17% higher), PSSELB (117.67% higher) and OLB (398.51% higher). Similarly, for the makespan results, the Sufferage scheduling heuristic produces a higher throughput while the OLB scheduler produces a lower throughput for the GoCJ workload. Fig. 2.8 presents the average throughput results of the ten scheduling heuristics for the GoCJ workload. The Sufferage scheduling heuristic attains a higher throughput as compared to the TASA (2.22% higher throughput), RASA (1.32% higher throughput), LBIMM (5.25% higher throughput), Max-Min (11.62% higher throughput), MCT (19.74% higher throughput), Min-Min (28.77% higher throughput), PSSLB (11.62% higher throughput), PSSELB (108.60% higher throughput) and OLB (145.21% higher throughput).

ARUR-based Investigation

ARUR shows average resource utilization ratio for a Cloud system. ARUR is calculated as follows [37, 50], which is the ratio of average makespan to the makespan of the Cloud system.

$$ARUR = \frac{\frac{\sum_{j=1}^{M} VM\_CT_j}{M}}{Makespan} \tag{2.6}$$

ARUR value remains between 0 and 1, where value close to 1 shows exceptional resource utilization (i.e., nearest to 100% resource utilization). Fig. 2.9 shows the ARUR based experimental results of the ten scheduling heuristics for Synthetic-I, Synthetic-II, and GoCJ workloads. Mean ARUR value for each heuristic is reported based on the following equation:

$$Mean\_ARUR = \frac{\sum_{i=1}^{NE} ARUR_i}{NE} \tag{2.7}$$

The ARUR results for scheduling heuristics are presented in Fig. 2.10 for Synthetic-I, Synthetic-II, and GoCJ workloads. The LBIMM heuristic attains the highest ARUR (76.5% resource utilization) as compared to other scheduling heuristics for Synthetic-I workload. However, Sufferage produces 75.7% and 86.3% resource utilization (the highest resource utilization as compared to the other heuristics) for Synthetic-II and GoCJ workloads, respectively.

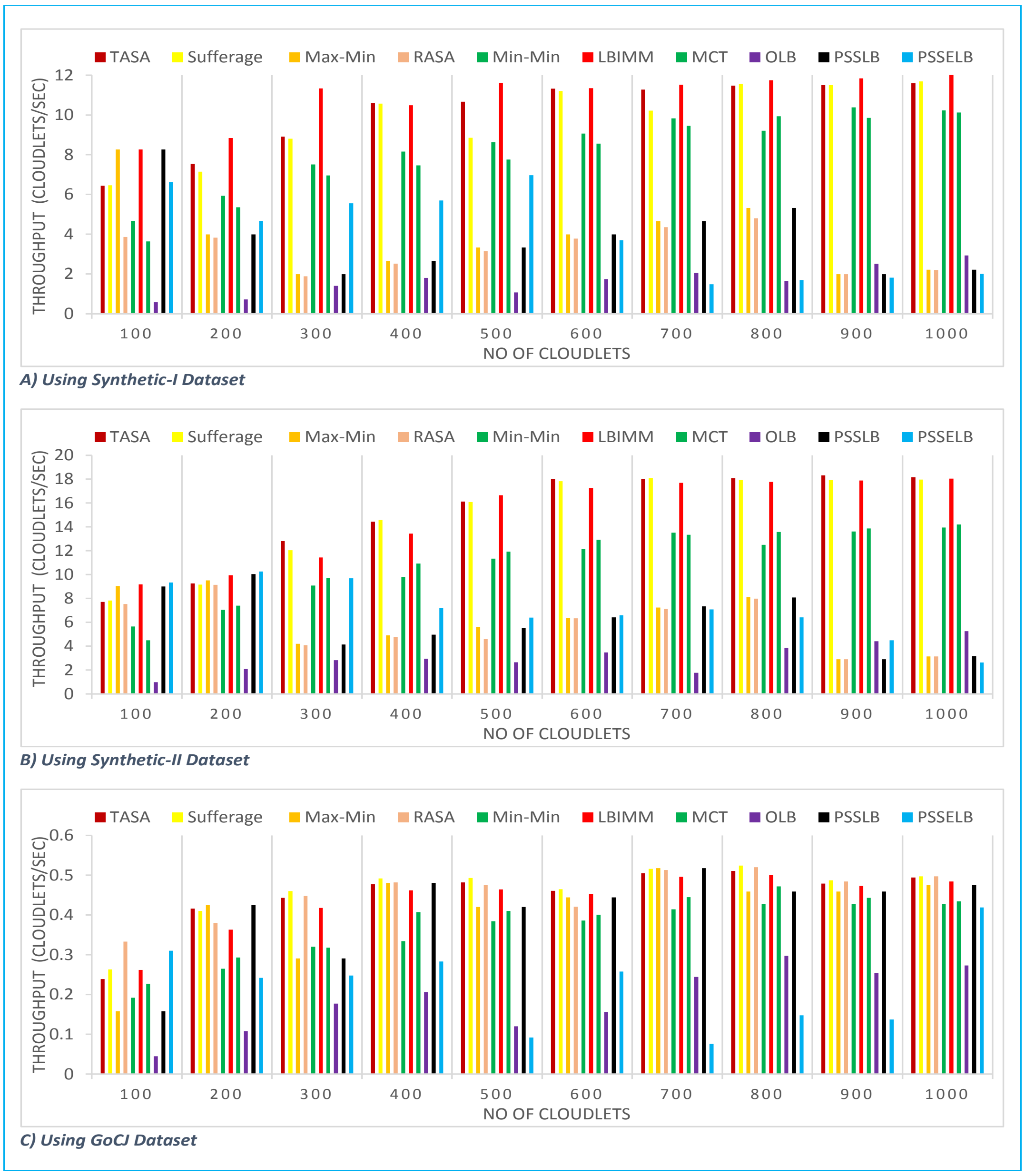


FIGURE 2.7: Throughput Results.

Fig. 2.10 presents the mean ARUR results for execution of Synthetic-I workload.

The results (presented in Fig. 2.10) show that the LBIMM attains the higher resource utilization as compared to the TASA (8.36% higher), Sufferage (11.68% higher), MCT (39.09% higher), Min-Min (64.52% higher), Max-Min (97.67% higher), RASA (124.34% higher), PSSLB (97.67% higher), PSSELB (144.4% higher) and OLB (306.92% higher). For Synthetic-II workload, the Sufferage scheduling heuristic produces higher resource utilization as compared to TASA (0.66% higher),

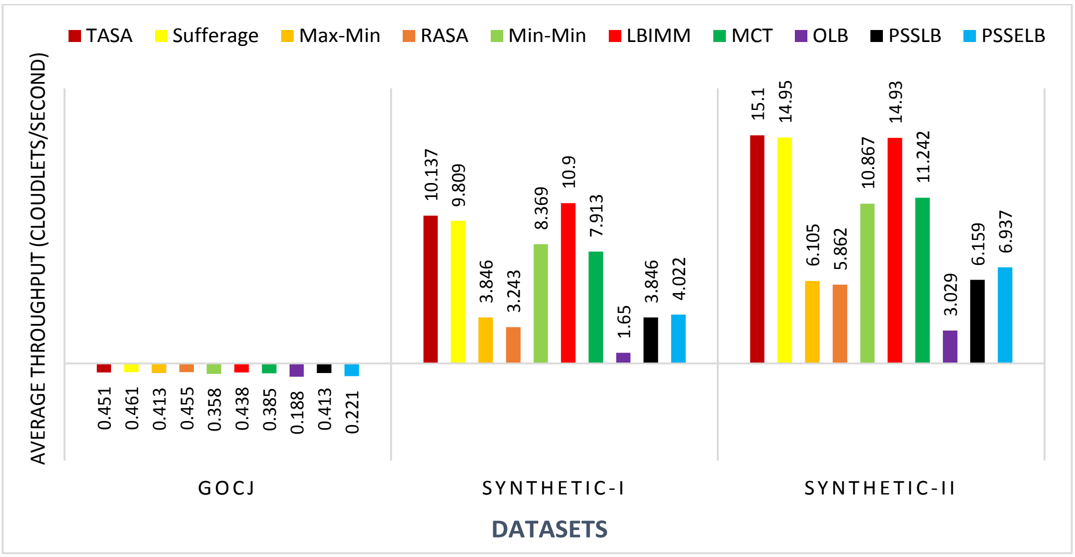


Figure 2.8: Average Throughput Results

LBIMM (9.24% higher), MCT (49.90% higher), Max-Min (94.60% higher), RASA (110.28% higher), Min-Min (101.87% higher), PSSLB (93.60% higher), PSSELB (105.15% higher) and OLB (265.70% higher) heuristics (as shown in Fig. 2.10). The mean ARUR based experimental results for the execution of GoCJ workload are presented in Fig. 2.10. The results affirm that the Sufferage has attained higher resource utilization as compared to LBIMM (2% higher utilization), RASA (2.98% higher utilization), Max-Min (2.13% higher utilization), TASA (8.55% higher utilization), MCT (23.64% higher utilization), Min-Min (65.96% higher utilization), PSSLB (2.13% higher utilization), PSSELB (84.80% higher utilization) and OLB (109.47% higher utilization) heuristics (see Fig. 2.10).

Comparative Discussion on Empirical Investigation

OLB produces poor makespan and very low resource utilization. However, OLB technique requires simple implementation, causes minimal scheduling overhead, and result in lower time complexity. MCT provides improved makespan for the workload execution and minimal completion time for each job. However, MCT results in low resource utilization for both skewed and non-skewed workloads because it overloads the faster machines that result in an imbalanced distribution of workload.

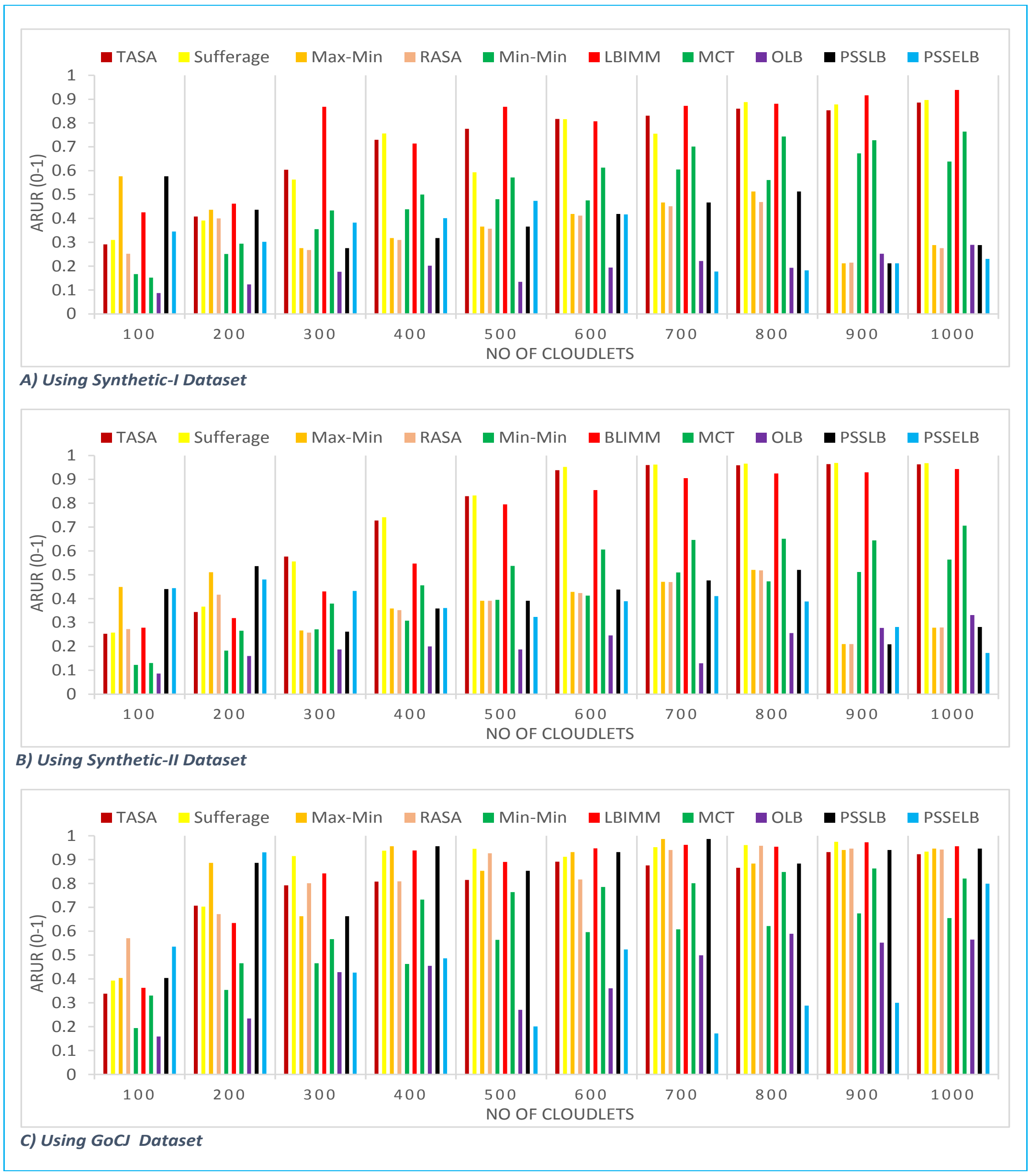


FIGURE 2.9: ARUR Results.

A workload is referred to as positively skewed if it contains a large number of shorter size jobs with a few very long jobs [28]. On the other hand, if the workload comprises a large number of longer jobs with a few shorter jobs then the workload is referred to as negatively skewed [28]. The skewness in synthetic and GoCJ benchmark workloads used in this study is shown in Fig. 2.3 and Fig. 2.4.

Min-Min and Max-Min do not produce good results (in terms of execution time) for a skewed workload [28]. Min-Min scheduling attains improved execution time

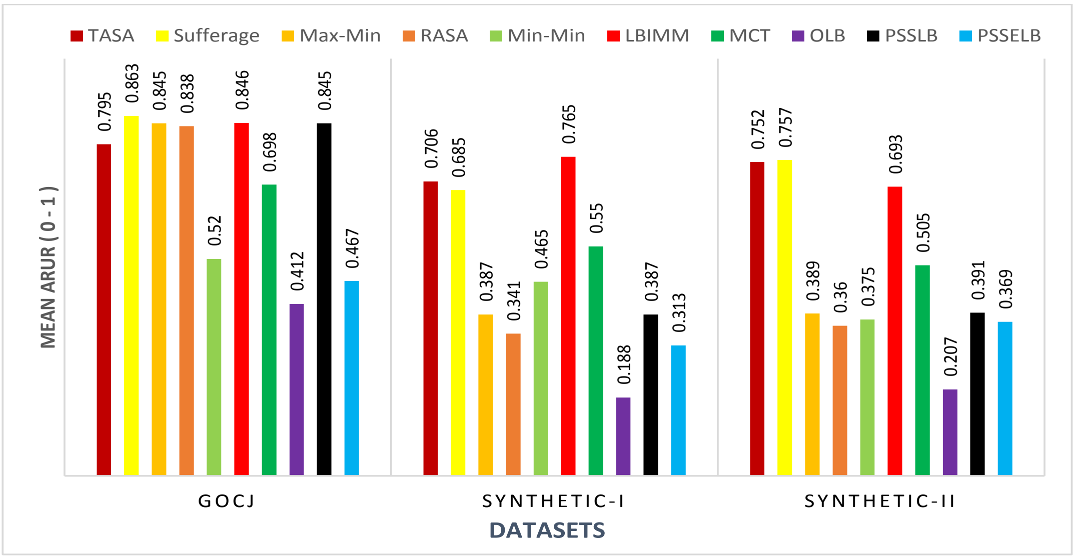


FIGURE 2.10: Mean ARUR Results

when the workload has shorter size jobs or cloudlets. On the other hand, Min-Min produces longer makespan for a positively skewed workload (due to the inherent penalty for larger jobs). In case of workload containing most of the shorter jobs with few longer jobs, Max-Min achieves improved makespan by executing longer jobs on faster machines; and concurrently executing the shorter jobs on comparatively slower machines. However, Max-Min and Min-Min performs worst for a skewed workload and provides improved results for a non-skewed workload [28].

RASA scheduling mechanism benefits from the merits of Min-Min and Max-Min producing a lower response time for both smaller and larger size jobs [29]. TASA, Sufferage, and LBIMM are modified versions of Min-Min [29, 39] Therefore, these scheduling techniques provide better results (for a non-skewed workload) as compared to Min-Min. Additionally; TASA produces higher resource utilization (as evident by the results are shown in Section 2.5.3). Similarly, PSSLB and PSSELB algorithms are the modified version of Max-Min techniques (as shown in Fig. 2.1). PSSELB provides better resource utilization as compared to Max-Min, while it introduces a slight degradation in makespan and throughput of PSSELB as compared to Max-Min. On the other hand, PSSLB shows resemblance in results (i.e., makespan, resource utilization, and throughput results) as compared to Max-Min technique.

2.5.4 Resource Utilization and Load-Imbalance

The literature [26, 37, 44, 64, 68, 69, 77, 78] is scrutinized to examine the scheduling aspects related to load balance and resource utilization for Cloud computing platform. The detailed analysis of the literature revealed that most of the existing scheduling heuristics [22, 24, 26, 29, 31, 37, 39, 44, 45, 64, 68, 69, 73–75, 75–78] are more inclined towards the decrease in turnaround and response time of a Cloud workload. For example, MCT, Min-Min, Max-Min, RASA, TASA, and Sufferage heuristics are designed and proposed to minimize the makespan of the Cloud workload. The OLB, LBIMM, PSSLB, and PSSELB scheduling heuristics consider additional consideration of load balance too. However, most of these scheduling heuristics are unable to fully utilize the computing resources and imbalanced work distribution among the virtual machines.

Our empirical analysis reveals that the LBIMM, TASA, and the Sufferage heuristics produce comparatively better utilize the computing resources (i.e., higher resource utilization) as compared to the other scheduling heuristics presented in this work. In Table 2.6, the results show that the LBIMM heuristic attains higher resource utilization (i.e., 8.36% and 11.52% higher, respectively) as compared to the TASA, and Sufferage for the execution of Synthetic-I workload. The Sufferage heuristic attained higher resource utilization (i.e., 0.66% and 9.24% higher, respectively) as compared to the TASA, and LBIMM for the execution of Synthetic-II workload. Similarly, the Sufferage scheduling achieves higher resource utilization (i.e., 2.01% and 8.55% higher resource utilization, respectively) as compared to the LBIMM, and TASA for the GoCJ workload. Moreover, the Sufferage, Max-Min, and PSSLB heuristics attain higher resource utilization too for the execution of a GoCJ workload. The Max-Min and RASA have achieved higher resource utilization of 5.40% and 6.29% as compared to TASA (for GoCJ workload), respectively. This minor improved resource utilization by the RASA and the Max-Min over TASA is due to the lower resource utilization incurred by the Min-Min heuristic (as compared to the Max-Min and RASA for the execution of GoCJ workload).

TABLE 2.6: Percentage of Resource Utilization of Scheduling Algorithms.

Algorithms	GoCJ Dataset	Synthetic-I Dataset	Synthetic-II Dataset
TASA	79.5%	70.6%	75.2%
Sufferage	86.3%	68.5%	75.7%
Max-Min	84.5%	38.7%	38.9%
RASA	83.8%	34.1%	36.0%
Min-Min	52.0%	46.5%	37.5%
LBIMM	84.6%	76.5%	69.3%
MCT	69.8%	55.0%	50.5%
PSSLB	84.5%	38.7%	39.1%
PSSELB	46.7%	31.3%	36.9%
OLB	41.2%	18.8%	20.7%

It is empirically observed that the LBIMM, TASA and Sufferage schedulers achieve the minimal completion time and better resource utilization as compared to Max-Min, RASA, Min-Min, MCT, OLB, PSSLB, and PSSELB. The experimental results reveal that the LBIMM, TASA and Sufferage mechanisms attain on average lower makespan compared to the other state-of-the-art. However, most of these mechanisms still lack a higher resource utilization (see Table 2.6) that could be improved to further lessen the makespan for the execution of a Cloud workload.

Improving resource utilization is very crucial to the reduces the cost and energy consumption for the execution of Cloud workload [23], [40]. Therefore, the issue concerning low resource utilization should be addressed in a comprehensive manner. For the optimal resource utilization, the workload should be assigned to the computing resources according to the computing capabilities and application need. A resource- and application-aware Cloud scheduling algorithm with load balancing will greatly benefit in terms of reduced execution time and cost. Therefore, we have investigated machine-level load balancing (i.e., VM category and VM-level load distribution) by using these ten scheduling algorithms.

VM Category and VM-wise Load Imbalance

For the empirical investigation, we employ the simulation environment based on 50 VMs (08 different sizes as shown in Fig. 2.2). For a balanced workload distribution, the Cloudlets (i.e., Cloud jobs) must be submitted for execution by considering the computing capabilities of the employed VMs. Moreover, it is critical to consider the current load of a VM too. The computing load or share of each VMj is presented as Sharej and can be calculated as:

$$Share_j = \sum_{i=1}^{N} Cloudlet_i.MI \times \frac{VM_j.MIPS}{\sum_{k=1}^{m} VM_k.MIPS} \tag{2.8}$$

where, $Share_j$ is the amount of workload in terms of MI that needs to be allocated to VM_j to attain a load-balanced scheduling. The balance share for each VM category in terms of percentage workload is represented as $VMCat\_Share_c$ and is calculated as follows:

$$VMCat\_Share_c = \frac{\sum_{a=1}^{cm} Share_a}{\sum_{i=1}^{N} Cloudlet_i.MI} \times 100 \tag{2.9}$$

where, c represents the VM category and cm are the number of VMs in the VM category c.

Percentage workload distribution by the 10 scheduling heuristics is presented in a tabular form to highlight the load imbalance (see Fig. 2.11). VMCat_Share_c of VM categories (see Fig. 2.2) in a simulation environment is presented as a reference for load distribution attained by the employed scheduling heuristics (see Fig. 2.11, presented in the last row). The imbalanced workload assignments are depicted such as the underutilized resources are filled with orange-color background, heavily loaded resources with red-background, and idle resources with a green background.

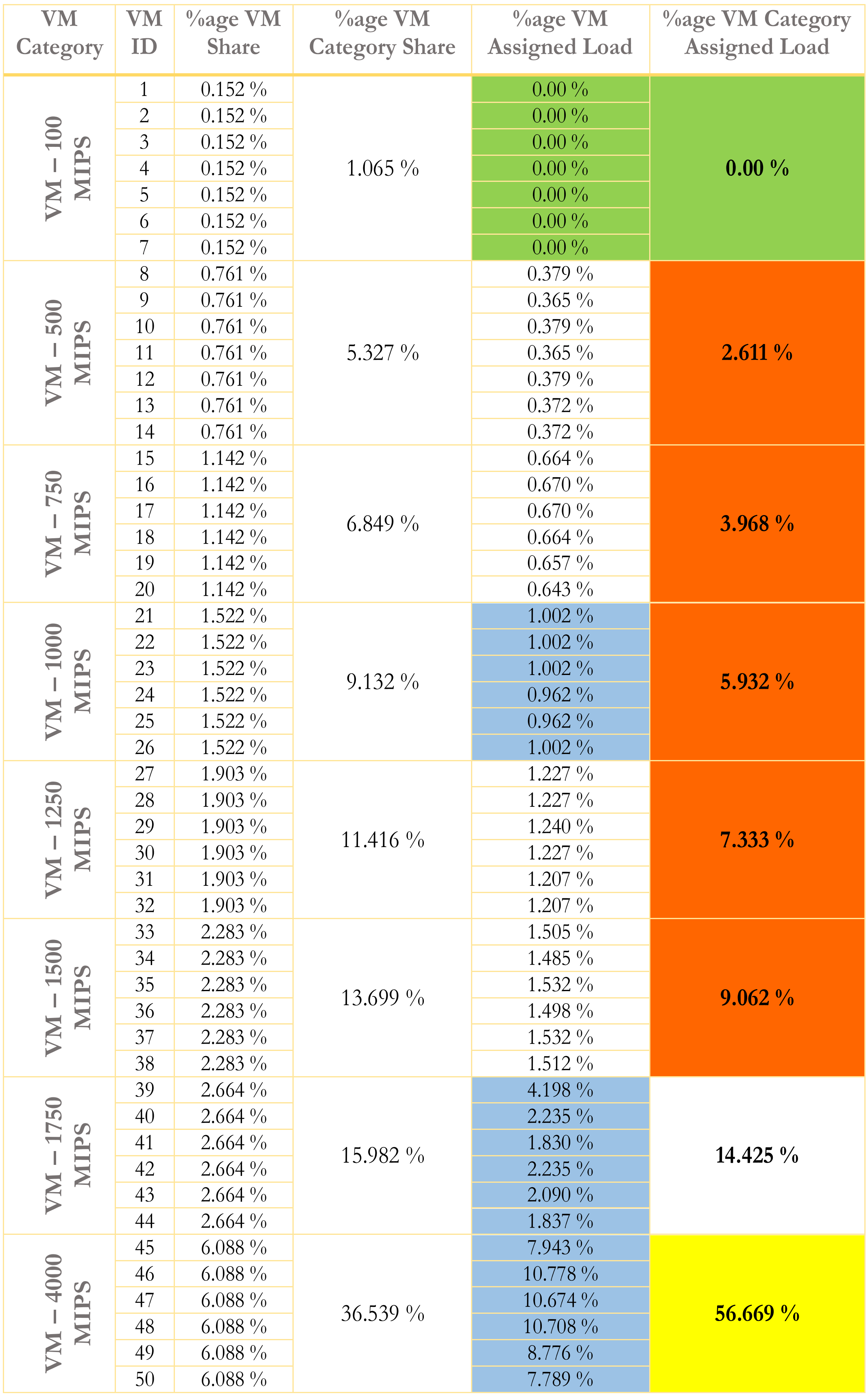
All the VMs based on 100 MIPS (14% of the computing nodes of the experimental setup) and a few of the VM with 500 and 750 MIPS remain idle when the GoCJ workload is scheduled using the Min-Min heuristic. Additionally, the Min-Min overloads the fastest VMs (based on 4000 MIPS). On the other hand, the Max-Min scheduling heuristic produces better workload distribution as compared to the Min-Min; however, only a few VMs (both the slow and faster) are overloaded. This load mapping scenario is mainly contributed by the composition of GoCJ workload; where a small portion of large-sized Cloudlets or Cloud jobs is present along with a majority of small-sized Cloudlets. The Max-Min heuristic overcomes the imbalance produced by the Min-Min due to the presence of a few large-sized

Cloudlets which suits the Max-Min scheduling. The PSSLB heuristic shows almost the same behavior for the workload distribution like Max-Min because both of these heuristics favor larger jobs. The LBIMM, Sufferage, RASA, Max-Min, and TASA produce comparatively a load-balanced schedule. Among these heuristics, Sufferage produces the highest resource utilization because of the resource-aware mechanism. However, an interesting observation is that the VMs based on 100 MIPS remain idle. Moreover, the recourse-ware mechanism employed by the Suffrage also produces a notable imbalance where the number of scheduled Cloudlets is lesser (i.e., 100 Cloudlets) as shown in Fig. 2.11. Similarly, the LBIMM heuristic shows an improved load balancing in workload distribution. Also, the RASA scheduling heuristic produces better load balancing due to the inherent usage of Min-Min and Max-Min (in alternate scheduling steps). However, the slowest machines remain idle (due to the inherent use of Min-Min heuristic) and the fastest VMs remain overloaded when a small number of Cloudlets are scheduled by the RASA (see Fig. 2.11). The employed alternate Min-Min and Max-Min mechanisms (by the RASA) produce fair scheduling (for both the large and small size Cloudlets). Similarly, the TASA technique produces minimal makespan among the ten employed scheduling heuristics. However, most of the slower VMs (i.e., 100 and 500 MIPS based) become idle due to the use of Min-Min in alternate scheduling steps. On the other hand, the TASA overcomes the load imbalance (caused by the Min-Min heuristic) to some extent with the help of inherent Sufferage based mechanism (in alternate scheduling steps). The Sufferage scheduling heuristic produces a better load-balanced schedule; however, very few slow VMs (with 100 MIPS) remain idle. The results reveal that there is sufficient possibility of imbalance workload distribution (among VMs) even a scheduling heuristic attains an improved ARUR value; (as presented in Section 2.5.3, ARUR-based Investigation). It is empirically evident that most of the existing scheduling mechanisms produce a reduced makespan with higher throughput. However, often these heuristics result in a load imbalanced scheduling.

For example, Sufferage produces a higher ARUR value of 0.863 (i.e., 86.3% resource utilization) for the GoCJ workload (see Fig. 2.10). However, the scheduling by

Scheduling Heuristics	No. of Cloudlets	VMs 100 MIPS	VMs 500 MIPS	VMs 750 MIPS	VMs 1000 MIPS	VMs 1250 MIPS	VMs 1500 MIPS	VMs 1750 MIPS	VMs 4000 MIPS
OLB	100	6.58 %	7.15 %	6.61 %	9.90 %	12.09 %	14.45 %	15.42 %	27.80 %
	300	3.00 %	7.44 %	7.03 %	14.47 %	10.76 %	12.29 %	15.12 %	28.89 %
	500	1.96 %	6.61 %	7.70 %	9.83 %	11.16 %	13.23 %	15.90 %	33.62 %
	800	1.68 %	5.91 %	6.81 %	9.70 %	11.35 %	14.00 %	15.84 %	34.70 %
	1000	2.02 %	5.92 %	7.25 %	9.32 %	11.03 %	13.75 %	16.38 %	34.33 %
MCT	100	0.00 %	3.07 %	4.54 %	6.92 %	8.83 %	10.66 %	13.81 %	52.16 %
	300	0.47 %	4.34 %	5.98 %	8.06 %	10.18 %	12.86 %	14.87 %	43.25 %
	500	0.65 %	4.72 %	6.29 %	8.67 %	11.00 %	13.37 %	15.92 %	39.38 %
	800	0.85 %	5.00 %	6.57 %	8.80 %	11.16 %	13.53 %	15.96 %	38.15 %
	1000	0.88 %	5.08 %	6.61 %	8.86 %	11.11 %	13.48 %	15.74 %	38.25 %
Min-Min	100	0.00 %	0.00 %	0.00 %	4.34 %	3.64 %	7.68 %	11.93 %	72.40 %
	300	0.00 %	1.57 %	3.06 %	5.25 %	6.36 %	18.10 %	20.20 %	45.45 %
	500	0.00 %	3.28 %	4.33 %	4.99 %	11.49 %	12.56 %	17.74 %	45.62 %
	800	0.00 %	3.24 %	4.38 %	7.74 %	11.79 %	12.18 %	18.63 %	42.04 %
	1000	0.00 %	2.90 %	4.16 %	10.09 %	10.79 %	14.24 %	15.27 %	42.54 %
LBIMM	100	0.00 %	1.95 %	3.43 %	4.34 %	7.95 %	10.69 %	11.93 %	60.07 %
	300	0.54 %	4.90 %	6.53 %	9.03 %	11.22 %	13.71 %	16.47 %	37.52 %
	500	0.62 %	4.90 %	6.53 %	9.03 %	11.22 %	13.71 %	16.47 %	37.52 %
	800	0.93 %	5.09 %	6.64 %	8.97 %	11.55 %	13.67 %	16.03 %	37.12 %
	1000	0.88 %	5.21 %	6.80 %	9.05 %	11.45 %	13.66 %	16.01 %	36.94 %
Max-Min	100	3.09 %	4.66 %	6.93 %	8.42 %	11.28 %	12.57 %	17.81 %	35.25 %
	300	1.73 %	5.17 %	6.94 %	9.13 %	11.42 %	13.55 %	15.78 %	36.29 %
	500	1.28 %	5.31 %	6.84 %	9.16 %	11.31 %	13.69 %	15.94 %	36.48 %
	800	1.22 %	5.38 %	6.92 %	9.10 %	11.40 %	13.62 %	15.92 %	36.43 %
	1000	1.13 %	5.33 %	6.79 %	9.10 %	11.40 %	13.70 %	15.99 %	36.55 %
RASA	100	0.00 %	4.52 %	6.02 %	9.10 %	10.96 %	12.75 %	18.26 %	38.38 %
	300	0.00 %	4.78 %	6.81 %	8.93 %	11.43 %	13.56 %	16.16 %	38.31 %
	500	0.99 %	5.28 %	6.72 %	8.90 %	11.36 %	13.55 %	15.93 %	37.27 %
	800	0.96 %	5.15 %	6.72 %	8.99 %	11.24 %	13.77 %	16.08 %	37.10 %
	1000	0.80 %	5.17 %	6.75 %	9.06 %	11.41 %	13.75 %	16.05 %	37.03 %
TASA	100	0.00 %	1.61 %	3.80 %	6.99 %	10.30 %	11.02 %	14.67 %	51.61 %
	300	0.00 %	4.90 %	6.34 %	9.03 %	11.23 %	13.91 %	16.49 %	38.10 %
	500	0.00 %	5.01 %	6.72 %	8.97 %	11.38 %	13.74 %	16.37 %	37.81 %
	800	0.36 %	4.93 %	6.64 %	9.11 %	11.55 %	13.71 %	16.08 %	37.61 %
	1000	0.71 %	5.11 %	6.60 %	9.03 %	11.45 %	13.67 %	16.13 %	37.30 %
Sufferage	100	0.00 %	2.71 %	4.40 %	7.57 %	9.70 %	12.22 %	14.13 %	49.28 %
	300	0.70 %	4.87 %	6.86 %	9.08 %	11.42 %	13.79 %	16.18 %	37.11 %
	500	0.88 %	5.10 %	6.63 %	9.09 %	11.49 %	13.81 %	16.05 %	36.96 %
	800	0.89 %	5.19 %	6.78 %	9.08 %	11.42 %	13.71 %	16.04 %	36.89 %
	1000	0.72 %	5.21 %	6.73 %	9.05 %	11.44 %	13.79 %	16.04 %	37.01 %
PSSELB	100	0.22 %	4.42 %	6.82 %	7.82 %	11.01 %	12.42 %	14.38 %	42.90 %
	300	0.00 %	3.75 %	5.47 %	9.43 %	9.84 %	17.11 %	15.80 %	38.60 %
	500	1.60 %	6.49 %	6.69 %	10.56 %	10.55 %	13.74 %	15.83 %	34.55 %
	800	1.37 %	5.31 %	6.43 %	9.04 %	11.40 %	13.89 %	16.19 %	36.37 %
	1000	0.72 %	5.35 %	6.90 %	9.75 %	12.00 %	13.76 %	15.39 %	36.12 %
PSSLB	100	3.09 %	4.66 %	6.93 %	8.42 %	11.28 %	12.57 %	17.81 %	35.25 %
	300	1.73 %	5.17 %	6.94 %	9.13 %	11.42 %	13.55 %	15.78 %	36.29 %
	500	1.28 %	5.31 %	6.84 %	9.16 %	11.31 %	13.69 %	15.94 %	36.48 %
	800	1.22 %	5.38 %	6.92 %	9.10 %	11.40 %	13.62 %	15.92 %	36.43 %
	1000	1.13 %	5.33 %	6.79 %	9.10 %	11.40 %	13.70 %	15.99 %	36.55 %
Balanced workload for VM categories.									
%age Load of VMs		**1.07 %**	**5.33 %**	**6.85 %**	**9.13 %**	**11.42 %**	**13.70 %**	**15.98 %**	**36.53 %**

FIGURE 2.11: VM Category-wise %age Workload Distribution for GoCJ Dataset.



VM Category	VM ID	%age VM Share	%age VM Category Share	%age VM Assigned Load	%age VM Category Assigned Load
VM – 100 MIPS	1	0.152 %	1.065 %	0.00 %	**0.00 %**
	2	0.152 %		0.00 %	
	3	0.152 %		0.00 %	
	4	0.152 %		0.00 %	
	5	0.152 %		0.00 %	
	6	0.152 %		0.00 %	
	7	0.152 %		0.00 %	
VM – 500 MIPS	8	0.761 %	5.327 %	0.379 %	**2.611 %**
	9	0.761 %		0.365 %	
	10	0.761 %		0.379 %	
	11	0.761 %		0.365 %	
	12	0.761 %		0.379 %	
	13	0.761 %		0.372 %	
	14	0.761 %		0.372 %	
VM – 750 MIPS	15	1.142 %	6.849 %	0.664 %	**3.968 %**
	16	1.142 %		0.670 %	
	17	1.142 %		0.670 %	
	18	1.142 %		0.664 %	
	19	1.142 %		0.657 %	
	20	1.142 %		0.643 %	
VM – 1000 MIPS	21	1.522 %	9.132 %	1.002 %	**5.932 %**
	22	1.522 %		1.002 %	
	23	1.522 %		1.002 %	
	24	1.522 %		0.962 %	
	25	1.522 %		0.962 %	
	26	1.522 %		1.002 %	
VM – 1250 MIPS	27	1.903 %	11.416 %	1.227 %	**7.333 %**
	28	1.903 %		1.227 %	
	29	1.903 %		1.240 %	
	30	1.903 %		1.227 %	
	31	1.903 %		1.207 %	
	32	1.903 %		1.207 %	
VM – 1500 MIPS	33	2.283 %	13.699 %	1.505 %	**9.062 %**
	34	2.283 %		1.485 %	
	35	2.283 %		1.532 %	
	36	2.283 %		1.498 %	
	37	2.283 %		1.532 %	
	38	2.283 %		1.512 %	
VM – 1750 MIPS	39	2.664 %	15.982 %	4.198 %	**14.425 %**
	40	2.664 %		2.235 %	
	41	2.664 %		1.830 %	
	42	2.664 %		2.235 %	
	43	2.664 %		2.090 %	
	44	2.664 %		1.837 %	
VM – 4000 MIPS	45	6.088 %	36.539 %	7.943 %	**56.669 %**
	46	6.088 %		10.778 %	
	47	6.088 %		10.674 %	
	48	6.088 %		10.708 %	
	49	6.088 %		8.776 %	
	50	6.088 %		7.789 %	

FIGURE 2.12: VM-wise %age Workload Distribution for GoCJ Dataset by Min-Min Algorithm using 250 Cloudlets.

Sufferage heuristic in this scenario does not utilize the VMs (with computer power of 100 MIPS) i.e., those VMs remained idle. In addition to that, VMs with the computing capability of 500 and 750 MIPS are underutilized too and the VMs with 4000 MIPS were heavily loaded (for the schedule of 100 Cloudlets-based job pool (see Fig. 2.11)). On the other hand, the VMs with the computing power of 100 MIPS were being utilized by the Sufferage scheduling when the number of Cloudlets in the job pool increased. Similarly, LBIMM heuristic attains 0.846 ARUR (i.e., 84.6% resource utilization); however, VMs with 100 MIPS remain idle. Moreover, VMs with 4000 MIPS were observed heavily overloaded and all the other VMs (in the employed experimental setup) are observed as underutilized (for 100 Cloudlets-based scheduling). TASA technique produces 0.795 ARUR (i.e., 79.5% resource utilization) for the GoCJ workload (see Fig. 2.10); however, VMs with 100 MIPS remain idle (see Fig. 2.11). TASA utilizes the VMs with 100 MIPS; however, most of these VMs (100 MIPS based) remained underutilized when the number of Cloudlets, to be scheduled, are increased in the experiments. Min-Min scheduling technique produces 0.52 ARUR (i.e., 52% resource utilization) for the GoCJ workload (see Fig. 2.10). Fig. 2.11 depicts a load imbalance profile of the workload distribution by the Min-Min scheduling heuristic. VMs with 100 MIPS remain idle due to the imbalanced scheduling of the Min-Min. The VMs with 500 and 750 MIPS were assigned with Cloud jobs (i.e., Cloudlets); however, these VMs were significantly underutilized (as shown in Fig. 2.11), when the number of Cloudlets increases. Similarly, VMs with 1000, 1250, 1500, and 1750 MIPS also remain underutilized for the Min-Min based scheduling. Contrarily, VMs with 4000 MIPS are heavily overloaded by Min-Min technique (see Fig. 2.11).

This empirical investigation reveals that for the workload distribution the scheduling heuristics producing better ARUR value still result in load-imbalance on the machine level. The imbalanced distribution of workload among the VMs within the same VM category is observed too. Fig. 2.12 presents the workload assignment among all VMs by the Min-Min scheduling heuristics (using 250 Cloudlets) for the GoCJ workload. Fig. 2.12 highlights the imbalanced distribution of workload. The underutilized VM categories are highlighted in orange color. The heavily overloaded or idle VM categories are highlighted with the yellow and green background, respectively. Similarly, the load imbalance of VMs within a specific VM category is shown with a light-blue color. Despite balancing the workload assigned to VM category with 1750 MIPS, it can be seen that the VM with ID 39 is heavily overloaded (i.e.; 4.198% workload is assigned) and VMs with ID 41 and ID 45 are underutilized with only 1.830 and 1.837% workload assignment, respectively (see Fig. 2.12).

Higher resource utilization can be attained if all the VMs in Cloud exhibit approximately same makespan. The load balance execution guarantees that all the computing resources (i.e., VMs) are being fully utilized and there are no idle resources. Ultimately, the minimal makespan with maximal throughput will be ensured. Load balanced execution in Cloud is a highly desirable aspect that will ensure lower makespan in amalgamation with higher throughput, higher resource utilization, and less energy cost. In summary, this empirical investigation highlights the following issues and potential research directions.

1. A balanced distribution of workload among computing resources be accomplished to achieve improved resource utilization with reduced makespan, and increased throughput in Cloud computing;

2. Designing and implementing resource-aware holistic scheduling that not only considers the application's computing requirements but also contemplates virtual machine level attributes to provide a higher ARUR and near-optimal load balancing for the Cloud workload execution.

Chapter 3

Solution Methodology

This chapter presents the proposed works in the thesis. A new benchmark dataset in the form of a GoCJ dataset for distributed and Cloud computing, and a novel resource-aware scheduling mechanism (i.e., RALBA scheduler) that is capable of mapping users jobs in Cloud environment in a balanced manner is proposed and presented. In addition, a second novel cost-efficient and resource-aware scheduling mechanism (i.e., SLA-RALBA scheduler) that provides jobs to virtual machines mapping with an improved balance among execution time, cost and resource utilization is also presented.

3.1 GoCJ Dataset

Datasets are becoming increasingly more pertinent when executing the performance assessment of Cloud-scheduling, resource-allocation, and load-balancing algorithms used for eagle-eyed examination of efficiency and performance in a real-world Cloud. A minor change in the behavior and nature of the dataset is reflected in the performance of scheduling and resource-allocation policies. Assessing the scheduling and allocation policies on Cloud infrastructures under a varying load and system size is a challenging problem. Real Cloud workload is hard to acquire for performance analysis and investigation due to the users' data confidentiality and policies maintained by CSPs. In addition, using real testbeds limits the experiments to the scale of the testbed. Hence, testing the accuracy performance with real-world datasets is crucial in the field of research, and synthetic data does not ally represent an actual dataset [63]. The most appropriate alternative is to make the investigation in a simulation environment with a load of varying behavior in the Cloud environment. For Cloud computing research, it is valuable to formulate and ensure a widespread availability of realistic datasets that show how resourcefully the Cloud addresses the user requirements.

The contemporary state-of-the-art has been scrutinized to explore a real workload behavior in Google cluster traces [88–91, 93] and MapReduce logs from the M45 supercomputing cluster [92]. Liu and Cho analyzed large-scale Google cluster traces of 29-days to examine the machine properties, workload behavior and resource usage [88]. The analysis affirms that the majority of the jobs execute for fairly a short duration of fewer than 15 min, while a few jobs execute over 300 min. The median length of a job in Google cluster traces is witnessed as approximately 3 min. Furthermore, it establishes the fact that approximately two-thirds of the jobs execute for less than 5 min and approximately 20% of the jobs execute for less than one minute. It is found that most jobs in Google cluster traces are shorter length. The shorter jobs are generally used for test runs on Google cluster [88]. Likewise, the MapReduce logs of the M45 supercomputing cluster presented in [114] is also scrutinized. The MapReduce logs for the duration of 10 months (i.e., Hadoop logs from April 25, 2008, to April 24, 2009, except logs from Nov 12, 2008, to Jan 18, 2009) have been released by the Yahoo. It is shown that most of jobs (i.e., 95% of the jobs) complete the execution within 20 min, and approximately 4% of jobs take up to 30 min [114] to execute.

The main contribution of GoCJ dataset is to introduce a realistic Google Cloud Jobs dataset based on Google Cloud infrastructure as a benchmark for Cloud-scheduling researchers. This data descriptor is presented to address the research issue of ensuring the public availability of a realistic dataset that satisfies the need for researchers in performance analysis of Cloud infrastructure. A sample dataset (with a small number of jobs) based on jobs trend behavior in Google cluster traces

Table 3.1: Description of GoCJ Dataset

Feature	**Description**
Subject area	Computer Science, Cloud Computing
Type of data	Text files, Excel files
Acquisition	Analysis of Google cluster traces [88–91, 93].
Generation	GoCJ is created using Monte Carlo simulation
Availability	Online available on Mendeley [94]
Articles	GoCJ is used in research [50, 52, 96, 97, 116–120]

is formulated and provided to the GoCJ dataset generator, which simulates the desired dataset comprised of any required number of jobs. The proposed GoCJ is used in [50, 62, 96] for performance analysis of several load-balancing Cloud schedulers. The value and importance of the GoCJ dataset can be enumerated as follows:

1. The GoCJ dataset provides a reflection of real workload behavior as perceived in Google cluster traces [88–93] and MapReduce logs [92] from the M45 supercomputing cluster, so it has more significance and usefulness for researchers in the scheduling of cluster and Cloud-based applications;

2. The GoCJ dataset can serve as an alternative to benchmark workload for scheduling and resource-allocation mechanisms using realistic HPC jobs in Cloud computing.

3.1.1 GoCJ Overview

The GoCJ dataset is provided as a supplementary data in text and Excel file formats in amalgamation with the two dataset generator files: 1) an Excel worksheet generator, and 2) a Java tool generator. Each row in the text file describes the size of a specific job in terms of MIs. The Monte Carlo simulation method [115] is employed to generate the dataset comprised of any required number of jobs. The specification of the GoCJ dataset is presented in Table 3.1. The GoCJ dataset,

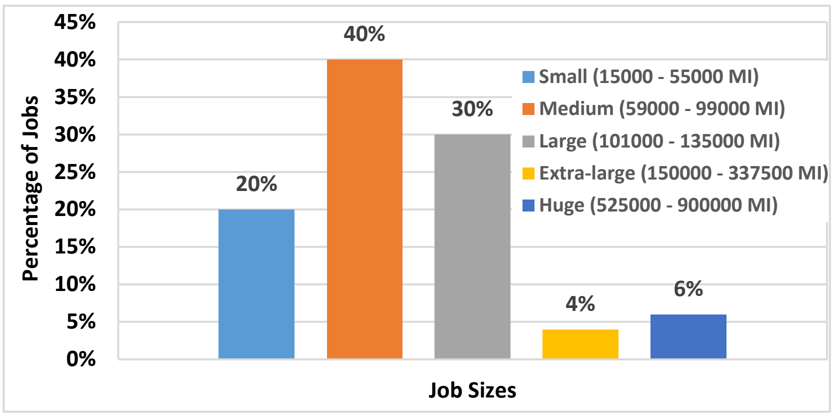


FIGURE 3.1: The Composition of GoCJ Dataset.

stored in a Mendeley Data repository, is comprised of 21 text files. Each dataset file is named as “GoCJ_Dataset_XXX.txt”, where XXX is the number of jobs in the file i.e., “GoCJ_Dataset_100.txt” has the sizes of 100 jobs. Each text file consists of a set of rows, where each row has a numeric value presenting the size of a job in terms of MI. Job completion time for GoCJ dataset follows a long-tailed distribution (with 90% of the jobs completing on average within 1.6 min). The longest-executing job witnessed in the dataset lasts up to 15 min (i.e., 6% of the jobs execute for less than 5 min and 4% of the jobs execute for 15 min). The average size of a job in the GoCJ dataset is 5 min. Fig. 3.1 displays ratios and sizes of jobs distribution in GoCJ dataset in terms of percentage and MIs, respectively.

3.1.2 GoCJ Dataset Method

In this section, the data acquisition for input dataset and the method of generating the GoCJ dataset is described in detail. In addition, the complexity of and data distribution in GoCJ dataset is also presented.

3.1.2.1 Input Dataset

The Monte Carlo simulation method is used to generate the GoCJ dataset. A sample input dataset is formulated based on workload behavior in Google cluster traces, which is input into the Monte Carlo simulation. The composition of jobs in the simulated GoCJ dataset is generated based on the job sizes in the input dataset.

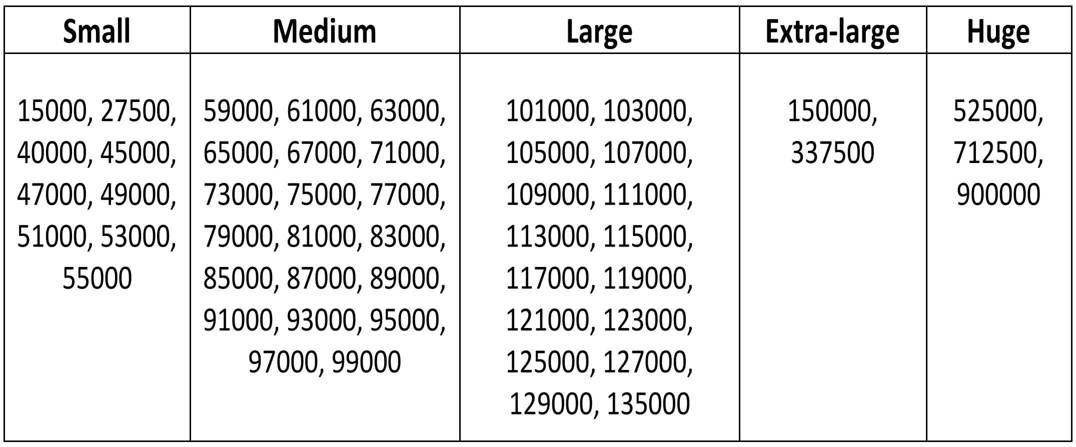
Based on the analysis of [88–91, 93], a GoCJ realistic data set is generated using the bootstrapped *Monte Carlo* (MC) simulation method [115]. Instead of *Random Number Generation* (RNG), the input dataset is repeatedly sampled by selecting one of the data points from the input dataset in bootstrapping [115]. A list of 50 different-size jobs (i.e., shown in Fig. 3.2) are identified and input into MC bootstrapping as the input dataset. Each job size in the dataset is treated with equal probability in repeated sampling by bootstrapping. The average power of machine in the distributed computing environment for finding job sizes is assumed as 1000 Million Instructions Per Second (MIPS). The majority of the smaller jobs in Google-like realistic dataset (i.e., 90% of jobs) execute for up to 1.6 min. However, the longest-executing jobs in the GoCJ dataset execute for up to 15 min (i.e., 900,000 MIs / 1000 MIPS = 900 s = 15 min). The sizes of jobs in the GoCJ dataset is presented in terms of MIs instead of *ETC* as used in other available datasets [24, 98]. The job size is calculated from the *ETC* of the job, using the following relationship, where *Job_MI* presents the size of job in MIs, *Machine_MIPS* is the power of computing machine in MIPS, and *ETC* is the expected time to complete a job in seconds [50]. The job size can be calculated using Equation 3.1.

$$Job\_MI = Machine\_MIPS \times ETC\_Seconds \tag{3.1}$$

The converse of Equation 3.1 is to determine the *ETC* of a job, which is presented [50] as:

$$ETC\_Seconds = \frac{Job\_MI}{Machines\_MIPS} \tag{3.2}$$

The job sizes in HCSP instances [95], GWA-T traces [99], ETM datasets [98], Facebook Hadoop workload [100], and Yahoo cluster traces [102] are presented in terms of ETC. On the other hand, the composition of the input dataset is presented in Fig. 3.2 (which is derived using Equation 3.1). Fifty different job sizes are



Small	Medium	Large	Extra-large	Huge
15000, 27500, 40000, 45000, 47000, 49000, 51000, 53000, 55000	59000, 61000, 63000, 65000, 67000, 71000, 73000, 75000, 77000, 79000, 81000, 83000, 85000, 87000, 89000, 91000, 93000, 95000, 97000, 99000	101000, 103000, 105000, 107000, 109000, 111000, 113000, 115000, 117000, 119000, 121000, 123000, 125000, 127000, 129000, 135000	150000, 337500	525000, 712500, 900000

FIGURE 3.2: Sizes of Jobs in Input Dataset for GoCJ (in MIs)..

identified based on the analysis of Google cluster traces with an equal probability of occurrences in the desired GoCJ dataset. Using Equation 3.1 by considering the computing power of machine as 1000 MIPS and the ETC of jobs (derived from workload behavior studied in Google Cloud infrastructure [88–91, 93]), the input dataset is created and presented in Fig. 3.2.

3.1.2.2 GoCJ Dataset

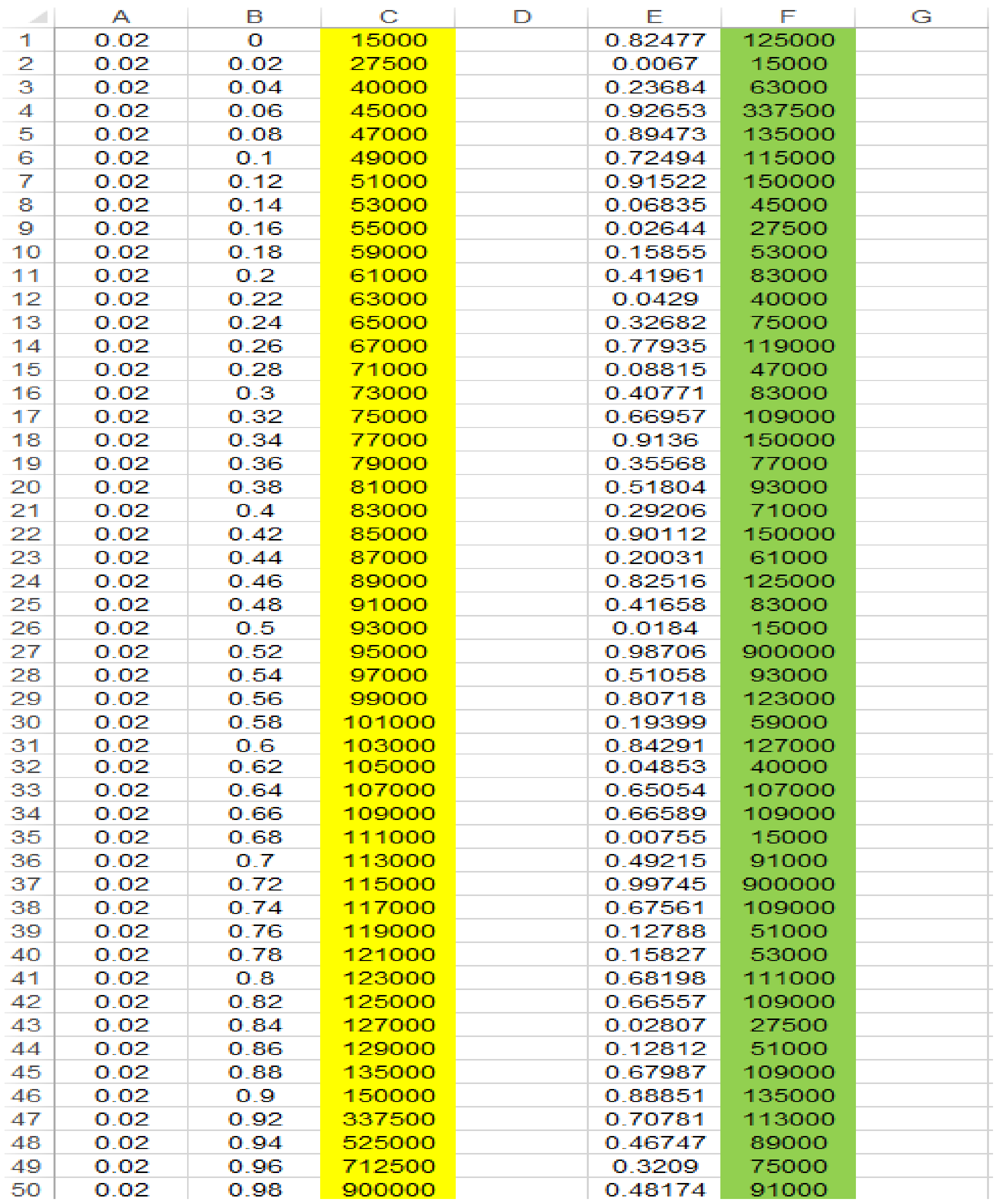
The dataset is generated by bootstrapped MC simulation using an Excel worksheet (as shown in Fig. 3.3). The input dataset is input in column C from cell C1 through C50, highlighted with a yellow background in Fig. 3.2. The individual probability of occurrence of each job is placed in the corresponding row of column A (i.e., from cell A1 through A50). Congruently, the cumulative probability of each job is placed in column B from cell B1 through B50. Therefore, a data table (i.e., from cell B1 through C50) is generated where each job size in column C is referenced by a cumulative probability in column B. As presented in column I in the worksheet, uniform RNG is used to generate a random number among the indices of job sizes in the data table. Now, a built-in function VLOOKUP in Excel is used to create the GoCJ dataset based on the input dataset provided in column C. The VLOOKUP function inputs the generated random number (i.e., in the corresponding row of column E), and the data table (i.e., from cells B1 through C50). Then, VLOOKUP searches for the entry in the data table based on the input random number (i.e., search in cumulative probabilities in column B) and returns the corresponding job-size entry to store it in column F, highlighted with a green background as shown in Fig. 3.2. The VLOOKUP-based formula used to find a job size is as follows:

$$F1 = VLOOKUP(E1, \$B\$1 : \$C\$50, 2) \tag{3.3}$$

The GoCJ dataset with the specified number of jobs can be created by extending the formulas in cells $E1$ and $F1$ by copy/paste to the row equal to the desired number of jobs in the dataset.

3.1.2.3 GoCJ Dataset Tool

An automated dataset generator program is provided using Java programming. The algorithm is presented as Algorithm 3.1, which presents the formal algorithm for creating a dataset as discussed in Section 3.1.1. The GoCJ generator performs the necessary initialization (lines 1–4). cPer variable presents the cumulative percentage of probability for each job in the input dataset to occur in the simulated dataset, jobSize is the size of the job to occur in the simulated dataset, and jList is the list of jobs finally produced in the simulated dataset. Afterward, the input dataset from a text file named "Original_Dataset" is read and resided in bufferReader (in lines 5–6). Line 5 reads all the job sizes in the input dataset derived from the Google cluster traces. A while-loop is used to copy the sample job sizes from bufferReader to fill the dataTable. The dataTable contains the job sizes in input dataset along with its cumulative probability (i.e., lines 7–10 of Algorithm 3.1). Line 9 maps the cumulative probability of each job in the dataTable. The second while-loop is used to produce the GoCJ dataset with the desired number of jobs. The job size produced in each iteration of the loop (i.e., lines 12–15) is stored in a job list (i.e., jList variable). To scrutinize the complexity and efficiency of the GoCJ generator, N number of desired jobs are considered in the GoCJ dataset. As mentioned in Section 3.1.2.1, the average computing power of



	A	B	C	D	E	F	G
1	0.02	0	15000		0.82477	125000	
2	0.02	0.02	27500		0.0067	15000	
3	0.02	0.04	40000		0.23684	63000	
4	0.02	0.06	45000		0.92653	337500	
5	0.02	0.08	47000		0.89473	135000	
6	0.02	0.1	49000		0.72494	115000	
7	0.02	0.12	51000		0.91522	150000	
8	0.02	0.14	53000		0.06835	45000	
9	0.02	0.16	55000		0.02644	27500	
10	0.02	0.18	59000		0.15855	53000	
11	0.02	0.2	61000		0.41961	83000	
12	0.02	0.22	63000		0.0429	40000	
13	0.02	0.24	65000		0.32682	75000	
14	0.02	0.26	67000		0.77935	119000	
15	0.02	0.28	71000		0.08815	47000	
16	0.02	0.3	73000		0.40771	83000	
17	0.02	0.32	75000		0.66957	109000	
18	0.02	0.34	77000		0.9136	150000	
19	0.02	0.36	79000		0.35568	77000	
20	0.02	0.38	81000		0.51804	93000	
21	0.02	0.4	83000		0.29206	71000	
22	0.02	0.42	85000		0.90112	150000	
23	0.02	0.44	87000		0.20031	61000	
24	0.02	0.46	89000		0.82516	125000	
25	0.02	0.48	91000		0.41658	83000	
26	0.02	0.5	93000		0.0184	15000	
27	0.02	0.52	95000		0.98706	900000	
28	0.02	0.54	97000		0.51058	93000	
29	0.02	0.56	99000		0.80718	123000	
30	0.02	0.58	101000		0.19399	59000	
31	0.02	0.6	103000		0.84291	127000	
32	0.02	0.62	105000		0.04853	40000	
33	0.02	0.64	107000		0.65054	107000	
34	0.02	0.66	109000		0.66589	109000	
35	0.02	0.68	111000		0.00755	15000	
36	0.02	0.7	113000		0.49215	91000	
37	0.02	0.72	115000		0.99745	900000	
38	0.02	0.74	117000		0.67561	109000	
39	0.02	0.76	119000		0.12788	51000	
40	0.02	0.78	121000		0.15827	53000	
41	0.02	0.8	123000		0.68198	111000	
42	0.02	0.82	125000		0.66557	109000	
43	0.02	0.84	127000		0.02807	27500	
44	0.02	0.86	129000		0.12812	51000	
45	0.02	0.88	135000		0.67987	109000	
46	0.02	0.9	150000		0.88851	135000	
47	0.02	0.92	337500		0.70781	113000	
48	0.02	0.94	525000		0.46747	89000	
49	0.02	0.96	712500		0.3209	75000	
50	0.02	0.98	900000		0.48174	91000	

FIGURE 3.3: GoCJ Excel Worksheet Generator.

a machine that is used to generate job sizes in the input dataset is 1000 MIPS. Algorithm 3.1 inputs two parameters; first num is desired number of jobs in the simulated dataset and second is the Original_Dataset with a fixed number of jobs (i.e., 50 jobs). Therefore, the first while-loop in Algorithm 3.1 iterates for a fixed number of times (i.e., equal to the number of job sizes in the input dataset). On the other hand, the second while-loop iterates till N that is the desired number of jobs in the simulated dataset. The computational complexity of GoCJ generator is linear or O(N) depending only on the desired number of jobs in the simulated

Algorithm 3.1: GoCJ Generator

```
Input: $num$ - desired number of jobs in dataset,
$Original\_DataSet$ - file of original dataset sample
Output: $jList$ - list of job sizes in the desired dataset
1  $cPer = 0$
2  $jobSize = 0$
3  $jList = Null$
4  $dataTable < cPer, jobSize >= Null$
5  $fileReader$ = readFile($Original\_DataSet$)
6  $bufferReader$ = read($fileReader$)
7  while $bufferReader$ is Not Empty do
8      $jobSize = long.parseLong$(bufferRedear.readLine())
9      $dataTable.add$(cPer,jobSize)
10     $cPer = cPer + 2$
11 $a = 1$
12 while $num \geq a$ do
13     $rand = Random.nextInt(100)$
14     $jList.add((rand$ Mod 2)?$dataTable.get(rand) : getJobSize(rand))$
15     $a++$
16 return $jList$
```

dataset. Due to the linear complexity of GoCJ generator, it will not create any overhead even for a dataset creation with a large number (N) of jobs.

3.1.3 Availability of GoCJ Dataset

To address the research question 1, the GoCJ dataset is kept publically available on following two repositories since September 26, 2018: **1)** Mendeley Repository with DOI: 10.17632/b7bp6xhrcd.1 [1]. **2)** MDIP Data in Supplementary Material.

The source code of the given algorithm in Java program is also available on Mendeley along with the GoCJ dataset files and provided with the supplementary files as well. One Excel file, named "GoCJ_Dataset_Monte_Carlo.xlsx", is also placed along with the text files in the dataset. This Excel file can be used to generate a GoCJ dataset comprised of the desired number of jobs. New dataset instance can be created by copying any worksheets in the "GoCJ_Dataset_Monte_Carlo.xlsx"

[1] $https://data.mendeley.com/datasets/b7bp6xhrcd/1$

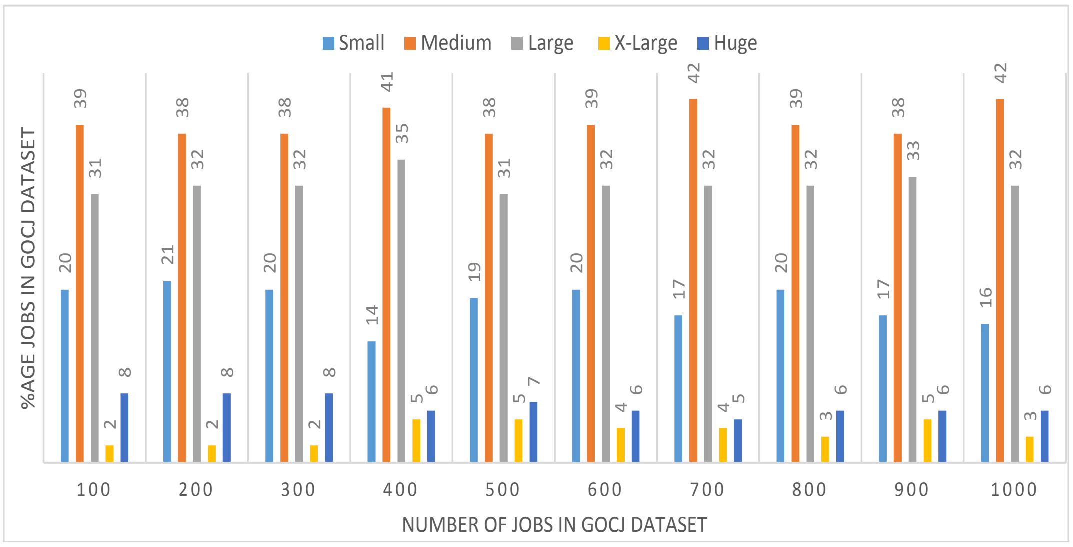


FIGURE 3.4: Ratios of Jobs Distribution in the GoCJ Dataset.

Excel file, and then extending the formulas in column E and F by copy/paste, as discussed.

3.1.4 GoCJ Data Distribution

To test the compliance of the data distribution trend in the input dataset to the simulated dataset produced by the GoCJ generator, the covariance statistical test is performed. The average data distribution of 19 GoCJ dataset files is determined and its covariance with the input dataset is calculated to find the correlation of data distribution. The covariance of the input dataset to the average simulated dataset is 2.49, demonstrating that both the original and simulated datasets are positively dependent. Furthermore, the ratios of job-size distribution are also presented in Fig. 3.4 to highlight the job distribution trends in original and simulated datasets. Another statistical measure that displays the data distribution based on five-number summary (i.e., minimum, first quartile, median, third quartile and maximum) is boxplot visualization. The five-number summary for input datasets and ten sample datasets produced using GoCJ generated is determined. To reflect the job-size trends in the original and simulated dataset, a boxplot visualization is shown in Fig. 3.5. The boxplot presents the median of job sizes in the original,

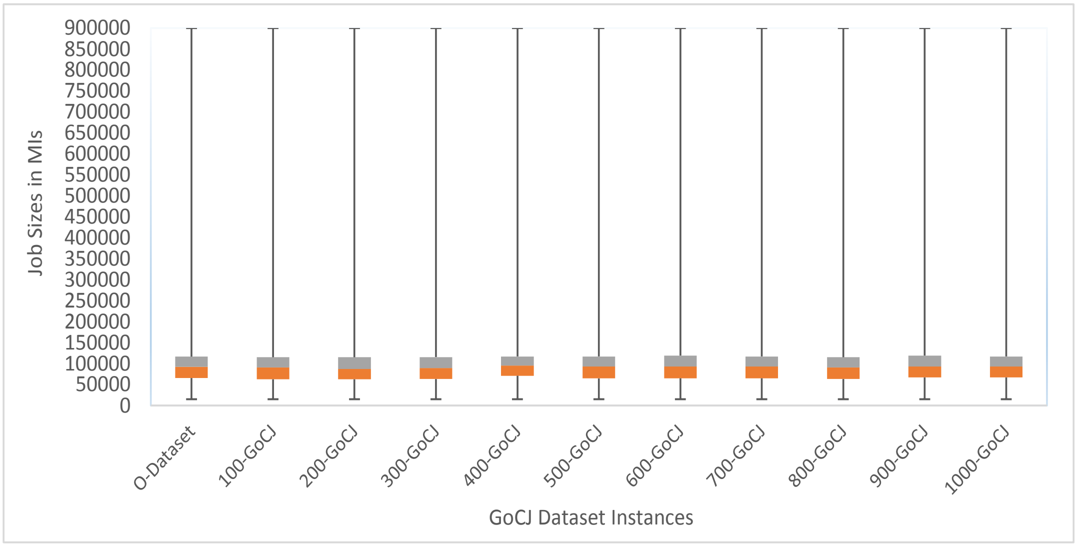


Figure 3.5: Boxplot Visualization of Jobs Distribution in Input and Generated GoCJ Datasets.

and ten simulated GoCJ datasets which are 92,000, 91,000, 87,000, 89,000, 95,000, 93,000, 93,000, 93,000, 91,000, 93,000, and 93,000 MIs, respectively. These job sizes belong to the medium job sizes as shown in Fig. 3.1 and 3.2. However, the minimum and maximum job sizes in both original and simulated GoCJ datasets are same as 15,000 and 900,000 MIs, respectively. Similarly, the first quartile and third quartile of original and simulated GoCJ datasets belongs to a category of medium- and large-size jobs as presented in Fig. 3.1. Thus, the boxplot visualization proves that the data distribution in the simulated GoCJ datasets complies with the data distribution in the input dataset in the same manner.

3.2 Resource-Aware Load Balancing Algorithm (RALBA)

This section presents a detailed overview of the proposed RALBA scheduling heuristic. The system architecture, overall system and performance model, algorithm, computational complexity, and overhead analysis are presented in this section.

3.2.1 Overview

RALBA is a resource-aware load balancing algorithm that is comprised of two sub-schedulers: Fill and Spill. Fig. 3.6 presents the RALBA based Cloud computing system architecture. A renowned simulator CloudSim [1] is employed to evaluate the performance of the proposed RALBA scheduler. Cloudlet is used as a synonym for the job in CloudSim simulator. Physical and virtual instance layers provide a foundation for the delivery of IaaS and PaaS to the Cloud users [1]. The computing power of a Cloud datacenter is represented as a collection of physical host machines along with the storage servers at the physical instance layer. These resources are transparently managed by harnessing the virtualization concept to provide dynamic sharing of computing and storage resources at the virtual instance layer. A Cloud resource manager maintains the track and records the status of VMs in the virtual instance layer. The resource manager is accountable for creation, termination, and migration of VMs when required. Cloud resource manager provides to RALBA the information pertaining to the available VMs and their computing capabilities. To utilize the VMs for their full capacity, a resource-aware scheduler is required on top of the virtual instance layer for a balanced distribution of Cloudlets among VMs. The system accessibility layer provides a user-friendly interface for the submission of HPC based Cloudlet instances to the Cloud. To improve resource utilization and balanced job scheduling, we propose RALBA on top of the virtual instance layer (as shown in Fig. 3.6). RALBA system and performance model are presented using mathematical expressions. The basic definitions, terminologies, and the notations used in the mathematical expressions are listed in the Symbols section of the thesis.

3.2.2 System Architecture

RALBA is based on batch dynamic scheduling technique wherein a batch of Cloudlets is formulated and a balanced mapping of Cloudlets is performed. The system architecture of RALBA is depicted in Fig. 3.7. RALBA comprises of two

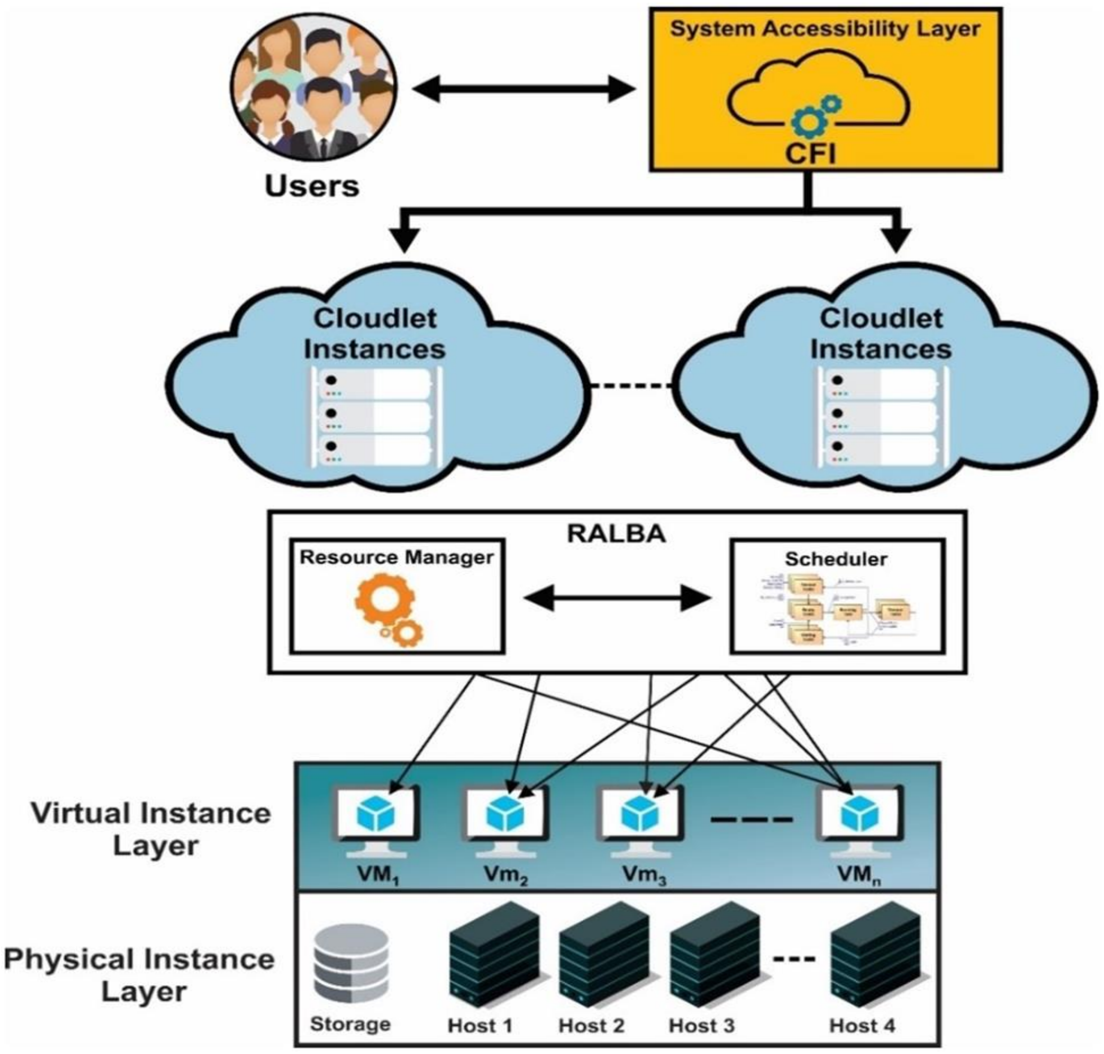


FIGURE 3.6: Cloud Computing System Architecture with RALBA.

sub-schedulers i.e., Fill and Spill schedulers. Fill Scheduler performs Cloudlet to VM allocation via considering the computing share of VMs. Fill scheduler selects VM_j with Largest_VMShare and determines $maxPCloudletVM_j$ for VM_j. The candidate Cloudlet to VM allocation is performed and $VMShare_j$ of VM_j is modified after allocation of the Cloudlet. Fill scheduler repeats the process of Cloudlets mapping until there does not exist VM_j with non-empty $RPCloudlet_j$ or the CLS becomes empty. Spill scheduler performs Cloudlets to VMs allocation based on EFT of candidate Cloudlet. After Fill scheduler, RALBA system switches to the Spill scheduler to allocate the remaining Cloudlets of CLS. The maxCloudlet is selected and allocated to the specific VM that produces EFT for this maxCloudlet. On candidate Cloudlet—VM allocation, the completion time

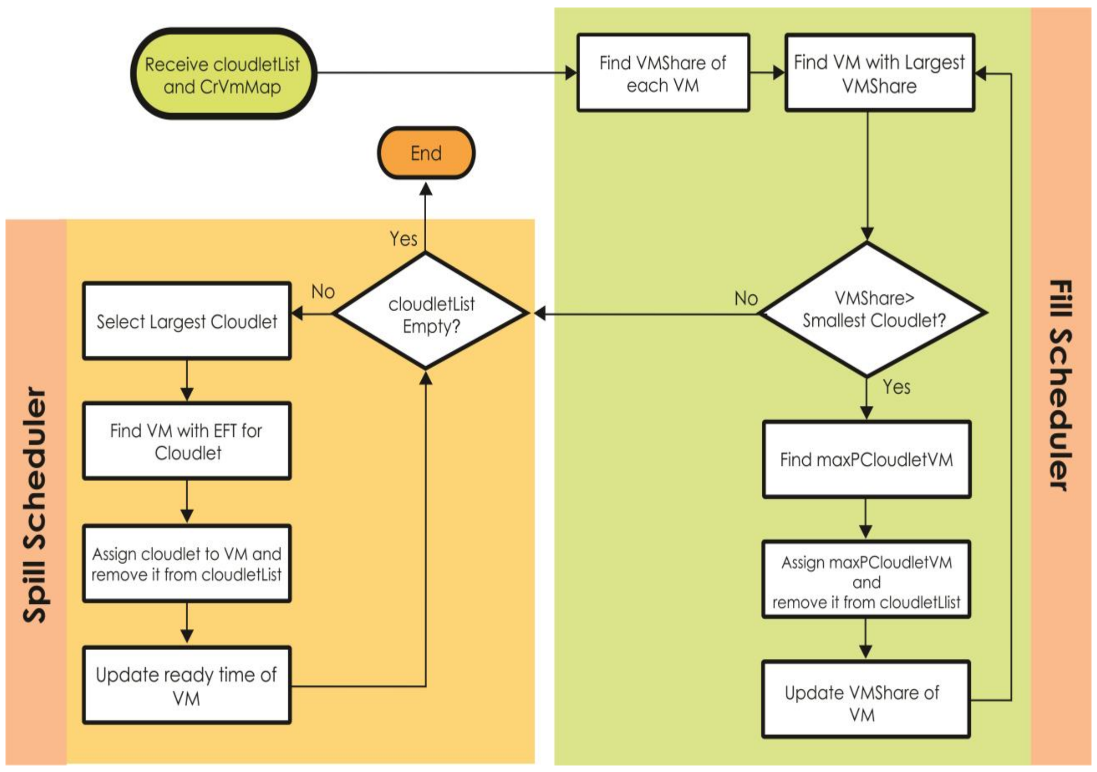

Receive cloudletList and CrVmMap
Find VMShare of each VM
Find VM with Largest VMShare
End
Yes
No
cloudletList Empty?
No
VMShare> Smallest Cloudlet?
Yes
Select Largest Cloudlet
Find VM with EFT for Cloudlet
Assign cloudlet to VM and remove it from cloudletList
Update ready time of VM
Find maxPCloudletVM
Assign maxPCloudletVM and remove it from cloudletList
Update VMShare of VM
Spill Scheduler
Fill Scheduler

FIGURE 3.7: RALBA System Architecture.

of the VM is updated. Cloudlet to VM allocation is repeated until CLS becomes empty.

The computation share based allocation of jobs to VMs by Fill scheduler, enables RALBA to produce a load-balanced schedule with improved resource utilization in amalgamation with improved makespan and throughput. This is evident in the performance assessment of RALBA technique against existing scheduling in Cloud computing (presented in Section 5.1 of Chapter 5).

3.2.3 System Model

The basic definitions, terminologies, and the notations used in the in system model of RALBA are listed in Table 3.2. A unified Cloud model is formed to delineate the performance of RALBA based Cloudlet scheduling. VMS=$\{VM_1$,VM_2,...,$VM_M\}$ represents a set of VMs in a Cloud, where M is the total number of VMs and a specific VM can be represented as VM_j $(1 \leq j \leq M)$. VMS represents the computing resources responsible for the execution of Cloudlets. The set of cloudlets

TABLE 3.2: Preliminary Notations used in RALBA System Model.

Notations	Descriptions
VMS	Set of VMs in a datacenter.
$vmCrMap$	Set of sorted VMs with computing ratio.
$vmCrMap_j$	Computating Ratio of VM_j.
CLS	Set of all cloudlets (to be scheduled).
$Cloudlet_i.MI$	Size of $Cloudlet_i$ in MI.
$VM_j.MIPS$	Computing power of VM_j in MIPS.
$RPCloudlet_j$	Remaining possible cloudlets (may map on VM_j).
$maxPCloudletVM_j$	Largest cloudlet in $RPCloudlet_j$ (mapped to VM_j).
$minCloudlet$	Smallest sized Cloudlet (to be scheduled).
$maxCloudlet$	Largest sized Cloudlet (to be scheduled).
$VMShare$	Set of sorted VMs with computing share.
$VMShare_j$	Computing share of VM_j (in terms of MI).
$Largest\_VMShare$	Largest computing share for any VM in VMS.
$Cloudlet\_CT\_i_j$	Expected completion time of $Cloudlet_i$ on VM_j.
VM_CT_j	Completion time of VM_j.
$Cloudlet\_EFT_i$	Earliest finish time of $Cloudlet_i$.
$Fill\_Scheduler$	Maps $Cloudlet_i$ to VM_j based on $VMShare_j$.
$Spill\_Scheduler$	Maps $Cloudlet_i$ to VM_j based on $Cloudlet\_EFT_i$.

is presented as CLS=$\{Cloudlet_1$, $Cloudlet_2$,...,$Cloudlet_N\}$, where N is the total number of Cloudlets and a specified Cloudlet can be represented as $Cloudlet_i$ $(1 \leq i \leq N)$. A resource manager in Cloud is responsible to provide the computation ratios (vmCrMap) of all VMs to RALBA [50], where $vmCrMap_j$ can be computed as:

$$vmCrMap_j = \frac{VM_j.MIPS}{\sum_{k=1}^{M} VM_k.MIPS} \tag{3.4}$$

$vmCrMap$ is used to maintain a balanced workload allocation to VMs and assists in calculating the computing share of all VMs (VMShare). The $VMShare_j$ [50]

of VM_j is mathematically expressed as:

$$VMShare_j = \sum_{i=1}^{N} Cloudlet_i.MI \times vmCrMap_j \tag{3.5}$$

Fill scheduler allocates Cloudlet to VM based on $VMShare$. The $RPCloudlet_j$ of a specified VM_j can be computed as:

$$RPCloudlet_j = \{Cloudlet|Cloudlet \in CLS, Cloudlet \leq VMShare_j\} \tag{3.6}$$

$maxPCloudletVM_j$ in each scheduling decision of Fill scheduler can be computed as:

$$maxPCloudletVM_j = \max_{\forall p \in RPCloudlet_j} Size(p) \tag{3.7}$$

In addition, on each scheduling decision of Fill and Spill schedulers, the candidate Cloudlet is removed from the list of Cloudlets with Equation 3.8 [50]:

$$CLS = \begin{cases} CLS - maxPCloudletVM_j, & \text{Using Fill\_Schduler} \\ CLS - maxCloudlet, & \text{Using Spill\_Scheduler} \end{cases} \tag{3.8}$$

$minCloudlet$ and $maxCloudlet$ are computed using Equation 3.9 and Equation 3.10, respectively.

$$minCloudlet = \min_{\forall c \in CLS} Size(c) \tag{3.9}$$

$$maxCloudlet = \max_{\forall c \in CLS} Size(c) \tag{3.10}$$

On Cloudlet—VM allocation, $VMShare_j$ of VM_j is modified by Fill Scheduler [50] as:

$$VMShare_j = \begin{cases} VMShare_j - Cloudlet.MI, & \text{Using Fill\_Scheduler} \\ \text{Switch to Spill\_Scheduler}, & \text{Otherwise} \end{cases} \tag{3.11}$$

Fill scheduler selects the VM with largest $VMShare_j$ in each scheduling decision.The $Largest\_VMShare$. In each scheduling decision of Fill scheduler, the

VM with largest $VMShare_j$ is selected. The $Largest\_VMShare$ can be computed [50] as:

$$Largest\_VMShare = \max_{\forall j \in \{1,2,3,..,M\}} (VMShare_j) \tag{3.12}$$

Spill scheduler uses $Cloudlet\_EFT_i$ to allocate the remaining Cloudlets to VMs. The $Cloudlet\_EFT_i$ of a specified $Cloudlet_i$ is computed using the mathematical relation:

$$Cloudlet\_EFT_i = \max_{\forall j \in \{1,2,3,..,M\}} (Cloudlet\_CT_{ij}) \tag{3.13}$$

where $Cloudlet\_CT_{ij}$ is the expected completion time of $Cloudlet_i$ on VM_j. The $Cloudlet\_CT_{ij}$ is mathematically defined and expressed [50] as:

$$Cloudlet\_CT_{ij} = (\frac{Cloudlet_i.MI}{VM_j.MIPS}) + (VM\_CT_j) \tag{3.14}$$

The VM_CT_j is the completion time of VM_j for already assigned workload prior to the allocation of $Cloudlet_i$ and computed [50] as:

$$VM\_CT_j = \sum_{i=1}^{N} \frac{Cloudlet_i.MI \times F[i,j]}{VM_j.MIPS} \tag{3.15}$$

Where F[i, j] is a Boolean variable that determines the allocation of $Cloudlet_i$ to VM_j and represented in Equation 3.16.

$$F[i,j] = \begin{cases} 1, & \text{Cloudlet i is assigned to } VM_j \\ 0, & \text{Otherwise} \end{cases} \tag{3.16}$$

3.2.4 Performance Model

This study evaluates RALBA based scheduling using makespan, throughput, and resource utilization performance metrics. Mathematical presentation of makespan is [35, 37]:

$$Makespan = \max_{\forall j=1,2,3,...,M} (VM\_CT_j) \tag{3.17}$$

Throughput definition is presented in Equation 3.18. A higher throughput value indicates high workload execution per unit time.

$$Throughput = \frac{N}{Makespan} \tag{3.18}$$

where, N represents total Cloud jobs executed. ARUR [35, 37] is a measure to present the overall utilization of Cloud (expressed in Equation 3.19). The value of ARUR remains between 0 and 1. The higher ARUR value represents higher resource utilization in Cloud.

$$ARUR = \frac{\frac{\sum_{j=1}^{M} VM\_CT_j}{M}}{Makespan} \tag{3.19}$$

3.2.5 RALBA Algorithm

This section elaborates the proposed scheduling algorithm RALBA (shown in Algorithm 3.2) with its two primary modules i.e., Fill Scheduler (presented in Algorithm 3.3) and Spill Scheduler (shown in Algorithm 3.4). The data items harnessed in RALBA, Fill Scheduler, and Spill Scheduler are illustrated in the Symbol section of the thesis. List of Cloudlets and list of all VMs computation ratio (provided

Algorithm 3.2: RALBA

```
Input: $vmCrMap$ - set of VMs with computation ratio,
$cloudletList$ - list of cloudlets $c_1, c_2, \ldots, c_n$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping
1 $cloudletVmMap = Null$
2 $cloudletVmMap =$ FillScheduler($vmCrMap$,$cloudletList$)
3 $vmList =$ getVmList($vmCrMap$)
4 if $cloudletList.size() \geq 1$ then
5     $cloudletVmMap =$SpillScheduler($vmList$,$cloudletList$,$cloudletVmMap$)
6 return $cloudletVmMap$
```

by Cloud resource manager) are used as input parameters in RALBA (Algorithm 3.2). To perform mapping of Cloudlets to VMs, RALBA invokes Fill scheduler (line 2 of Algorithm 3.2) and then invokes the Spill scheduler (line 5 of Algorithm 3.2).

Algorithm 3.3: FillScheduler

```
Input: $vmCrMap$ - set of VMs with computation ratio ,
$cloudletList$ - list of cloudlets $c_1, c_2, \ldots, c_n$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping
$totalLength = 0$
$newVShare = 0$
$cloudletVmMap = Null$
$vShareMap < v, share >= Null$
forall $cloudlet$ in $cloudletList$ do
    $totalLenght$ = totalLenght + cloudlet.getCloudletLength()
forall $v$ in $vmList$ do
    $vShareMap_v$ = totalLenght * $vmCrMap_v$
while $getLargeShare(vShareMap) \geq getMinCloudlet(cloudletList)$ do
    $v = getLargeShareVm$(vShareMap)
    $cloudlet = getMaxPCloudletVm$(v,cloudletList)
    $cloudletVmMap.add$(cloudlet,v)
    $newVShare = vShareMap.get$(v)$-$ cloudlet.$getCloudletLength$
    $vShareMap.modify$(v,newVShare)
    $cloudletList.remove$(cloudlet)
return $cloudletVmMap$
```

Algorithm 3.4: SpillScheduler

```
Input: $cloudletList$ - list of cloudlets $c_1, c_2, \ldots, c_n$,
$vmList$ - list of VMs $v_1, v_2, \ldots, v_m$,
$cloudletVmMap$ - set of cloudlets to VMs mapping by $FillScheduler$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping
while $cloudletList.size() \geq 1$ do
    $cloudlet = getMaxCloudlet$(cloudletList)
    $v = getVmWithEFT$(cloudlet,vmList)
    $cloudletVmMap.add$(cloudlet,v)
    $cloudletList.remove$(cloudlet)
return $cloudletVmMap$
```

Fill Scheduler (Algorithm 3.3) performs the necessary initializations (lines 1–4) and computes the totalLength of all the Cloudlets in the submitted batch (lines 5–6). Afterward, the computing share of each VM (vShareMapv) is calculated (lines 7–8 of Algorithm 3.3) by using Equation 3.5. A while-loop (lines 9–15 of Algorithm 3.3) is used to select the VM with the largest computation share and to determine the maxPCloudletVm for scheduling. On each scheduling decision (lines 13–15 of Algorithm 3.3), the candidate Cloudlet is removed from the cloudletList and the vShareMapv of the candidate VM is modified (using Equation 3.11). This process (lines 9–15 of Algorithm 3.3) is repeated until the smallest size Cloudlet becomes greater than the largest vShareMapv in vShareMap.

After Cloudlets–VMs allocation by the Fill scheduler, the Spill Scheduler (Algorithm 3.4) is employed for the allocation of the remaining Cloudlets to VMs. While-loop (in line 1 of 3.4) allocates the remaining Cloudlets (in descending order of Cloudlets size) to VMs based on EFT. The maxCloudlet is selected on line 2 and the VM producing $Cloudlet\_EFT$ is determined in line 3. This candidate Cloudlet to VM allocation is performed in line 4 within each loop-iteration. At each scheduling decision, the candidate Cloudlet is removed from the cloudletList and the VM_CT_j of VM_j is modified (using Equation 3.15). This process is repeated until cloudletList becomes empty. RALBA produces a load-balanced schedule in the form of cloudletVmMap.

The source code of the RALBA is available on the Bitbucket Repository1 in a project named prjRALBA [2].

3.2.6 Complexity and Overhead Analysis

This section delineates the complexity and overhead analysis of the scheduling algorithms. To scrutinize the complexity of RALBA, we consider the total number of N Cloudlets and the total number of M VMs in a Cloud data-center.

In worst-case, Fill scheduler selects a VM with largest VMShare in M number of comparisons (as full list of VMs is iterated in worst case) and it determines maxPCloudletVM in N comparisons (as full list of Cloudlets is iterated in worst case). After Cloudlet—VM allocation, the $vShareMap_v$ is modified in vShareMap with a maximum of M comparisons. The time complexity of Fill scheduler is O($M^2.N$); where $M << N$ on the real Cloud. On the other hand, Spill scheduler selects the maxCloudlet from a sorted cloudletList and assigns to the VM producing EFT for it. In worst-case, when N Cloudlets are scheduled by Spill scheduler, the time

[2] $https://bitbucket.org/RALBA_18/ralba/src$

Table 3.3: Computational Complexity of Scheduling Algorithms [38, 39].

Algorithms	**Time Complexity**
RALBA	O($M^2.n + M.N - n$)
Sufferage	O($M.N^2$)
MCT	O($M.N$)
Min-Min	O($M.N^2$)
Max-Min	O($M.N^2$)
RASA	O($M.N^2$)
TASA	O($M.N^2$)
RR	O(N)
RS	O(N)

Table 3.4: Overhead Analysis of Scheduling Algorithms (N times of RR).

Algorithms	RS	MCT	RALBA	Max-Min	RASA	Min-Min	TASA	Sufferage
Overhead	1.01	1.24	1.43	2.05	3.45	6.40	11.11	13.36

complexity of Spill scheduler is O($M.N$). If n is the number of Cloudlets scheduled by Fill scheduler, then remaining $N - n$ Cloudlets will be scheduled by Spill scheduler. The computational complexity of RALBA becomes O($M^2.n+M.N-n$) [50]. Table 3.3 provides the computational complexities of RALBA and the existing scheduling algorithms.

Furthermore, we profile the simulation code of RALBA and other existing scheduling heuristics to collect the time overhead related to the scheduling decisions. RR presents the minimal scheduling overhead against the other heuristics. Table 3.4 delineates the relative scheduling overhead in terms of complexity order (N times of RR scheduling) from best to worst i.e., RS, MCT, RALBA, Max–Min, RASA, Min–Min, TASA, and Sufferage.

The Sufferage scheduling algorithm has the most expensive time overhead among the given set of algorithms. Considering the computational complexity (Table 3.3) and the overhead analysis (Table 3.4), RALBA is promised to schedule jobs in a scalable manner for large dataset.

3.3 SLA and Resource-Aware Load Balancing Algorithm (SLA-RALBA)

This section describes a comprehensive summary of the proposed SLA-RALBA technique that is based on cost-efficient and resource-aware scheduling. The overview, system and performance model, and algorithm of SLA-RALBA are presented. In addition, the discussion on computation complexities of SLA-RALBA and existing SLA-aware scheduling algorithms.

3.3.1 Overview

SLA-RALBA is a cost-efficient load balancing algorithm comprises two sub schedulers like in RALBA [50]: SLA-Fill and SLA-Spill schedulers.

Fig. 3.8 presents the SLA-RALBA based Cloud computing system architecture. The user job is known as Cloudlet in CloudSim; a renowned simulator used in this study. The virtual instance layers is hosted by physical instance layer that provides a foundation for the delivery of SLA-aware computing services to the Cloud users [45]. The Cloud datacenters are comprised of an assortment of physical host machines and storage media at the physical instances layer. The computing, communication and storage resources are provisioned on-demand in a scalable and dynamic manner to the Cloud users in the form of VMs formulating the visual instance layer. The resource manager, on the top of the virtualization layer, is held responsible for creation, termination, and migration of VMs when required. In addition, resource manager and Cloudlet scheduler allocate VMs as needed and records the cost estimation of provisioned services. Therefore, resource manager provides the information pertaining to the available VMs that are the computing capabilities and the incurred gain cost of each VM. To ensure the utilization of VMs with its full computing capacity and minimal execution cost, a cost-efficient and resource-aware load balancer is required on top of the Cloud virtualization for a balanced workload distribution.

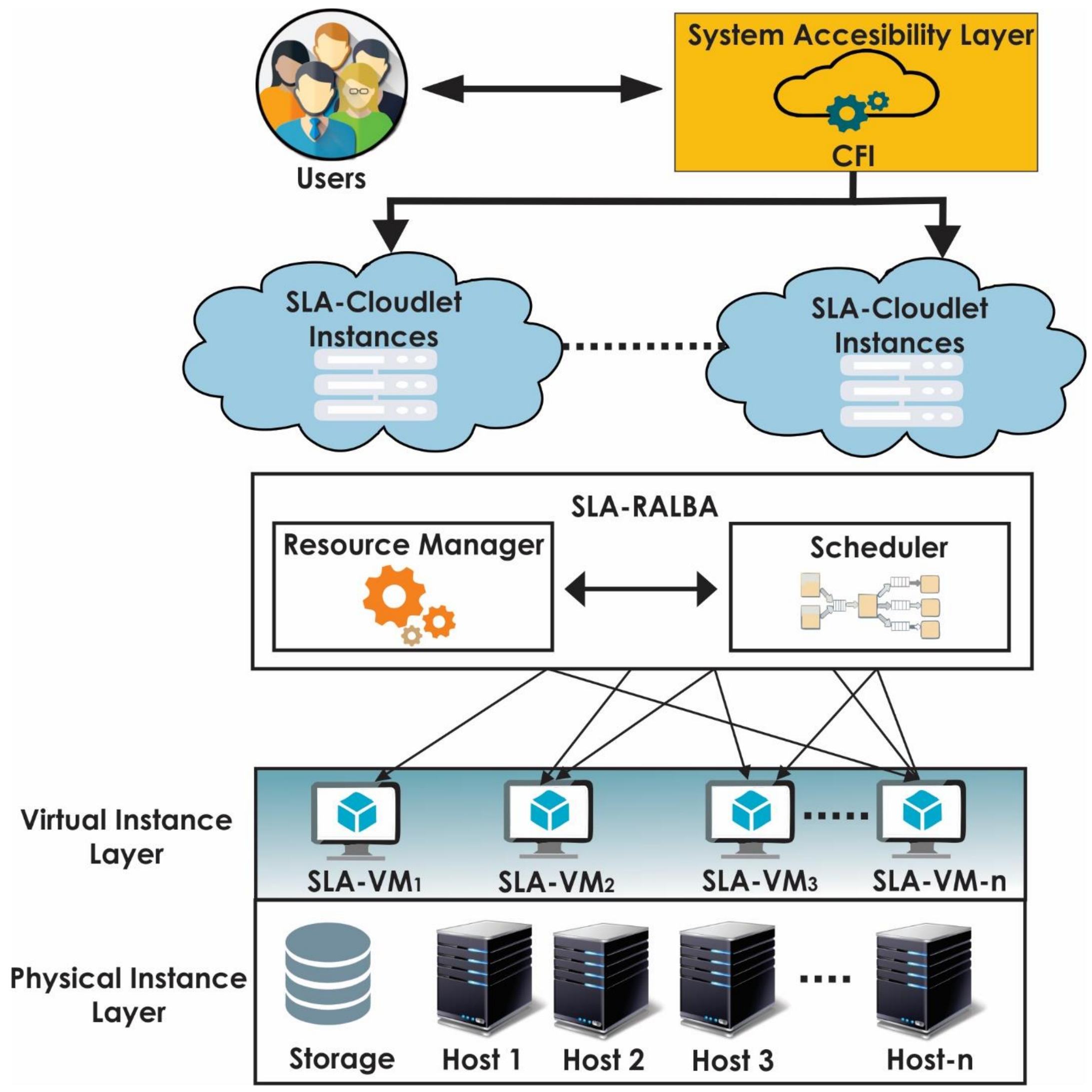


FIGURE 3.8: Cloud Computing System Architecture with SLA-RALBA.

A user-friendly interface is provided by system accessibility layer for the submission of SLA-aware Cloudlets on Cloud. SLA-RALBA scheduler is proposed on top of the SLA-aware virtualization to improve the resource utilization with a cost-efficient and load-balanced schedule ensuring the minimal execution time and cost (as shown in Fig. 3.8).

3.3.2 System Architecture

The system and performance model of proposed technique are presented using mathematical expressions. For easily and better understanding, the majority of basic definitions and terminologies used in RALBA are repeated for the representation of SLA-RALBA.

The system architecture of SLA-RALBA is depicted in Fig. 3.9. SLA-RALBA is comprised of two sub-scheduler i.e., SLA-Fill and SLA-Spill schedulers. SLA-Fill scheduler performs Cloudlet to VM allocation via considering the computing share of VMs. SLA-Fill scheduler selects VM_j with $Largest\_VMShare$ and determines the $maxPCloudlet$ for this VM_j. The $maxPCloudlet$ is checked for its SLA-type (i.e., SLA-1, SLA-2, or SLA-3). If the $maxPCloudlet$ is SLA-3, then the VM with $minEGain$ is determined for it. The maxPCloudlet is assigned to the VM with $minEGain$ and removed from the list of Cloudlets. The VMShare of the VM with minEGain is also updated. If the $maxPCloudlet$ is SLA-2, then the SLA-2 suitable VM is determined for it. The $maxPCloudlet$ is assigned to the SLA-2 suitable VM and removed from the list of Cloudlets. The $VMShare$ of the SLA-2 VM is also updated. If the $maxPCloudlet$ belongs to SLA-1, then the $maxPCloudlet$ is allocated to the VM_j with $Largest\_VMShare$ and removed from the list of Cloudlets. The VMShare of VM_j is updated accordingly. SLA-Fill scheduler repeats the process of Cloudlets mapping to VMs until there does not exist $maxPCloudlet$ for any of the VM in the computing setup.

SLA-Spill scheduler performs Cloudlets to VMs allocation based on descending order of Cloudlet sizes. The largest sized Cloudlet as $maxCloudlet$ is selected and it is checked for SLA-type. If the $maxCloudlet$ is SLA-3, then the $maxCloudlet$ is allocated to the economical VM (like mapped by SLA-Fill scheduler) and it is removed from the list of Cloudlets. If the $maxCloudlet$ is SLA-2, then the $maxCloudlet$ is assigned to the SLA-2 suitable VM and it is removed from the list of Cloudlets. If the maxCloudlet belongs to SLA-3, the $maxCloudlet$ is assigned to the VM producing EFT and it is removed from the list of Cloudlets. SLA-Spill scheduler repeats the Cloudlets to VMs mapping until the list of Cloudlets becomes empty.

As SLA-Fill scheduler allocates SLA jobs to VMs based on the computation share of each VM, which ensures load balancing in terms improved resource utilization.

This is evident in the performance assessment of SLA-RALBA technique against existing SLA-aware schedulers (presented in Section 5.3 of Chapter 5).

3.3.3 System Model

The Cloud has a service manager responsible for providing a unified service to the Cloud users over internet. The service manager knows the computation resources on the Cloud and the services are delivered to the Cloud user by deploying VMs in the datacenters. The Cloud system is comprised of a set of VMs that is VMS={VM_1,VM_2,...,VM_M}, where M is the number of VMs, VM_j $(1 \leq j \leq M)$ stands for VM No. j. The VMS is measured in terms of MIPS. However, the Expected Gain (EG) per unit time is the Gain Cost (GC) of executing user request on a specific VM_j on the Cloud. The user request on the Cloud is known as a Cloudlet. The set of cloudlets submitted by users for execution with some SLA levels is CLS={$Cloudlet_1$, $Cloudlet_2$,...,$Cloudlet_N$}, where N is the number of cloudlets, $Cloudlet_i$ $(1 \leq i \leq N)$ stands for Cloudet No. i. The Cloudlet is measured in terms of MIs. Each Cloudlet in the CLS belongs to one of the three SLA levels: SLA-1, SLA2, and SLA-3 as used in the [45]. The SLA-1 focuses only on execution time, however, SLA-3 focuses only on execution cost of the Cloudlet. SLA-2 is the combination of SLA-1 and SLA3, by considering both the execution time and cost of the Cloudlet in scheduling.

The computation share of VM_j is represented as $VMShare_j$ and mathematically expressed in Equation 3.5. The list of remaining possible Cloudlets those could be allocated to a particular VM_j is represented as $RPCloudlet_j$ and mathematically expressed in Equation 3.6. Largest possible Cloudlet for VM_j in remaining set of Cloudlets is represented as $maxPCloudlet4VM_j$ and can be determined using Equation 3.7. The VM_j with $minEGain_i$ for $Cloudlet_i$ is determined using Equation 3.20, where $(Cloudlet_i.MI \leq VMShare_j)$:

$$minEGain_i = \min_{\forall j \in VMS} f(Cloudlet_i.EGain_{ij}) \tag{3.20}$$

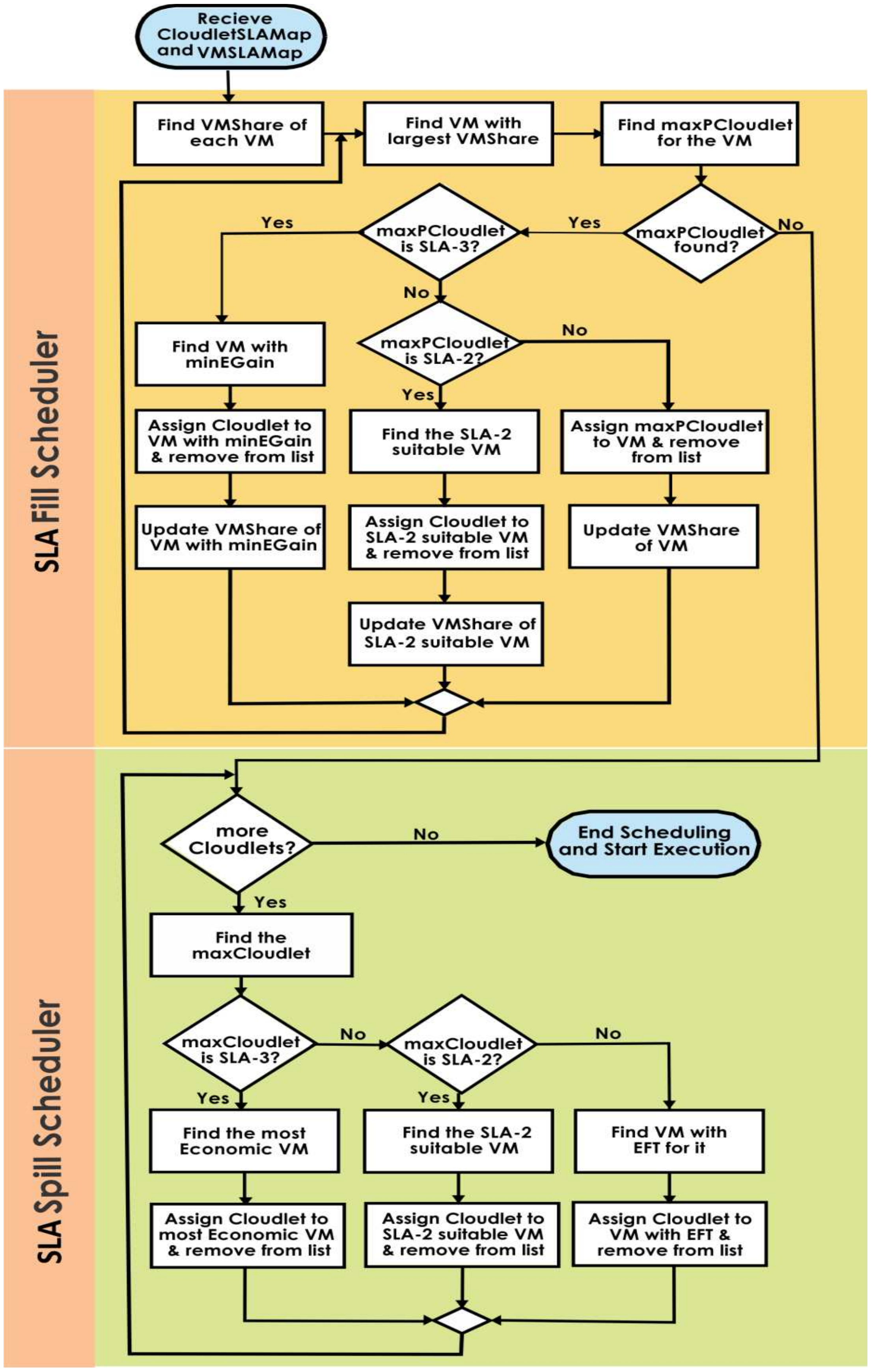

Recieve CloudletSLAMap and VMSLAMap
SLA Fill Scheduler
Find VMShare of each VM
Find VM with largest VMShare
Find maxPCloudlet for the VM
maxPCloudlet found?
Yes
No
maxPCloudlet is SLA-3?
Yes
No
Find VM with minEGain
Assign Cloudlet to VM with minEGain & remove from list
Update VMShare of VM with minEGain
maxPCloudlet is SLA-2?
No
Yes
Find the SLA-2 suitable VM
Assign Cloudlet to SLA-2 suitable VM & remove from list
Update VMShare of SLA-2 suitable VM
Assign maxPCloudlet to VM & remove from list
Update VMShare of VM
SLA Spill Scheduler
more Cloudlets?
No
End Scheduling and Start Execution
Yes
Find the maxCloudlet
maxCloudlet is SLA-3?
No
maxCloudlet is SLA-2?
No
Yes
Yes
Find the most Economic VM
Assign Cloudlet to most Economic VM & remove from list
Find the SLA-2 suitable VM
Assign Cloudlet to SLA-2 suitable VM & remove from list
Find VM with EFT for it
Assign Cloudlet to VM with EFT & remove from list

FIGURE 3.9: SLA-RALBA System Architecture.

The EGain of $Cloudlet_i$ on a particular VM_j is determined with following relation [96].

$$Cloudlet.EGain_{ij} = VM_j.ECost \times \frac{Cloudlet_i.MI}{VM_j.MIPS} \tag{3.21}$$

The most economic VM_j [96] for $Cloudlet_i$ is the virtual machine that produces minimum cost to execute $Cloudlet_i$ in the computing sytem.

$$mostEconomicVM_i = \min_{\forall v \in \{1,2,3,..,M\}} f(Cloudlet_i.EGain_{iv}) \tag{3.22}$$

$VMShare_j$ is modified in each step of SLA-Fill-scheduler using Equation 3.11. SLA-Fill-scheduler selects the VM_j with largest $VMShare_j$ in each scheduling decision. The $Largest\_VMShare$ is computed using Equation 3.12.

3.3.4 Performance Model

Earliest finish time of $Cloudlet_i$ on any available VM is presented as $Cloudlet\_EFT_i$ and mathematically defined using Equation 3.13. The completion time of $Cloudlet_i$ on VM_j is presented as $Cloudlet\_CT_ij$ and mathematically expressed in Equation 3.14. The VM_CT_j is the completion time of VM_j that can be calculated using Equation 3.15. Makespan is a performance measure of scheduling techniques that is the completion time to execute the whole workload by a computing system. It is known as the completion time of a VM/resource in the computing system that finishes to execute all of its allocated jobs and get ready for next batch of jobs to be allocated for execution. Makespan is mathematically expressed in Equation 3.17. The performance metric ARUR is calculated using Equation 3.19. The value of ARUR lies between 0 and 1; the higher value of ARUR dipects higher resource utilization. The total Gain Cost of particular VM_j is presented as VM_EGain_j and mathematically expressed in Equation 3.23 [96], followed by the Gain Cost of the computing system presented as $EGain$ and defined in Equation 3.24 [96].

$$VM\_EGain_j = \sum_{i=1}^{Nj} Cloudlet.EGain_ij \times F[i,j] \tag{3.23}$$

$$EGain = \sum_{j=1}^{M} VM\_EGain_j \tag{3.24}$$

Where F[i,j] is a boolean variable presented in Equation 3.16, and $Cloudlet.EGain_{ij}$ is the Gain Cost of $Cloudlet_i$ on VM_j.

3.3.5 SLA-RALBA Algorithm

The proposed SLA-RALBA load balancing technique is explained with its two primary schedulers in this section i.e., SLA-Fill Scheduler and SLA-Spill Scheduler. The SLA-RALBA is presented in Algorithm 3.5. List of Cloudlets with its SLA types are submitted by Cloud users as cloudSLAList. The Cloud resource manager provides the list of all VMs computation ratio as input parameter to SLA-RALBA scheduler. Scheduling SLA-based Cloudlets on VMs, SLA-RALBA invokes SLA-Fill scheduler (line 2 of Algorithm 3.5), and then invokes the SLA-Spill scheduler (line 5 of Algorithm 3.5). The necessary initialization is accomplished by SLA-Fill

Algorithm 3.5: SLA-RALBA

```
Input: $vmCrMap$ - set of VMs with computation ratio,
$cloudletSLAList$ - list of SLA-cloudlets $c_1, c_2, c_3, \dots, c_n$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping
1 $cloudletVmMap = Null$
2 $cloudletVmMap$ = SLA-FillScheduler($vmCrMap$,$cloudletSLAList$)
3 $vmList$ = getVmList($vmCrMap$)
4 if $cloudletSLAList.size() \geq 1$ then
5     $cloudletVmMap$ =SLA-SpillScheduler($vmList$,$cloudletSLAList$,$cloudletVmMap$)
6 return $cloudletVmMap$
```

scheduler, and then computes the totalLength of all the SLA-based Cloudlets in the submitted workload (lines 5—6 of Algorithm 3.6). Afterwards, the computing share (vShareMapv) of each VM is determined (lines 7—8 of Algorithm 3.6) by using Equation 3.5. A while-loop (lines 9—22 of Algorithm 3.6) is used to schedule the SLA Cloudlets on VMs based on computation share and the execution cost of the VMs in the computing system. On each scheduling decision, first

Algorithm 3.6: SLA-FillScheduler

```
Input: $vmCrMap$ - set of VMs with computation ratio ,
$cloudletSLAList$ - list of cloudlets $c_1, c_2, c_3, \ldots, c_n$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping
1  $totalLength = 0$
2  $newVShare = 0$
3  $cloudletVmMap = Null$
4  $vShareMap < v, share >= Null$
5  forall $cloudlet$ in $cloudletSLAList$ do
6      $totalLenght$ = totalLenght + cloudlet.getCloudletLength()
7  forall $v$ in $vmList$ do
8      $vShareMap_v$ = totalLenght * $vmCrMap_v$
9  while $cloudletSLAList.size() \geq 1$ do
10     $v = getLargeShareVm$(vShareMap)
11     $cloudlet = getMaxPCloudletVm$(v,cloudletList)
12     if $cloudlet$ != $Null$ then
13         if $cloudlet$ is $SLA$-3 then
14             $v$ = getVMWithMinEGain($cloudlet$,$vmList$)
15         else if $cloudlet$ is $SLA$-2 then
16             $v$ = getSLA2SuitableVM($cloudlet$,$vmList$)
17         $cloudletVmMap.add$(cloudlet,v)
18         $newVShare = vShareMap.get$(v)$-$ cloudlet.$getCloudletLength$
19         $vShareMap.modify$(v,newVShare)
20         $cloudletList.remove$(cloudlet)
21     else
22         break while-loop
23 return $cloudletVmMap$
```

the VM with largest computing share (using Equation 3.12) is determined and the maxPCloudletVm is selected (lines 10—11 of Algorithm 3.6) using Equation 3.7. If the maxPCloudletVm is of SLA-3 type, then the VM selection is revised and the VmWithMinEGain is determined for maxPCloudlet to be mapped on it (lines 13—14 and line 17 of Algorithm 3.6). If the maxPCloudletVm is of SLA-2 type, then the VM selection is revised and the SLA2SuitableVM is determined for maxPCloudlet to be mapped on it (lines 15—16 and line 17 of Algorithm 3.6). If the maxPCloudletVm is of SLA-1 type, the Cloudlet is mapped to the VM with the largest computation share (lines 17 of Algorithm 3.6). On each scheduling step, when the Cloudlet is assigned to a specific VM, then its computing share

Algorithm 3.7: SLA-SpillScheduler

```
Input: $cloudletSLAList$ - list of cloudlets $c_1, c_2, c_3, \dots, c_n$,
$vmList$ - list of VMs $v_1, v_2, v_3, \dots, v_m$,
$cloudletVmMap$ - set of cloudlets to VMs mapping by $SLA - FillScheduler$
Output: $cloudletVmMap$ - set of cloudlets to VMs mapping
1  $vm = Null$
2  $cloudlet = Null$
3  while $cloudletSLAList.size() \geq 1$ do
4      $cloudlet = getMaxCloudlet$(cloudletSLAList)
5      if $cloudlet$ is $SLA$-3 then
6          $vm$ = getMostEconomicVm($cloudlet$,$vmList$)
7      else if $cloudlet$ is $SLA$-2 then
8          $vm$ = getSuitableVM($cloudlet$,$vmList$)
9      else if $cloudlet$ is $SLA$-1 then
10         $vm$ = getVMWithEFT($cloudlet$,$vmList$)
11     $cloudletVmMap.add$(cloudlet,vm)
12     $cloudletSLAList.remove$(cloudlet)
13 return $cloudletVmMap$
```

is modified (lines 18—19 of Algorithm 3.6) using Equation 3.11 and the Cloudlet is detached from the list of Cloudlets to be scheduled (line 20 of Algorithm 3.6). This scheduling process is repeated until no VM with a remaining computation share is found to accept any Cloudlet for scheduling (in line 11 of Algorithm 3.6, when the maxPCloudletVm is not found and Cloudlet is initialized with Null).

The SLA-Spill scheduler is engaged for the allocation of remaining Cloudlets to VMs, immediately after SLA-Fill scheduler finishes its working. The while-loop (line 3—12 of Algorithm 3.7) assigns the remaining Cloudlets to VMs in descending order of Cloudlets sizes. On each scheduling decision, the maxCloudlet is determined (line 4 of Algorithm 3.7) and scheduled on VMs. If the maxCloudlet is of SLA-3, then the maxCloudlet is scheduled on most economic VM (lines 5—6 and lines 11—12 of Algorithm 3.7). If the maxCloudlet is of SLA-2, then the maxCloudlet is scheduled on the suitable VM (lines 7—8 of Algorithm 3.7, provides the tradeoff between execution time and execution cost). If the maxCloudlet is of SLA-1, then it is allocated to the VM that produces earliest finish time for it (lines 9—12 of Algorithm 3.7). On each scheduling step, the maxCloudlet is scheduled on a candidate VM and it is detached from Cloudlets to be scheduled. This scheduling process is repeated until cloudletSLAList is empty.

3.3.6 Complexity of SLA-RALBA Algorithm

Let M be total number of machines in computing setup, N be total number of SLA-based jobs submitted in Cloud, and n be the number of cloudlets scheduled by SLA-Fill scheduler of SLA-RALBA. The time complexities of SLA-Fill Scheduler (i.e., Algorithm 3.6) and SLA-Spill scheduler (i.e., Algorithm 3.7) are O$(M^2.n)$ and O$(M.N–n)$, respectively. Therefore, the overall time complexity of SLA-RALBA is O$(M^2.n)$ [96], where $M << N$ on real Cloud.

Chapter 4

Experiments

This chapter presents the experimental environment and the workloads employed for the experimentation of proposed RALBA and SLA-RALBA techniques. The simulation setup and the employed two benchmark datasets (i.e., HCSP and GoCJ) used for the VM-level empirical investigation of RALBA against eight existing state-of-the-art scheduling algorithms are also presented.

4.1 Simulation Environment

This section sheds a light on the simulation setup and delineates two benchmark workloads employed in this study. The extensive evaluation of scheduling techniques and resource allocation policies on real Cloud (with a varying heterogeneous, load and system size) is a challenging problem. The use of real testbeds restrict the experiments to the scale of the test Cloud environment. The Cloud computing model is based on a pay-per-use model, thus repeatable experiments on a real Cloud may incur a high monetary cost. Therefore, an ideal alternative to evaluate resource management related Cloud policies is to use a simulation environment that enables Cloud developers to conduct experiments by employing the desired and varying configurations related to computing infrastructure and dataset (i.e., Cloud jobs). In this work, the extensive investigation of scheduling

Table 4.1: Configuration of Simulation Setup for RALBA Evaluation.

Parameters	**Details**
Simulator/version	CloudSim version 3.0.2.
Host machines power	04 Dual-core (4000 MIPS), and 26 Quad-core (4000 MIPS)
No. of host machines	30
Host machine memory	16384 MBs each
No. of VMs	50 Heterogeneous VMs (as shown in Table 4.2)
No. of Cloudlets	100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1000

techniques using different instances of benchmark datasets with a varying heterogeneity of computing environment is performed using a renowned Cloud simulator CloudSim (version 3.0.2) [1].

4.1.1 Experimental Setup for RALBA Evaluation

The performance analysis of RALBA technique is conducted on a machine equipped with Intel Core i3-4030U Quad-core processor (having 1.9 GHz clock speed) and 04 GBs of main memory. Table 4.1 illustrates the configuration detail for the employed simulation environment. All experiments are performed by using 50 VMs, hosted on 30 host machines within a datacenter. Table 4.2 presents the overall statistics of the VMs and their computing power in terms of Millions of Instructions Per Second (MIPS). As shown in Table 4.2, the slowest and fastest VMs have the computing power of 100 and 4000 MIPS, respectively.

4.1.2 Experimental Setup for VM-level Load Balancing

The experiments are performed on a computing machine equipped with Intel Core i3-4030U Quad-core processor (having 1.9 GHz clock speed) and 4 GBs of main memory. Table 4.3 presents the configurations for the employed simulation setup. The experiments are conducted with 16 heterogeneous VMs, hosted on 30 host machines within 1 data-center. The computing powers of VMs are assigned in

TABLE 4.2: Heterogeneous VMs used for RALBA Evaluation.

Powers in MIPS	**Number of VMs**
100	07
500	07
750	06
1000	06
1250	06
1500	06
1750	06
4000	06

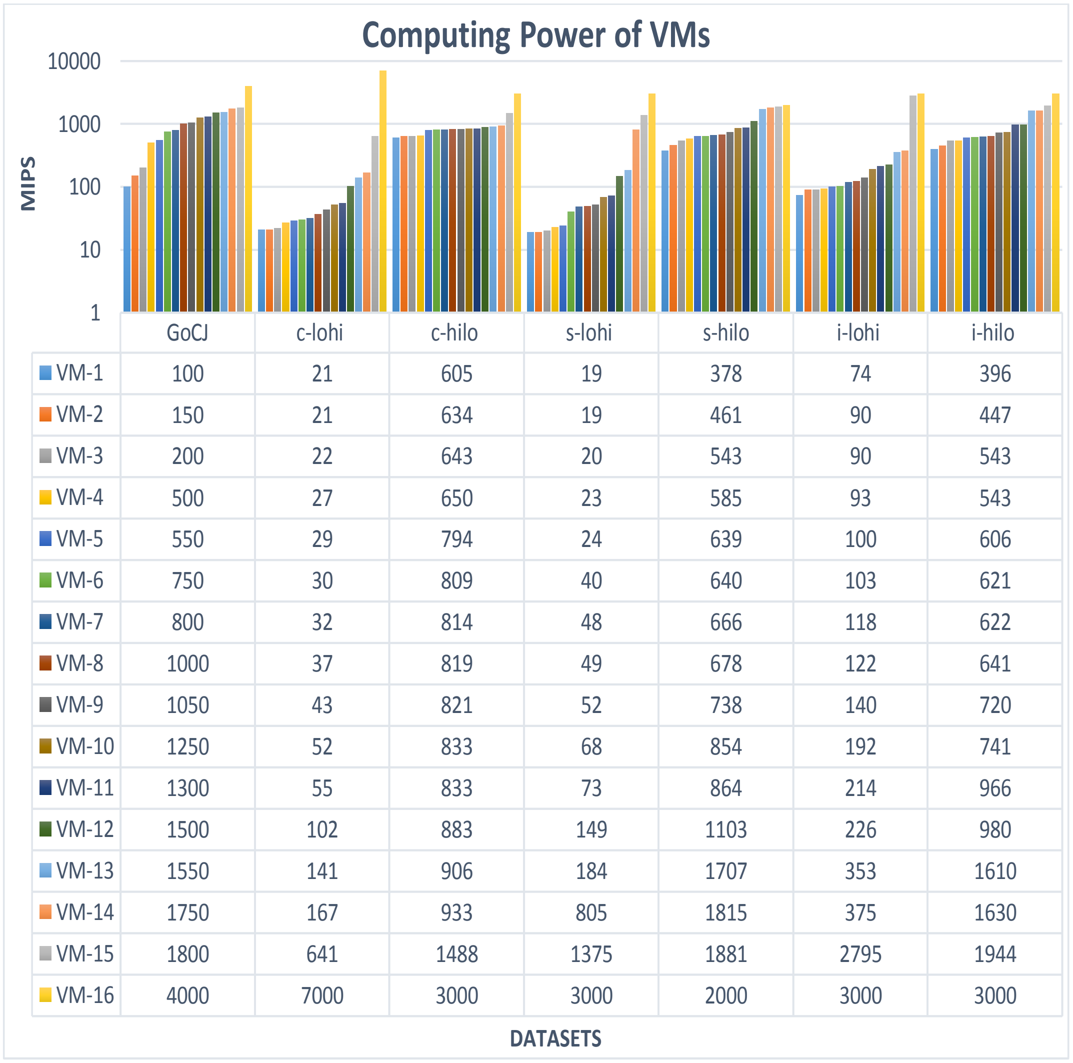
TABLE 4.3: Simulation Setup for VM-Level Load Balancing.

Parameters	**Description**
Simulation/version	CloudSim version 3.0.2
Host machines	04 Dual-core and 26 Quad-core
Power of each core	4000 MIPS
Total host machines	30 Host machines.
Host machines memory	16,384 MBs each
Total virtual machines	16 Heterogeneous VMs (as shown Fig. 4.1)

ascending order of their VMID. Fig. 4.1 shows the computing powers of VMs (in terms of MIPS) used for VM-level load balancing of RALBA.

4.1.3 Experimental Setup for SLA-RALBA Evaluation

The performance analysis of the proposed SLA-RALBA is accomplished on a machine equipped with Intel Core i3-4030U Quad-core 1.9 GHz processor and 04GBs main memory. The configuration of the simulation environment is presented in Table 4.3. It is comprised of host machines in the Cloud datacenter which virtualized the computing resources in the form of VMs. All the experiments are conducted with 30 VMs hosted on 30 host machines within a datacenter. Table 4.4 shows the information of VMs (i.e., the computing power in terms of MIPS and execution



	GoCJ	c-lohi	c-hilo	s-lohi	s-hilo	i-lohi	i-hilo
🟦 VM-1	100	21	605	19	378	74	396
🟧 VM-2	150	21	634	19	461	90	447
⬜ VM-3	200	22	643	20	543	90	543
🟨 VM-4	500	27	650	23	585	93	543
🟦 VM-5	550	29	794	24	639	100	606
🟩 VM-6	750	30	809	40	640	103	621
🟦 VM-7	800	32	814	48	666	118	622
🟫 VM-8	1000	37	819	49	678	122	641
⬛ VM-9	1050	43	821	52	738	140	720
🟫 VM-10	1250	52	833	68	854	192	741
🟦 VM-11	1300	55	833	73	864	214	966
🟩 VM-12	1500	102	883	149	1103	226	980
🟦 VM-13	1550	141	906	184	1707	353	1610
🟧 VM-14	1750	167	933	805	1815	375	1630
⬜ VM-15	1800	641	1488	1375	1881	2795	1944
🟨 VM-16	4000	7000	3000	3000	2000	3000	3000

FIGURE 4.1: Heterogeneous VMs in Computing Setup for VM-level Load Balancing.

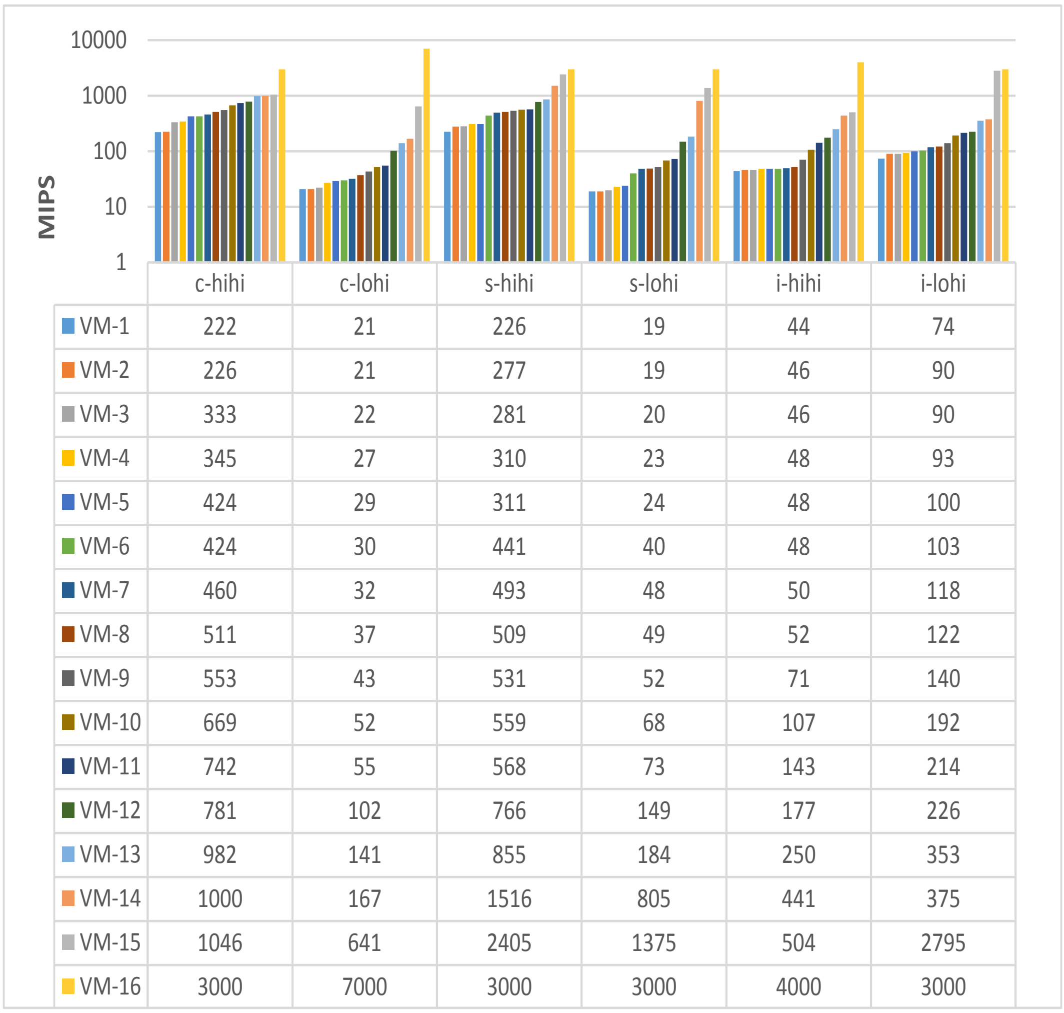
cost in terms of unit cost per second) for the experimentation of SLA-RALBA using GoCJ dataset [62, 94]. The computing powers and expected gain per second of VMs in computing setup for experimentation of SLA-RALBA evaluation with HCSP instances are presented in Fig. 4.2 and Fig. 4.3, respectively.

4.2 Employed Datasets

For the evaluation of RALBA, SLA-RALBA, and VM-level load balancing of RALBA, one synthetic [50] and two benchmark datasets [24, 62, 94, 95] are used.

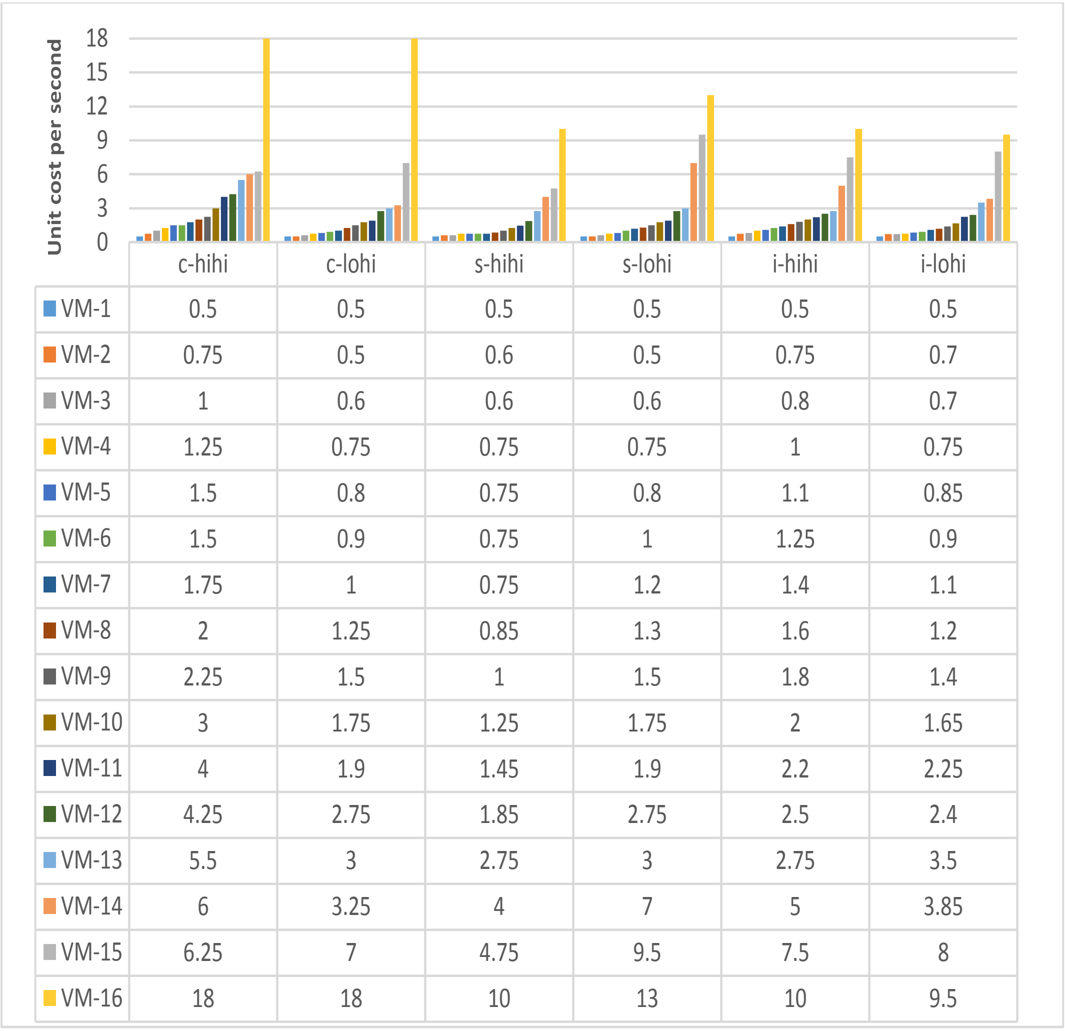
TABLE 4.4: VMs Composition for SLA-RALBA Evaluation using GoCJ Dataset.

Powers in MIPS	**Number of VMs**	**Unit Cost/Second**
100	04	0.5
500	04	2.0
750	04	3.0
1000	04	5.0
1250	04	6.5
1500	04	8.0
1750	03	9.0
3000	03	12.0



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟦 VM-1	222	21	226	19	44	74
🟧 VM-2	226	21	277	19	46	90
⬜ VM-3	333	22	281	20	46	90
🟨 VM-4	345	27	310	23	48	93
🟦 VM-5	424	29	311	24	48	100
🟩 VM-6	424	30	441	40	48	103
🟦 VM-7	460	32	493	48	50	118
🟫 VM-8	511	37	509	49	52	122
⬛ VM-9	553	43	531	52	71	140
🟫 VM-10	669	52	559	68	107	192
🟦 VM-11	742	55	568	73	143	214
🟩 VM-12	781	102	766	149	177	226
🟦 VM-13	982	141	855	184	250	353
🟧 VM-14	1000	167	1516	805	441	375
⬜ VM-15	1046	641	2405	1375	504	2795
🟨 VM-16	3000	7000	3000	3000	4000	3000

FIGURE 4.2: Powers of 16 VMs in Computing Setup for SLA-RALBA Evaluation using HCSP Instances.



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟦 VM-1	0.5	0.5	0.5	0.5	0.5	0.5
🟧 VM-2	0.75	0.5	0.6	0.5	0.75	0.7
⬜ VM-3	1	0.6	0.6	0.6	0.8	0.7
🟨 VM-4	1.25	0.75	0.75	0.75	1	0.75
🟦 VM-5	1.5	0.8	0.75	0.8	1.1	0.85
🟩 VM-6	1.5	0.9	0.75	1	1.25	0.9
🟦 VM-7	1.75	1	0.75	1.2	1.4	1.1
🟫 VM-8	2	1.25	0.85	1.3	1.6	1.2
⬛ VM-9	2.25	1.5	1	1.5	1.8	1.4
🟫 VM-10	3	1.75	1.25	1.75	2	1.65
🟦 VM-11	4	1.9	1.45	1.9	2.2	2.25
🟩 VM-12	4.25	2.75	1.85	2.75	2.5	2.4
🟦 VM-13	5.5	3	2.75	3	2.75	3.5
🟧 VM-14	6	3.25	4	7	5	3.85
⬜ VM-15	6.25	7	4.75	9.5	7.5	8
🟨 VM-16	18	18	10	13	10	9.5

FIGURE 4.3: Expected Gain of 16 VMs in Computing Setup for SLA-RALBA Evaluation using HSCP Instances.

4.2.1 Synthetic Dataset

A synthetic dataset of independent Cloudlets are generated using the guidelines available in the literature [88, 90, 107, 113]. The synthetic workload is created using random-number generation mechanism employing 05 different Cloudlet-size ranges i.e., tiny (1–250 MI), small (800–1200 MI), medium (1800–2500 MI), large (7000–10,000 MI), and extra-large (30,000–45,000 MI) (as shown in Table 4.5). This synthetic dataset (with different number of Cloudlets as shown in Table 4.1) is employed in the experimentation of RALBA, Sufferage, TASA, RASA, Max-Min, Min-Min, MCT, RR, and RS algorithms.

Table 4.5: Composition of Synthetic Dataset used for RALBA Evaluation.

Cloudlet Size	%age Cloudlets
Tiny (1-250 MI)	20%
Small (800-1200 MI)	60%
Medium (1800-2500 MI)	05%
Large (7000-10000 MI)	10%
Extra Large (30000-45000 MI)	05%

4.2.2 GoCJ Dataset [62, 94]

The first benchmark dataset used in the experiments of RALBA, SLA-RALBA, and VM-level load balancing of RALBA is the proposed GoCJ dataset [62] that is elaborated in detail in Section 3.1.1 of Chapter 3. The composition of GoCJ dataset is presented in Fig. 3.1.

The GoCJ dataset (with different number of Cloudlets as shown in Table 4.1) is employed in the experimentation of RALBA, Sufferage, TASA, RASA, Max-Min, Min-Min, MCT, RR, and RS algorithms. Three different instances of GoCJ dataset (i.e., with 256, 512, and 1024 Cloudlets, respectively) are employed in the experimentation of VM-level load balancing of RALBA against Sufferage, TASA, RASA, Max-Min, MCT, Min-Min, RASA, OLB, and PSSELB algorithms. In addition, 09 GoCJ instances are used for experimentation of SLA-RALBA, SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min algorithms (each instance comprised of 200, 300, 400, 500, 600, 700, 800, 900 and 1000 Cloudlets, respectively).

4.2.3 HCSP Instances [24, 95]

The second benchmark dataset used in the experiments of SLA-RALBA and VM-level load balancing is the ETC model of Braun et al. in the form of HCSP instances [24, 95]. HCSP instances are labeled using a name with pattern C-THMH, where c indicates the consistency type (c for consistent, s for semi-consistent, and i for inconsistent), TH and MH indicate the task and machine level heterogeneity, respectively (lo and hi for low and high heterogeneity, respectively).

The 06 HCSP instances used for VM-level load balancing of RALBA are c-hilo, clohi, i-hilo, i-lohi, s-hilo, and s-lohi instances. The low and high task heterogeneity with high machine heterogeneity are employed for the empirical investigation. Therefore, 06 HCSP instances are employed in this pragmatic examination of SLA-RALBA evaluation are c-hihi, c-lohi, i-hihi, i-lohi, s-hihi, and s-lohi instances.

Chapter 5

Results and Discussion

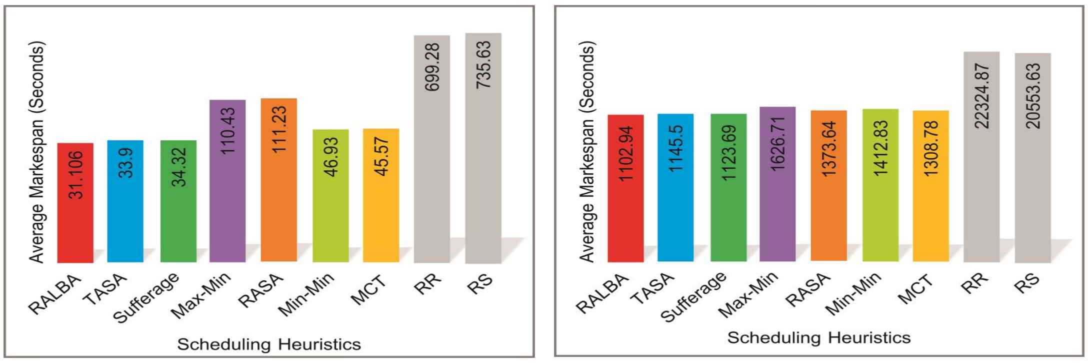
This chapter presents the discussions on the experimental results of RALBA, SLA-RALBA, and VM-level investigation of RALBA. The proposed RALBA and SLA-RALBA techniques are empirically examined using two benchmark datasets.

5.1 Empirical Results - RALBA Scheduler

The performance of RALBA and the other Cloud scheduling heuristics (i.e., TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RR, and RS) is compared by using makespan, ARUR, and throughput metrics. Each experiment is performed 05 times and the analysis is conducted on average values.

5.1.1 Average Makespan Results

Fig. 5.1-a presents the average makespan based results for the synthetic workload. Fig. 5.1-a shows that RALBA consumes on average 5.5, 6.9, 243.9, 246.4, 46.1, and 41.9% less time for the execution of synthetic workload as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, and MCT heuristics, respectively. For the execution of GoCJ workload (shown in Fig. 5.1-b), RALBA consumes on



(a) Average makespan – Synthetic Workload **(b) Average makespan – GoCJ Workload**

FIGURE 5.1: Average Makespan Results.

average 3.8, 1.8, 47.5, 24.5, 28.1, and 18.6% lower time as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, and MCT heuristics, respectively.

5.1.2 Mean ARUR Results

Fig. 5.2-a presents the mean ARUR based experimental results for execution of the synthetic workload.

Fig. 5.2-a shows that RALBA has attained 10.9, 9.8, 103.3, 117.6, 121.6, 65, 624.1, and 541.2% higher resource utilization (using synthetic workload) as compared to TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR heuristics, respectively. For the execution of GoCJ workload, RALBA has achieved higher resource utilization of 16.9, 7.4, 15.9, 13.9, 78.8, 30.1, 550.6, and 541.7% as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR scheduling heuristics, respectively (see Fig. 5.2-b).

5.1.3 Average Throughput Results

Fig. 5.3-a presents average throughput results for execution of the synthetic workload. As shown in Fig. 5.3-a, RALBA has achieved 6.2, 7.5, 151.6, 160.1, 47.2, 42.1, 2047, and 1812% higher throughput (using synthetic workload) as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR scheduling heuristics, respectively. Fig. 5.3-b shows the average throughput results for

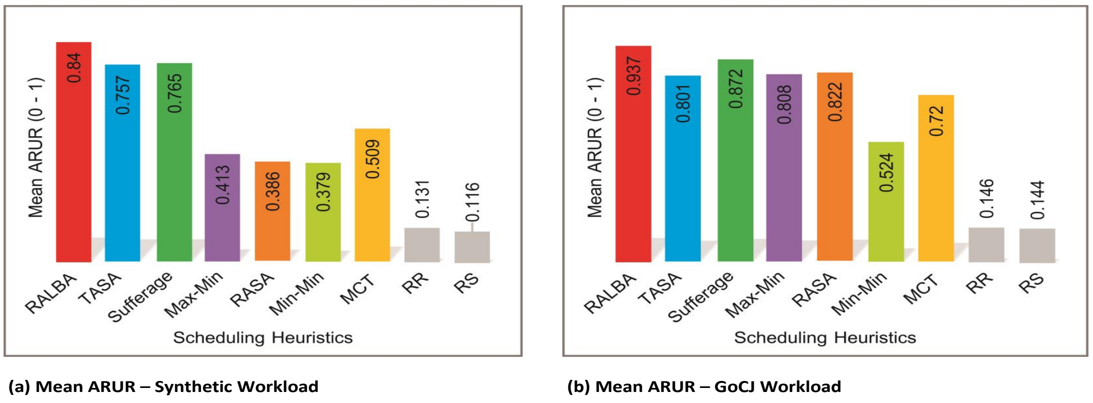


Figure 5.2: Mean ARUR Results.

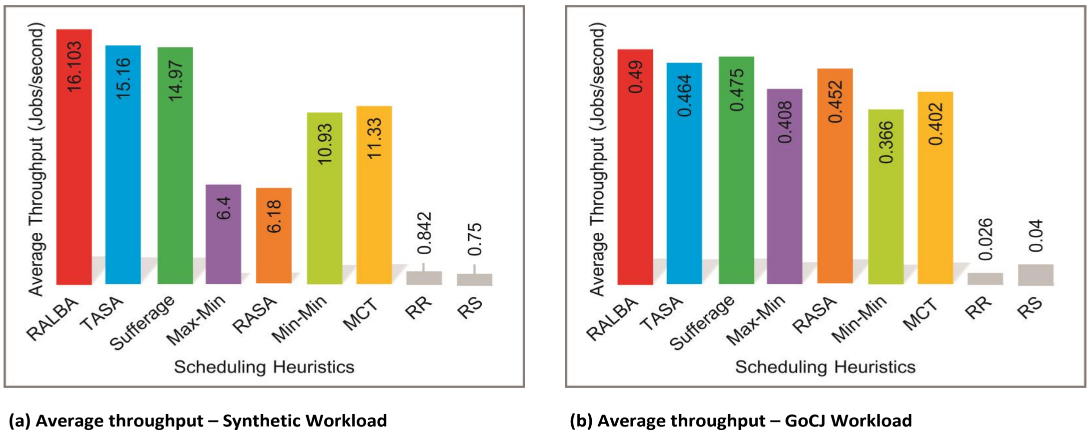


Figure 5.3: Average Throughput Results.

GoCJ workload. As shown in Fig. 5.3-b, RALBA has attained higher job execution throughput of 5.6, 3.1, 20.1, 8.4, 33.8, 21.9, 1125, and 1785% as compared to the TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR heuristics, respectively.

5.1.4 Results Discussion

It is scrutinized that the proposed scheduling technique has successively achieved significant improvements in terms of makespan and resource utilization for the skewed workload [28] (especially the positively skewed one) because of the inherent resource aware scheduling mechanism employed by RALBA. We refer workload having a large number of shorter Cloudlets (positively skewed) and a large number of longer size Cloudlets (negatively skewed). A modest improvement in makespan and significant higher resource utilization is observed for the execution of the non-skewed workload. The higher resource utilization achieved by RALBA is due to the resource aware mapping that results in a load balanced execution of the workload. The composition of the synthetic workload (as shown in Fig. 2.4) shows that there are a large number of smaller and a few outsized (i.e., 10% large and 05% extra-large) Cloudlets. This workload composition shows that the synthetic workload is more positively skewed as compared to the GoCJ workload (i.e., 05% more number of large Cloudlets). Experiments have revealed that the MCT scheduling heuristic reduces the makespan; however, it causes an imbalanced mapping of Cloudlets by overloading the faster VMs. The execution of the synthetic workload has resulted in more imbalanced mapping as compared to the scheduling of GoCJ workload which arose due to the more positively skewed nature of the synthetic workload as compared to the GoCJ workload.

The experimental evaluation has revealed that the Min–Min has acquired reduced makespan as compared to the Max–Min due to a large number of shorter size Cloudlets in both workloads (i.e., synthetic and GoCJ). The inherent scheduling mechanism of Max–Min maps some larger Cloudlets to slow VMs producing a longer makespan [31]. As compared to Min–Min heuristic, RALBA has achieved 28.1% (for GoCJ workload) and 46.1% (for synthetic workload) reduced makespan. Moreover, RALBA has consumed 243.9 and 47.5% reduced time as compared to Max–Min for the execution of the GoCJ and synthetic workloads, respectively. Max–Min has produced longer makespan (for GoCJ workload) due to a few huge size Cloudlets as compared to the synthetic workload. Alternatively, Max–Min has significantly improved resource utilization as compared to the Min–Min that overloads faster VMs (producing load imbalance). However, Max–Min has not significantly improved the resource utilization (for the execution of the synthetic workload) as compared to the GoCJ workload because the synthetic workload contains high number of large size Cloudlets. Our experimental evaluation exhibits that the Sufferage scheduling heuristic produces lower makespan and higher resource utilization as compared to the Min–Min, Max–Min, and RASA heuristics. This reduced makespan and higher resource utilization are achieved due to the selection of a suitable VM for Cloudlet mapping. The suitable VM represents the computing resource which will suffer most (in terms of makespan) if the candidate Cloudlet has not been mapped to that resource (in the current scheduling iteration).

The VM–Cloudlet allocation mechanism of TASA is based on Sufferage and Min-Min algorithms. Therefore, TASA provides reduced makespan among the existing algorithms. Fig. 5.1 shows that RALBA has produced slightly reduced makespan as compared to TASA and Sufferage for the GoCJ workload (i.e., 1.88 and 3.86% reduced, respectively). TASA and Sufferage have performed almost identical to RALBA due to the inherent resource-aware mapping mechanism employed by both of these scheduling heuristics. However, RALBA also considers load-balance factor for scheduling; therefore, it has produced higher resource utilization (i.e., 7.45 and 16.97% more resource utilization as compared to TASA and Sufferage, respectively) for GoCJ workload as well. It is evident that the TASA has produced lower resource utilization (as compared to the Sufferage) due to the inherent use of Min–Min scheduling mechanism.

The scheduling mechanism of RASA is based on Max–Min and Min–Min heuristics. Therefore, RASA provides improved resource utilization over TASA for the GoCJ workload (better resource utilization due to the inherent Max–Min mechanism). Overall scrutinization of the conducted experiments persuades that RALBA has achieved significant improvement in resource utilization, makespan, and throughput as compared to the existing scheduling heuristics. However, RALBA does not support SLA-aware scheduling of Cloud jobs. Therefore, the SLA based resource (e.g., execution-cost, bandwidth, memory, etc.) and deadline constrained Cloudlets may not be scheduled adequately.

5.2 RALBA: VM-Level Load Balancing

This section elaborates the experimental evaluation of 09 scheduling algorithms, including RALBA, using two benchmark datasets: GoCJ [62] and HCSP datasets

[95]. The workload distribution by each algorithm is measured and compared with the computation share [50] of each VM in the simulation environment as a machine-level load-imbalance.

5.2.1 Investigation using GoCJ Dataset

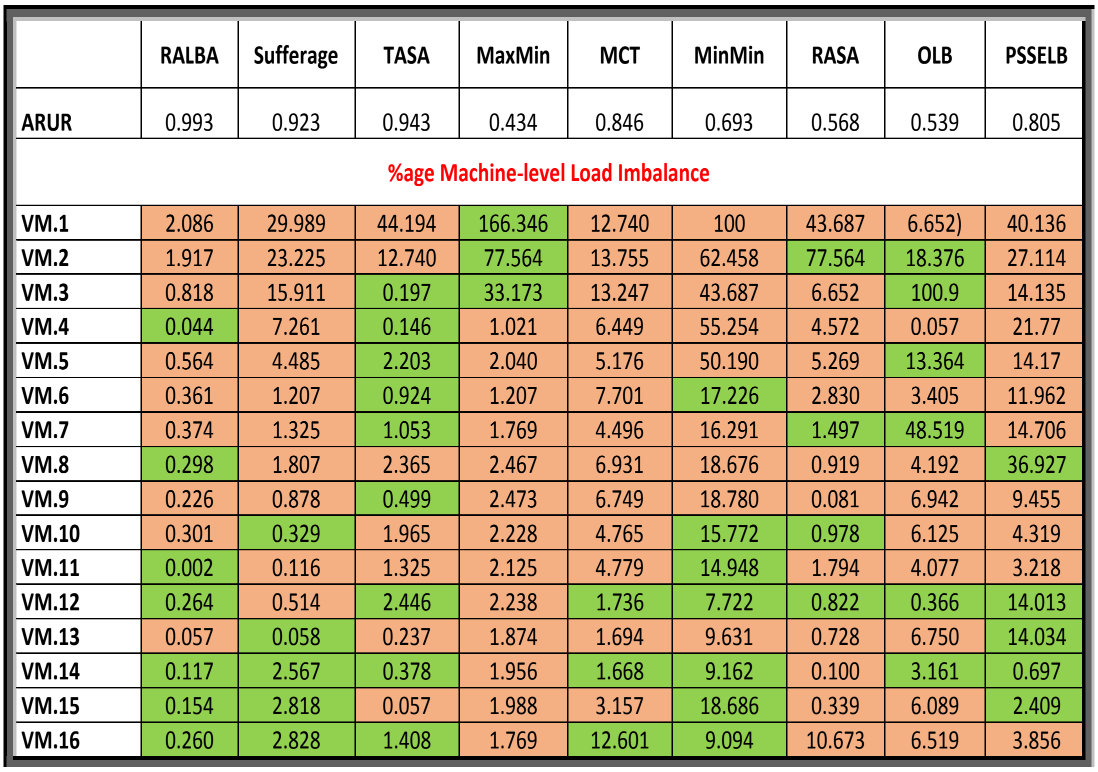
The load imbalance produced by the scheduling algorithms using different instances of GoCJ dataset is described below:

Load-Imbalance using 256x16 - GOCJ Instances

Fig. 5.4 shows the VMs level load-imbalance that is produced by scheduling heuristics using 16 VMs and 256 Cloudlets from GoCJ dataset. The load-imbalance is presented in terms of percentage load-imbalance at VM level. The numbers in the pink-background present the percentage under-utilization of virtual machines and the numbers in green-background present the percentage over-utilization of virtual machines as compared to the computation share of each VM.

The RALBA achieves highest resource utilization (i.e., 99.3%) among the given scheduling techniques by providing the machine-level load balancing as presented in Fig. 5.4. On the other hand, Max-Min produces the least resource utilization among scheduling techniques. Min-Min heuristic produces a sever VM-level load-imbalance by overloading the faster VMs and under-utilizing the slower VMs (i.e., the VM1 remains idle), although Min-Min produces 69.3% resource utilization. Likewise, the PSSELB under-utilizes the slower machines and overloads the machines with moderate computing powers. The TASA and sufferage achieve 94.3 and 92.3% resource utilization, respectively. On the other hand, both of these heuristics under-utilize the slower machines and provide comparatively moderate load-balancing at faster machines. It is observed that PSSELB produces higher resource utilization than Min-Min technique, but PSSELB produces more load-imbalance at each machine level as shown in Fig. 5.4.

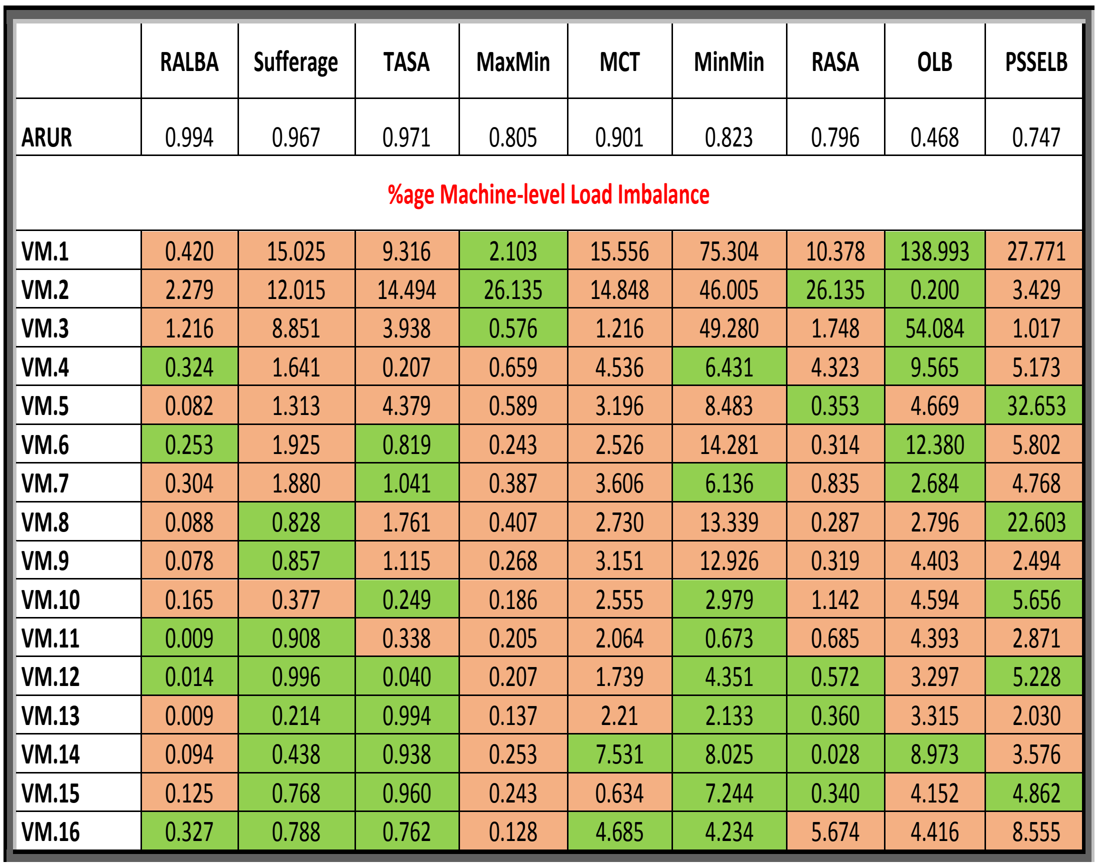
Load-Imbalance using 512x16 using GOCJ Instances



	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.993	0.923	0.943	0.434	0.846	0.693	0.568	0.539	0.805
					%age Machine-level Load Imbalance				
VM.1	2.086	29.989	44.194	166.346	12.740	100	43.687	6.652)	40.136
VM.2	1.917	23.225	12.740	77.564	13.755	62.458	77.564	18.376	27.114
VM.3	0.818	15.911	0.197	33.173	13.247	43.687	6.652	100.9	14.135
VM.4	0.044	7.261	0.146	1.021	6.449	55.254	4.572	0.057	21.77
VM.5	0.564	4.485	2.203	2.040	5.176	50.190	5.269	13.364	14.17
VM.6	0.361	1.207	0.924	1.207	7.701	17.226	2.830	3.405	11.962
VM.7	0.374	1.325	1.053	1.769	4.496	16.291	1.497	48.519	14.706
VM.8	0.298	1.807	2.365	2.467	6.931	18.676	0.919	4.192	36.927
VM.9	0.226	0.878	0.499	2.473	6.749	18.780	0.081	6.942	9.455
VM.10	0.301	0.329	1.965	2.228	4.765	15.772	0.978	6.125	4.319
VM.11	0.002	0.116	1.325	2.125	4.779	14.948	1.794	4.077	3.218
VM.12	0.264	0.514	2.446	2.238	1.736	7.722	0.822	0.366	14.013
VM.13	0.057	0.058	0.237	1.874	1.694	9.631	0.728	6.750	14.034
VM.14	0.117	2.567	0.378	1.956	1.668	9.162	0.100	3.161	0.697
VM.15	0.154	2.818	0.057	1.988	3.157	18.686	0.339	6.089	2.409
VM.16	0.260	2.828	1.408	1.769	12.601	9.094	10.673	6.519	3.856

FIGURE 5.4: Individual Machine-level Load Imbalance using 256 Cloudlets - GoCJ Dataset.

Fig. 5.5 shows the VMs level load-imbalance that is produced by scheduling heuristics using 16 VMs and 512 Cloudlets from GoCJ dataset. The highest resource utilization (i.e., 99.4%) is attained by RALBA, however, OLB produces the least resource utilization. RALBA scheduler produces a VM-level load balancing with less than 1% load-imbalance on each machine, while VM.1 and VM.2 are 2.279 and 1.216% under-utilized as compared to its computing share, respectively. TASA and sufferage heuristics achieve 97.1 and 96.7% resource utilization, respectively. But then again, TASA and sufferage under-utilize the slower machines with moderate load balancing at faster machines. The Max-Min overloads the slower machines, while the Min-Min under-utilizes the slower machines. On the other hand, Min-Min overloads the faster machines in the computing setup. The least resource utilization is produced by OLB (i.e., 46.8% resource utilization) by overloading the slower machines and underutilizing the faster machines. Likewise, the PSSELB heuristic also provides a sever load-imbalance at machine-level while producing 74.7% resource utilization. Likewise Min-Min, the MCT severely under-utilizes the slower machines that results in the over-utilization of faster machines as shown



	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.994	0.967	0.971	0.805	0.901	0.823	0.796	0.468	0.747
%age Machine-level Load Imbalance									
VM.1	0.420	15.025	9.316	2.103	15.556	75.304	10.378	138.993	27.771
VM.2	2.279	12.015	14.494	26.135	14.848	46.005	26.135	0.200	3.429
VM.3	1.216	8.851	3.938	0.576	1.216	49.280	1.748	54.084	1.017
VM.4	0.324	1.641	0.207	0.659	4.536	6.431	4.323	9.565	5.173
VM.5	0.082	1.313	4.379	0.589	3.196	8.483	0.353	4.669	32.653
VM.6	0.253	1.925	0.819	0.243	2.526	14.281	0.314	12.380	5.802
VM.7	0.304	1.880	1.041	0.387	3.606	6.136	0.835	2.684	4.768
VM.8	0.088	0.828	1.761	0.407	2.730	13.339	0.287	2.796	22.603
VM.9	0.078	0.857	1.115	0.268	3.151	12.926	0.319	4.403	2.494
VM.10	0.165	0.377	0.249	0.186	2.555	2.979	1.142	4.594	5.656
VM.11	0.009	0.908	0.338	0.205	2.064	0.673	0.685	4.393	2.871
VM.12	0.014	0.996	0.040	0.207	1.739	4.351	0.572	3.297	5.228
VM.13	0.009	0.214	0.994	0.137	2.21	2.133	0.360	3.315	2.030
VM.14	0.094	0.438	0.938	0.253	7.531	8.025	0.028	8.973	3.576
VM.15	0.125	0.768	0.960	0.243	0.634	7.244	0.340	4.152	4.862
VM.16	0.327	0.788	0.762	0.128	4.685	4.234	5.674	4.416	8.555

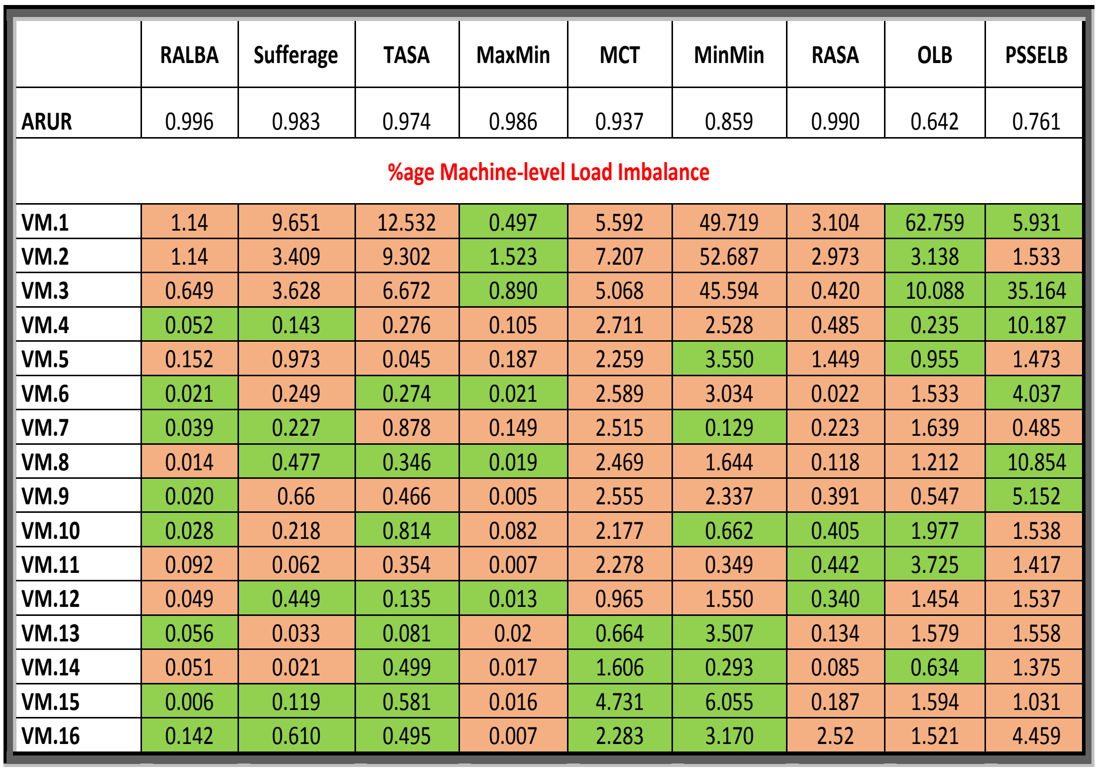
FIGURE 5.5: Individual Machine-level Load Imbalance using 512 Cloudlets - GoCJ Dataset.

in Fig. 5.5.

Load-Imbalance using 1024x16 - GOCJ Instances

Fig. 5.6 shows the VMs level load-imbalance that is produced by scheduling algorithms using 16 VMs and 1024 Cloudlets from GoCJ dataset.

RALBA heuristic achieves the highest resource utilization (i.e., 99.6%), while the second highest resource utilization (i.e., 98.6%) is produced by RASA. The RALBA provides almost a complete load balancing at VMs level as shown in Fig. 5.6. The scheduling mechanism of TASA and sufferage heuristics under-utilize the slower machines and provide a moderate load balancing at faster machines. Min-Min severely underutilized and the OLB overloads the slower machines. Likewise, the slower machines are under-utilized and the faster machines are overloaded by the MCT heuristic. The PSSELB severely overloads the slower VMs and the moderate speed VMs, while under-utilizing the faster VMs.



	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.996	0.983	0.974	0.986	0.937	0.859	0.990	0.642	0.761
					%age Machine-level Load Imbalance				
VM.1	1.14	9.651	12.532	0.497	5.592	49.719	3.104	62.759	5.931
VM.2	1.14	3.409	9.302	1.523	7.207	52.687	2.973	3.138	1.533
VM.3	0.649	3.628	6.672	0.890	5.068	45.594	0.420	10.088	35.164
VM.4	0.052	0.143	0.276	0.105	2.711	2.528	0.485	0.235	10.187
VM.5	0.152	0.973	0.045	0.187	2.259	3.550	1.449	0.955	1.473
VM.6	0.021	0.249	0.274	0.021	2.589	3.034	0.022	1.533	4.037
VM.7	0.039	0.227	0.878	0.149	2.515	0.129	0.223	1.639	0.485
VM.8	0.014	0.477	0.346	0.019	2.469	1.644	0.118	1.212	10.854
VM.9	0.020	0.66	0.466	0.005	2.555	2.337	0.391	0.547	5.152
VM.10	0.028	0.218	0.814	0.082	2.177	0.662	0.405	1.977	1.538
VM.11	0.092	0.062	0.354	0.007	2.278	0.349	0.442	3.725	1.417
VM.12	0.049	0.449	0.135	0.013	0.965	1.550	0.340	1.454	1.537
VM.13	0.056	0.033	0.081	0.02	0.664	3.507	0.134	1.579	1.558
VM.14	0.051	0.021	0.499	0.017	1.606	0.293	0.085	0.634	1.375
VM.15	0.006	0.119	0.581	0.016	4.731	6.055	0.187	1.594	1.031
VM.16	0.142	0.610	0.495	0.007	2.283	3.170	2.52	1.521	4.459

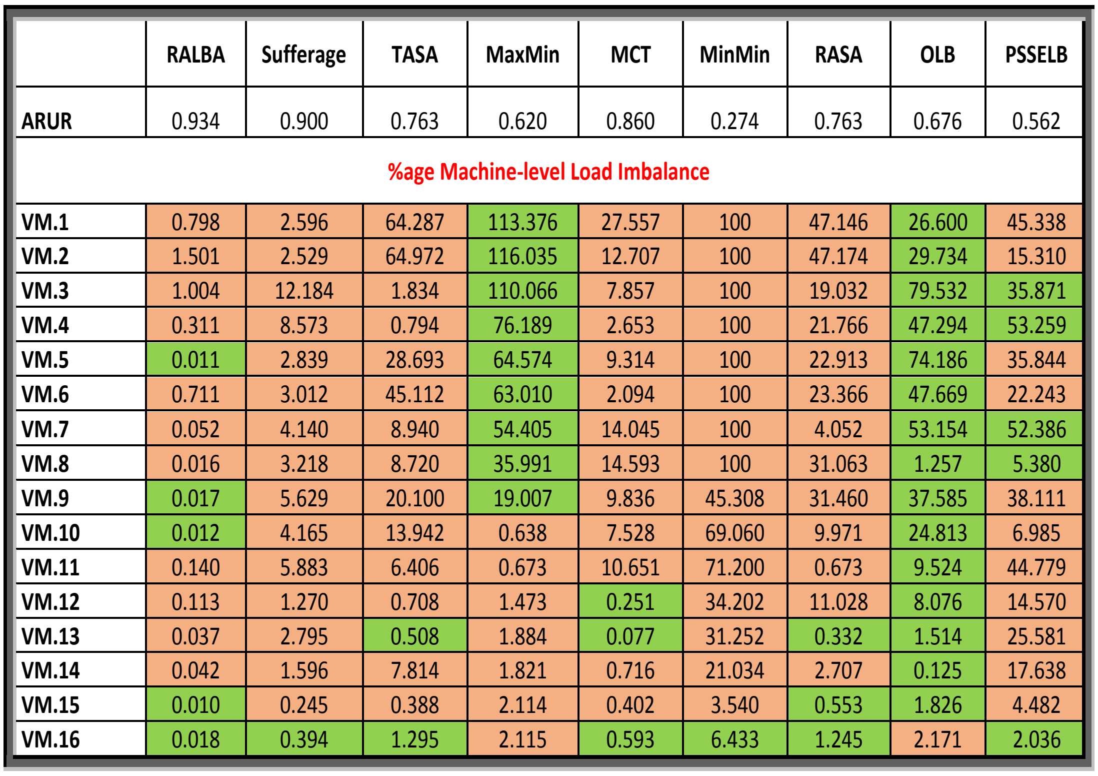
Figure 5.6: Individual Machine-level Load Imbalance using 1024 Cloudlets - GoCJ Dataset.

5.2.2 Investigation using HCSP Dataset

The load imbalance produced by the scheduling heuristics using different instances of HCSP dataset is described below:

Load-Imbalance using 512x16 c-lohi - HCSP Instances

Fig. 5.7 shows VMs level load-imbalance that is produced by scheduling heuristics using 512x16 c-lohi datasets. The RALBA scheduler achieves the highest resource utilization (i.e., 93.4%), while the Min-Min attains the least resource utilization (i.e., 27.4%) among the given scheduling algorithms. The Min-Min heuristic produces a sever load-imbalance by under-utilizing the VM.9–15, overloading the VM.16. In addition, Min-Min does not utilize the VM.1–8 during workload execution. Sufferage technique attains the second highest resource utilization (i.e., 90%), however, it underutilizes the slower machines. Likewise Min-Min, the Max-Min produces a sever load-imbalance by heavily overloading the slower machines. In the same way, OLB overloads the slower VMs and the VMs with moderate



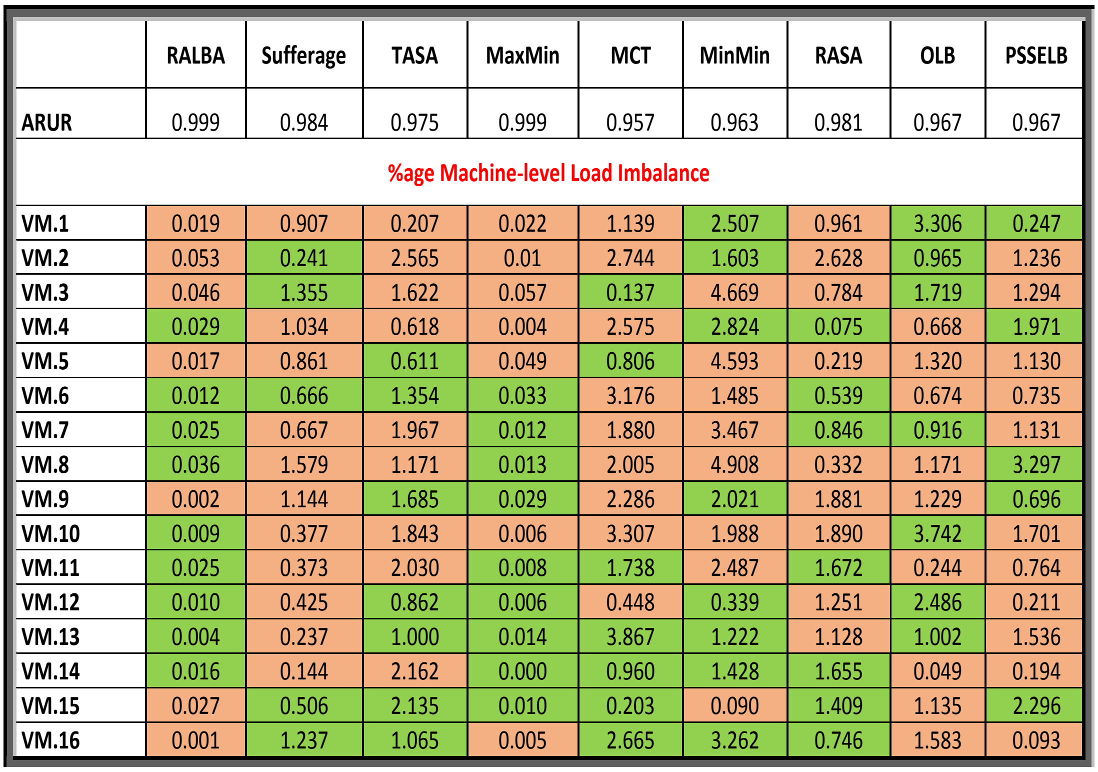
	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.934	0.900	0.763	0.620	0.860	0.274	0.763	0.676	0.562
%age Machine-level Load Imbalance									
VM.1	0.798	2.596	64.287	113.376	27.557	100	47.146	26.600	45.338
VM.2	1.501	2.529	64.972	116.035	12.707	100	47.174	29.734	15.310
VM.3	1.004	12.184	1.834	110.066	7.857	100	19.032	79.532	35.871
VM.4	0.311	8.573	0.794	76.189	2.653	100	21.766	47.294	53.259
VM.5	0.011	2.839	28.693	64.574	9.314	100	22.913	74.186	35.844
VM.6	0.711	3.012	45.112	63.010	2.094	100	23.366	47.669	22.243
VM.7	0.052	4.140	8.940	54.405	14.045	100	4.052	53.154	52.386
VM.8	0.016	3.218	8.720	35.991	14.593	100	31.063	1.257	5.380
VM.9	0.017	5.629	20.100	19.007	9.836	45.308	31.460	37.585	38.111
VM.10	0.012	4.165	13.942	0.638	7.528	69.060	9.971	24.813	6.985
VM.11	0.140	5.883	6.406	0.673	10.651	71.200	0.673	9.524	44.779
VM.12	0.113	1.270	0.708	1.473	0.251	34.202	11.028	8.076	14.570
VM.13	0.037	2.795	0.508	1.884	0.077	31.252	0.332	1.514	25.581
VM.14	0.042	1.596	7.814	1.821	0.716	21.034	2.707	0.125	17.638
VM.15	0.010	0.245	0.388	2.114	0.402	3.540	0.553	1.826	4.482
VM.16	0.018	0.394	1.295	2.115	0.593	6.433	1.245	2.171	2.036

FIGURE 5.7: Individual Machine-level Load Imbalance using 512x16 c-lohi - HCSP Instances.

computing power. Most of the slower VMs are assigned with the lesser workload as compared to its computing share by TASA and MCT heuristics. The PSSELB technique repeats its behavior of workload distribution by either heavily overloading or underutilizing the slower VMs and the VMs with moderate computing powers. On the other hand, RALBA produces a VM-level load balancing as presented in the results.

Load-Imbalance using 512x16 c-hilo - HCSP Instances

Fig. 5.8 shows VMs level load-imbalance that is produced by scheduling heuristics using 512 x 16 c-hilo dataset. The RALBA and Max-Min heuristics achieve the highest resource utilization (i.e., 99.9%), while the MCT attains the least resource utilization. The MCT slightly overloads the faster VMs and under-utilizes the slower machines. Min-Min heuristic produces the load-imbalance at the slower VMs, while slightly overloads the faster VMs. The PSSELB and OLB heuristics produce the machine-level load-imbalance by slightly overloading a few machines. Likewise, the TASA technique slightly under-utilizing the slower VMs and assigned



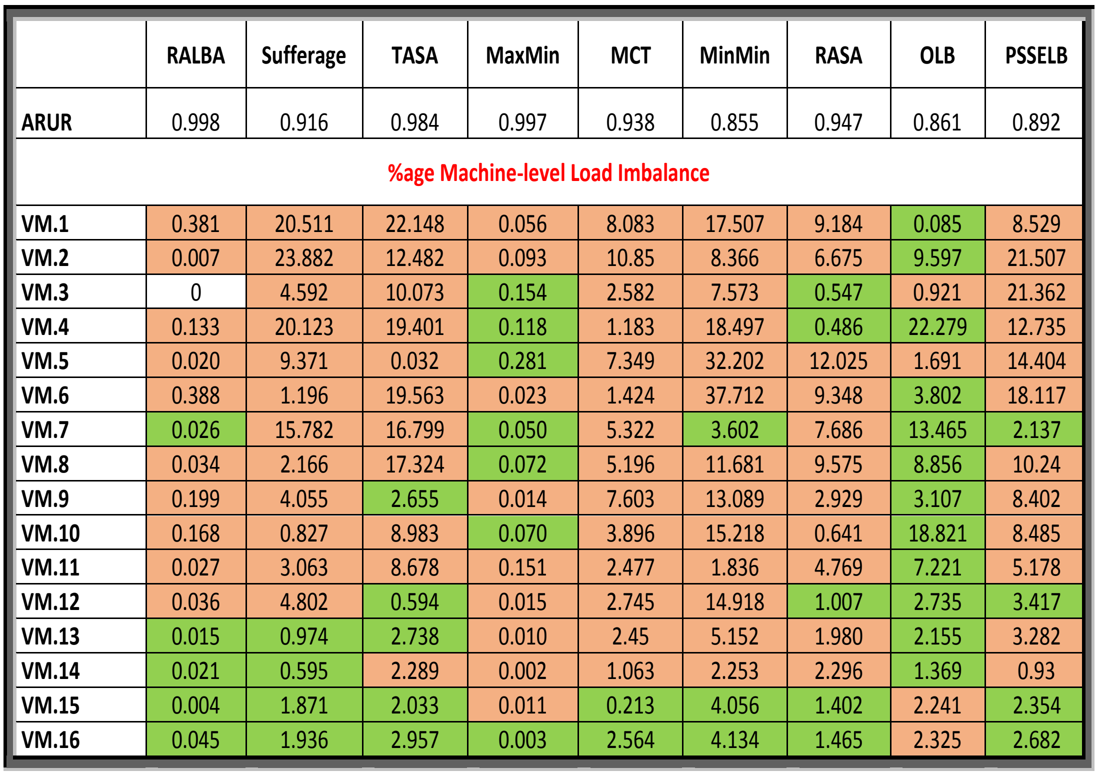
	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.999	0.984	0.975	0.999	0.957	0.963	0.981	0.967	0.967
%age Machine-level Load Imbalance									
VM.1	0.019	0.907	0.207	0.022	1.139	2.507	0.961	3.306	0.247
VM.2	0.053	0.241	2.565	0.01	2.744	1.603	2.628	0.965	1.236
VM.3	0.046	1.355	1.622	0.057	0.137	4.669	0.784	1.719	1.294
VM.4	0.029	1.034	0.618	0.004	2.575	2.824	0.075	0.668	1.971
VM.5	0.017	0.861	0.611	0.049	0.806	4.593	0.219	1.320	1.130
VM.6	0.012	0.666	1.354	0.033	3.176	1.485	0.539	0.674	0.735
VM.7	0.025	0.667	1.967	0.012	1.880	3.467	0.846	0.916	1.131
VM.8	0.036	1.579	1.171	0.013	2.005	4.908	0.332	1.171	3.297
VM.9	0.002	1.144	1.685	0.029	2.286	2.021	1.881	1.229	0.696
VM.10	0.009	0.377	1.843	0.006	3.307	1.988	1.890	3.742	1.701
VM.11	0.025	0.373	2.030	0.008	1.738	2.487	1.672	0.244	0.764
VM.12	0.010	0.425	0.862	0.006	0.448	0.339	1.251	2.486	0.211
VM.13	0.004	0.237	1.000	0.014	3.867	1.222	1.128	1.002	1.536
VM.14	0.016	0.144	2.162	0.000	0.960	1.428	1.655	0.049	0.194
VM.15	0.027	0.506	2.135	0.010	0.203	0.090	1.409	1.135	2.296
VM.16	0.001	1.237	1.065	0.005	2.665	3.262	0.746	1.583	0.093

FIGURE 5.8: Individual Machine-level Load Imbalance using 512x16 c-hilo - HCSP Instances.

the faster VMs with slightly more workload than its computing share. On the other hand, the Max-Min and RALBA produce almost a complete load balancing at VMs level.

Load-Imbalance using 512x16 i-lohi - HCSP Instances

Fig. 5.9 shows VMs level load-imbalance that is caused by scheduling heuristics using 512 x 16 i-lohi datasets. RALBA attains the highest resource utilization (i.e., 99.8%) followed by Max-Min that achieves the second highest resource utilization (i.e., 99.7%). RALBA and Max-Min schedulers produce almost a complete load balancing at the VM level. In addition, the VM.3 is fully utilized as per its computing share by RALBA. The Min-Min produces the least resource utilization and depicts the load-imbalance by heavily underutilizing the slower VMs and moderate speed VMs, while overloading the faster VMs. The PSSELB produces a load imbalance schedule by heavily under-utilizing the slower and moderate speed VMs that results in overloading the faster VMs. Likewise, the sufferage and TASA heuristics heavily under-utilize the slower VMs and slightly overload the faster



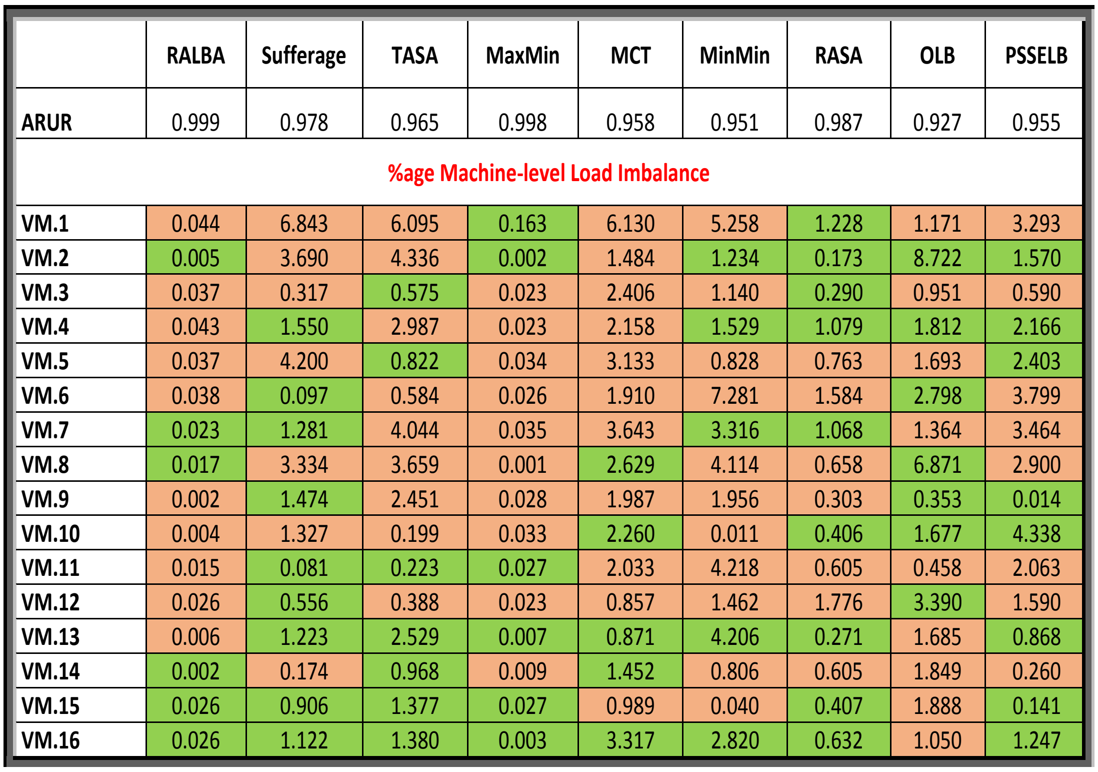
	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.998	0.916	0.984	0.997	0.938	0.855	0.947	0.861	0.892
%age Machine-level Load Imbalance									
VM.1	0.381	20.511	22.148	0.056	8.083	17.507	9.184	0.085	8.529
VM.2	0.007	23.882	12.482	0.093	10.85	8.366	6.675	9.597	21.507
VM.3	0	4.592	10.073	0.154	2.582	7.573	0.547	0.921	21.362
VM.4	0.133	20.123	19.401	0.118	1.183	18.497	0.486	22.279	12.735
VM.5	0.020	9.371	0.032	0.281	7.349	32.202	12.025	1.691	14.404
VM.6	0.388	1.196	19.563	0.023	1.424	37.712	9.348	3.802	18.117
VM.7	0.026	15.782	16.799	0.050	5.322	3.602	7.686	13.465	2.137
VM.8	0.034	2.166	17.324	0.072	5.196	11.681	9.575	8.856	10.24
VM.9	0.199	4.055	2.655	0.014	7.603	13.089	2.929	3.107	8.402
VM.10	0.168	0.827	8.983	0.070	3.896	15.218	0.641	18.821	8.485
VM.11	0.027	3.063	8.678	0.151	2.477	1.836	4.769	7.221	5.178
VM.12	0.036	4.802	0.594	0.015	2.745	14.918	1.007	2.735	3.417
VM.13	0.015	0.974	2.738	0.010	2.45	5.152	1.980	2.155	3.282
VM.14	0.021	0.595	2.289	0.002	1.063	2.253	2.296	1.369	0.93
VM.15	0.004	1.871	2.033	0.011	0.213	4.056	1.402	2.241	2.354
VM.16	0.045	1.936	2.957	0.003	2.564	4.134	1.465	2.325	2.682

FIGURE 5.9: Individual Machine-level Load Imbalance using 512x16 i-lohi - HCSP Instances.

VMs. The OLB overloads most of the VMs, while underutilizes the faster VMs. The MCT also repeats its behavior of under-utilizing the slower VMs and assigning slightly more workload to the faster VMs. The results portrays that even some algorithms achieve higher resource utilization but reproduces almost the resembling machine-level load imbalance to the other scheduling heuristics with least producing resource utilization (i.e., Sufferage, TASA, Min-Min, and PSSELB etc.)

Load-Imbalance using 512x16 i-hilo - HCSP Instances

Fig. 5.10 shows VMs level load-imbalance that is produced by scheduling heuristics using 512 x 16 i-hilo dataset. The highest resource utilization (i.e., 99.9%) is achieved by RALBA, and the Max-Min achieves second highest resource utilization (i.e., 99.8%). The RALBA and Max-Min heuristics provide almost balanced workload distribution among all VMs. Alternatively, the sufferage and TASA provides load imbalance by under-utilizing the slower VMs, and slightly overloading the faster VMs. Likewise, MCT under-utilizes the slower VMs and overloads the



	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.999	0.978	0.965	0.998	0.958	0.951	0.987	0.927	0.955
%age Machine-level Load Imbalance									
VM.1	0.044	6.843	6.095	0.163	6.130	5.258	1.228	1.171	3.293
VM.2	0.005	3.690	4.336	0.002	1.484	1.234	0.173	8.722	1.570
VM.3	0.037	0.317	0.575	0.023	2.406	1.140	0.290	0.951	0.590
VM.4	0.043	1.550	2.987	0.023	2.158	1.529	1.079	1.812	2.166
VM.5	0.037	4.200	0.822	0.034	3.133	0.828	0.763	1.693	2.403
VM.6	0.038	0.097	0.584	0.026	1.910	7.281	1.584	2.798	3.799
VM.7	0.023	1.281	4.044	0.035	3.643	3.316	1.068	1.364	3.464
VM.8	0.017	3.334	3.659	0.001	2.629	4.114	0.658	6.871	2.900
VM.9	0.002	1.474	2.451	0.028	1.987	1.956	0.303	0.353	0.014
VM.10	0.004	1.327	0.199	0.033	2.260	0.011	0.406	1.677	4.338
VM.11	0.015	0.081	0.223	0.027	2.033	4.218	0.605	0.458	2.063
VM.12	0.026	0.556	0.388	0.023	0.857	1.462	1.776	3.390	1.590
VM.13	0.006	1.223	2.529	0.007	0.871	4.206	0.271	1.685	0.868
VM.14	0.002	0.174	0.968	0.009	1.452	0.806	0.605	1.849	0.260
VM.15	0.026	0.906	1.377	0.027	0.989	0.040	0.407	1.888	0.141
VM.16	0.026	1.122	1.380	0.003	3.317	2.820	0.632	1.050	1.247

FIGURE 5.10: Individual Machine-level Load Imbalance using 512x16 i-hilo - HCSP Instances.

faster VMs. On the other hand, half of the machines are overloaded by Min-Min that results in the under-utilization of remaining machines. The same trend of machine-level load-imbalance is observed by the workload distribution mechanism of PSSELB and OLB heuristics as shown in Fig. 5.10. The RASA provides favorable load-balancing at VMs level as compared to sufferage, TASA, PSSELB, OLB, MCT, and Min-Min heuristics. RASA provides machine-level load balancing because it uses the Max-Min and Min-Min techniques in each alternate step of scheduling decisions, while the Max-Min has provided a promising load-balancing in this experiment.

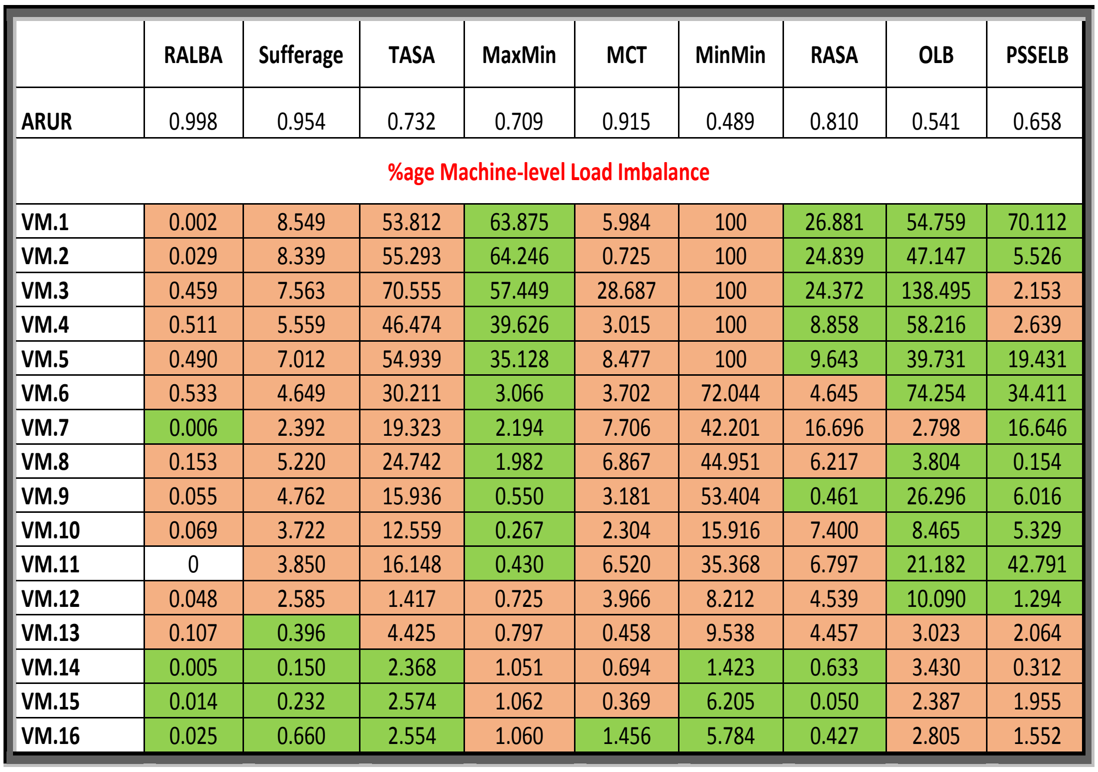
Load-Imbalance using 512x16 s-lohi - HCSP Instances

Fig. 5.11 shows VMs level load-imbalance produced by scheduling heuristics employing 512 x 16 s-lohi dataset. The resource utilization of Max-Min degraded to 70.9%, however, the RALBA retains its highest resource utilization (i.e., 99.8%) among all the given scheduling algorithms. The Max-Min provides a sever load-imbalance by heavily overloading the slower machines and slightly under-utilizing the faster machines. Contrariwise, the RALBA provides almost a complete load balancing at VM level. In addition, the VM.11 is fully utilized as per its computing share by RALBA. The Min-Min achieves the least resource utilization and provides absolutely undesirable load-imbalance at machine level (i.e. the slower machines VM.1–5 remain idle, the VM.6–13 are severely under-utilized, and the remaining faster VMs are overloaded). Likewise, the PSSELB heavily overloads the slower and moderate speed VMs, while under-utilize the remaining VMs. The RASA heuristic provides a sever load-imbalance by heavily overloading the slower VMs and under-utilizing the VMs with moderate computing powers. In the same way, the TASA provides a sever load-imbalance by heavily under-utilizing the slower VMs and overloading the faster VMs. Such a sever load-imbalance reproduced by RASA and TASA is due to the drastically degraded performance of Min-Min heuristic (the scheduling mechanism of RASA and TASA techniques use Min-Min in each alternate step of its scheduling decision as mentioned in Section 2.3).

The VM.1–6 and VM.8–12 are heavily overloaded and the remaining VMs are under-utilized by OLB technique. The sufferage under-utilizes the VMs with slower and moderate computing powers.

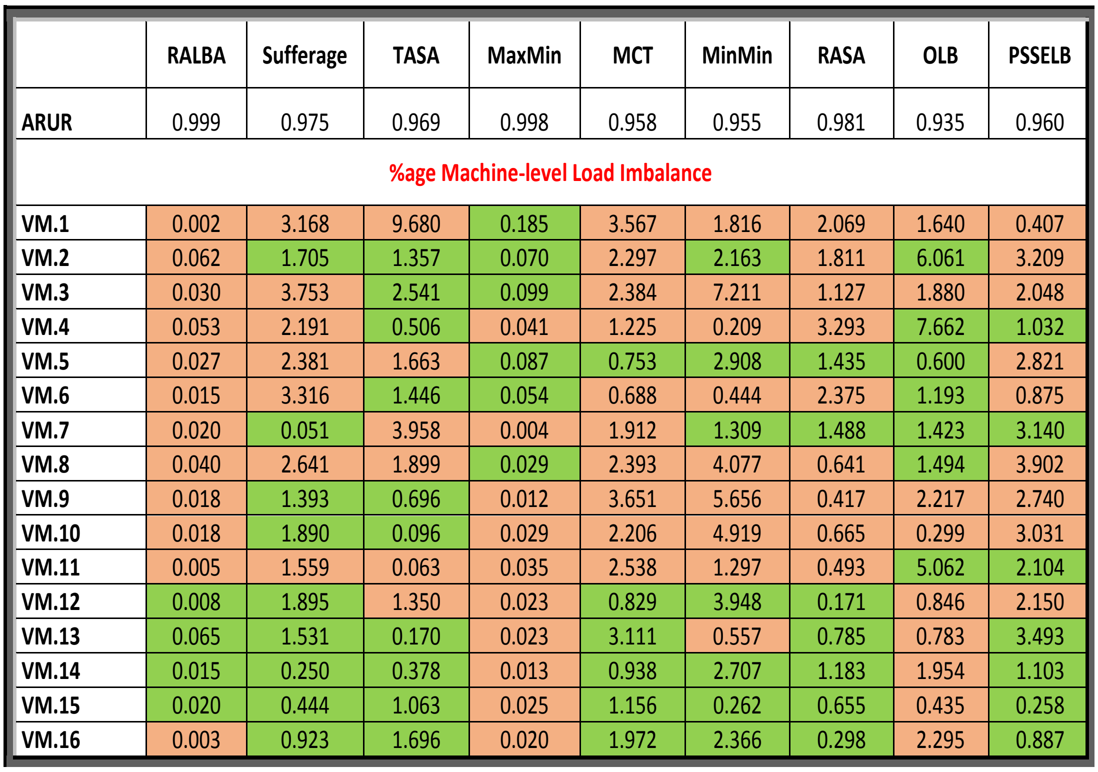
Load-Imbalance using 512x16 s-hilo - HCSP Instances

Fig. 5.12 shows VMs level load-imbalance produced by scheduling heuristics using 512 x 16 s-hilo dataset. The highest resource utilization (i.e., 99.9%) is achieved by RALBA and the Max-Min achieves the second highest resource utilization (99.8%). The RALBA and Max-Min provides almost a complete load-balancing at VMs-level. The sufferage and TASA under-utilize several VMs that results in overloading the remaining VMs. The MCT produces load-imbalance that cause the under-utilization of slower machines while overloading the faster VMs. The OLB heavily overloads the VM.2, VM.4, and VM.12 that produces the load-imbalance at remaining VMs. The Min-Min heavily overloads VM.2, VM.5, VM.12, VM.14, and VM.16 machines, however, severely under-utilizes the VM.3 and VM.8–10 computing machines. The RASA and PSSELB under-utilize the VMs with slow and moderate computing powers, while overloads the remaining VMs.



	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.998	0.954	0.732	0.709	0.915	0.489	0.810	0.541	0.658
%age Machine-level Load Imbalance									
VM.1	0.002	8.549	53.812	63.875	5.984	100	26.881	54.759	70.112
VM.2	0.029	8.339	55.293	64.246	0.725	100	24.839	47.147	5.526
VM.3	0.459	7.563	70.555	57.449	28.687	100	24.372	138.495	2.153
VM.4	0.511	5.559	46.474	39.626	3.015	100	8.858	58.216	2.639
VM.5	0.490	7.012	54.939	35.128	8.477	100	9.643	39.731	19.431
VM.6	0.533	4.649	30.211	3.066	3.702	72.044	4.645	74.254	34.411
VM.7	0.006	2.392	19.323	2.194	7.706	42.201	16.696	2.798	16.646
VM.8	0.153	5.220	24.742	1.982	6.867	44.951	6.217	3.804	0.154
VM.9	0.055	4.762	15.936	0.550	3.181	53.404	0.461	26.296	6.016
VM.10	0.069	3.722	12.559	0.267	2.304	15.916	7.400	8.465	5.329
VM.11	0	3.850	16.148	0.430	6.520	35.368	6.797	21.182	42.791
VM.12	0.048	2.585	1.417	0.725	3.966	8.212	4.539	10.090	1.294
VM.13	0.107	0.396	4.425	0.797	0.458	9.538	4.457	3.023	2.064
VM.14	0.005	0.150	2.368	1.051	0.694	1.423	0.633	3.430	0.312
VM.15	0.014	0.232	2.574	1.062	0.369	6.205	0.050	2.387	1.955
VM.16	0.025	0.660	2.554	1.060	1.456	5.784	0.427	2.805	1.552

FIGURE 5.11: Individual Machine-level Load Imbalance using 512x16 s-lohi - HCSP Instances.



	RALBA	Sufferage	TASA	MaxMin	MCT	MinMin	RASA	OLB	PSSELB
ARUR	0.999	0.975	0.969	0.998	0.958	0.955	0.981	0.935	0.960
%age Machine-level Load Imbalance									
VM.1	0.002	3.168	9.680	0.185	3.567	1.816	2.069	1.640	0.407
VM.2	0.062	1.705	1.357	0.070	2.297	2.163	1.811	6.061	3.209
VM.3	0.030	3.753	2.541	0.099	2.384	7.211	1.127	1.880	2.048
VM.4	0.053	2.191	0.506	0.041	1.225	0.209	3.293	7.662	1.032
VM.5	0.027	2.381	1.663	0.087	0.753	2.908	1.435	0.600	2.821
VM.6	0.015	3.316	1.446	0.054	0.688	0.444	2.375	1.193	0.875
VM.7	0.020	0.051	3.958	0.004	1.912	1.309	1.488	1.423	3.140
VM.8	0.040	2.641	1.899	0.029	2.393	4.077	0.641	1.494	3.902
VM.9	0.018	1.393	0.696	0.012	3.651	5.656	0.417	2.217	2.740
VM.10	0.018	1.890	0.096	0.029	2.206	4.919	0.665	0.299	3.031
VM.11	0.005	1.559	0.063	0.035	2.538	1.297	0.493	5.062	2.104
VM.12	0.008	1.895	1.350	0.023	0.829	3.948	0.171	0.846	2.150
VM.13	0.065	1.531	0.170	0.023	3.111	0.557	0.785	0.783	3.493
VM.14	0.015	0.250	0.378	0.013	0.938	2.707	1.183	1.954	1.103
VM.15	0.020	0.444	1.063	0.025	1.156	0.262	0.655	0.435	0.258
VM.16	0.003	0.923	1.696	0.020	1.972	2.366	0.298	2.295	0.887

FIGURE 5.12: Individual Machine-level Load Imbalance using 512x16 s-hilo - HCSP Instances.

The experimental results show that these scheduling techniques depict a varying behavior of machine-level load balancing using different instances of HCSP and GoCJ datasets. However, the RALBA technique shows a consistent behavior of workload distribution that produces a load-balanced computing schedule and the highest resource utilization using all instances of both HCSP and GoCJ datasets.

5.2.3 Results Discussion

This section delineates the analysis of the results obtained employing the aforementioned simulation-based experiments.

The resource utilization of RALBA in terms of VM-level load balancing has been compared with 8 different scheduling algorithms using two benchmark datasets (i.e., HCSP instances [24, 95] and GoCJ [62, 94]). Among all the schedulers, RALBA has attained the highest resource utilization for all the instances of HCSP and GoCJ datasets. Max-Min has achieved almost the same resource utilization as RALBA (i.e., 99.9% resource utilization) using c-hilo HCSP instances. Though, the RALBA scheduling heuristics has shown minor improvements (i.e., 0.1% improved ARUR using s-hilo, ihilo, and i-lohi HCSP instances) over the Max-Min in terms of resource utilization. However, the RALBA has attained an improved load balancing at individual machine level with shorter makespan (i.e., 0.12%, 0.14%, and 0.24% shorter makespan using the s-hilo, i-hilo, and i-lohi HCSP instances, respectively) than Max-Min.

Resource utilization is a very critical need for Cloud service providers. Higher resource utilization often leads to a balanced and energy-efficient scheduling [1, 44]. The issue of low resource utilization (as identified in the literature and accredited in our experimental results) should be addressed in a comprehensive manner. For optimal resource utilization, the workload should be assigned to the computing resources according to the resource capabilities and already assigned workload. The resource-aware and balanced distribution of Cloud jobs can eliminate the issues of under-/over-utilized computing resources and higher energy costs.

The empirical investigation has revealed an interesting fact that the scheduling heuristics with similar ARUR values have produced different load distribution at machine level. This different load distributions by the employed scheduling algorithms have constituted in varying makespan and throughput. The RALBA and TASA heuristics have attained 0.994 and 0.974 ARUR values respectively, using GoCJ workload with 512 Cloudlets and 16 VMs. In the same case, RALBA has produced 2.27% and 1.21% load–imbalance at two of the slowest VMs with less than 1% load–imbalance on all the other remaining VMs. However, TASA has caused more load–imbalance at VMs level (i.e., 9.32, 14.49, 3.95, 4.38, 1.05, 1.76, and 1.12% load–imbalance by seven VMs and less than 1% load–imbalance by the other remaining VMs used in the experimentation). Similarly, the Sufferage and Max-Min heuristics have attained almost the similar ARUR values (i.e., 0.983 and 0.986 ARUR values by Sufferage and Max-Min, respectively) using GoCJ workload with 1024 Cloudlets and 16 VMs (as discussed). In the same case, Sufferage has caused 9.65%, 3.41%, and 3.62% load imbalance at slowest three VMs and less than 1% load–imbalance by other remaining VMs. However, Max-Min has caused 1.52% load–imbalance at one of the slowest VM and less than 1% load–imbalance by the rest of the VMs. Similarly, the OLB and PSSELB heuristics have achieved the same ARUR value (i.e., 0.967 ARUR) using HCSP c-hilo instances. However, PSSELB has produced more VM-level load-balancing as compared to OLB heuristic, as discussed. PSSELB has caused 3.30, 2.30, 1.97, 1.70, 1.53, 1.29, 1.24, 1.13, and 1.13% load–imbalance by the nine VMs and less than 1% load–imbalance by remaining 7 VMs. On the other hand, OLB has resulted in 3.74, 3.31, 2.49, 1.72, 1.58, 1.32, 1.22, 1.17, 1.13, and 1.01% load–imbalance by the 10 VMs and less than 1% of load–imbalance by the remaining 6 VMs. The same trend of different load–imbalance produced at the machine level by different scheduling heuristics (i.e., producing exactly same or almost similar ARUR values) is discerned in the experiments using the benchmark HCSP instance and GoCJ workloads, as discussed in Section 5.2. Contrariwise, RALBA has outperformed all other heuristics by attaining the highest value of ARUR with minimum makespan and more VM–level load–balancing as compared to other eight scheduling heuristics using all GoCJ and HCSP instances.

In Section 5.2, the empirical investigation of VM-level load balancing is conducted and presented to address the Research Question 3 of this dissertation mentioned in Section 1.5.1, and concluded with following empirical findings.

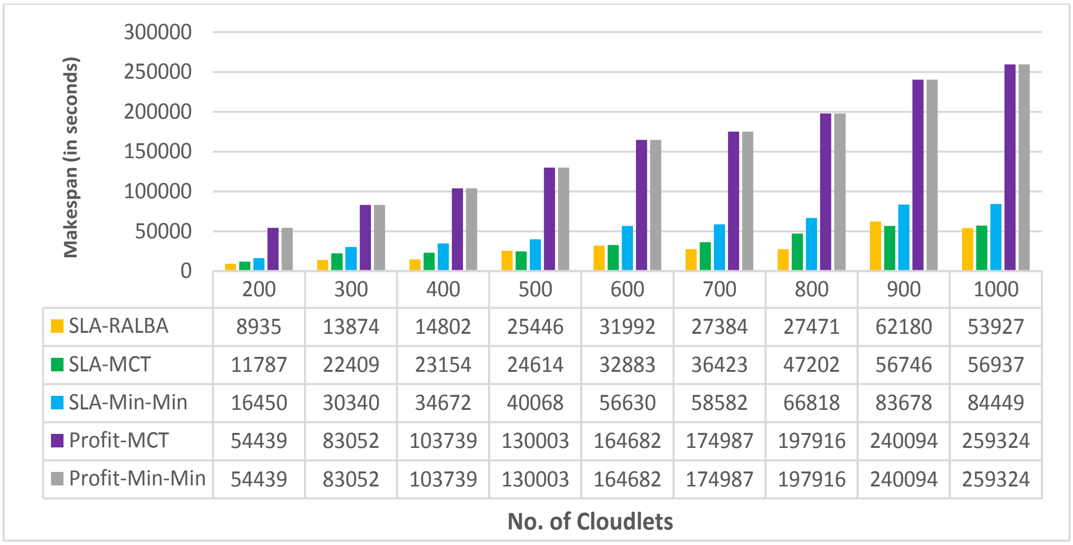
1. The utilization of the full capacity of computing resources in a Cloud is achieved by machine-level load-balancing in combination with higher resource utilization and minimal makespan. The RALBA heuristic is empirically endorsed to produce a more load-balanced schedule by releasing all the computing resources in almost similar ready time. This will effectuate the CSP to minimize the delay time in initializing the execution of next batch of workload.;
2. The CSP can maximize its revenue generation and minimize the energy-consumption in Cloud datacenters by executing the workload in minimal time with improved resource utilization and machine-level load balancing. In addition, the Cloud users can also be served with minimal execution time and cost for their HPC jobs on the Cloud.

5.3 Empirical Results - SLA-RALBA Scheduler

5.3.1 Results using GoCJ Dataset

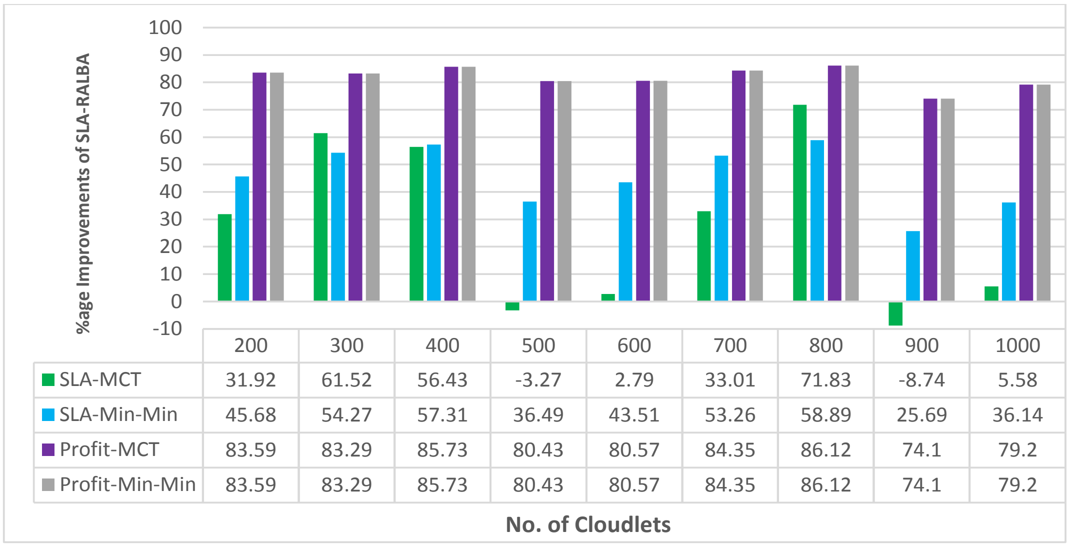
The efficacy of SLA-RALBA and existing scheduling techniques (i.e., Execution-MCT, Execution-Min-Min, Profit-MCT, Profit-Min-Min, SLA-MCT, and SLA-Min-Min) is evaluated by using ARUR, Expected Gain (i.e., execution cost) and makespan metrics. Each experiment is performed 10 times and the investigation is shown on average values. The empirical evaluation is performed using benchmark GoCJ [62, 94] and HCSP [24, 95] datasets.

Fig. 5.13 displays the makespan results for the instances of GoCJ with 200, 300, 400, 500, 600, 700, 800, 900, and 1000 Cloudlets, respectively. Fig. 5.14 shows the



	200	300	400	500	600	700	800	900	1000
🟨 SLA-RALBA	8935	13874	14802	25446	31992	27384	27471	62180	53927
🟩 SLA-MCT	11787	22409	23154	24614	32883	36423	47202	56746	56937
🟦 SLA-Min-Min	16450	30340	34672	40068	56630	58582	66818	83678	84449
🟪 Profit-MCT	54439	83052	103739	130003	164682	174987	197916	240094	259324
⬜ Profit-Min-Min	54439	83052	103739	130003	164682	174987	197916	240094	259324

Figure 5.13: Makespan Results - GoCJ Instances.

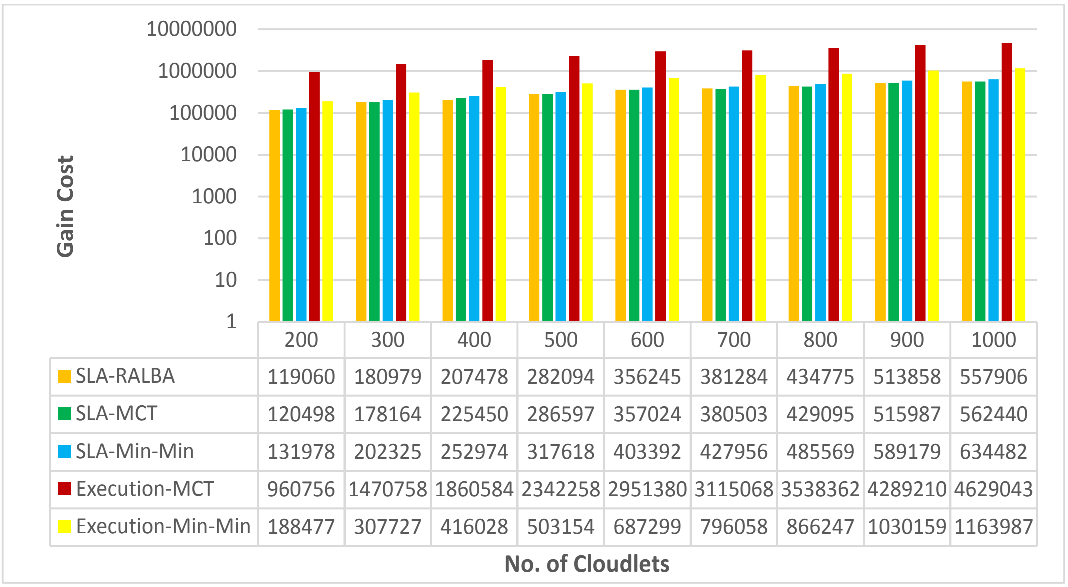


	200	300	400	500	600	700	800	900	1000
🟩 SLA-MCT	31.92	61.52	56.43	-3.27	2.79	33.01	71.83	-8.74	5.58
🟦 SLA-Min-Min	45.68	54.27	57.31	36.49	43.51	53.26	58.89	25.69	36.14
🟪 Profit-MCT	83.59	83.29	85.73	80.43	80.57	84.35	86.12	74.1	79.2
⬜ Profit-Min-Min	83.59	83.29	85.73	80.43	80.57	84.35	86.12	74.1	79.2

Figure 5.14: %age Makespan Improvements of SLA-RALBA – GoCJ Instances
.

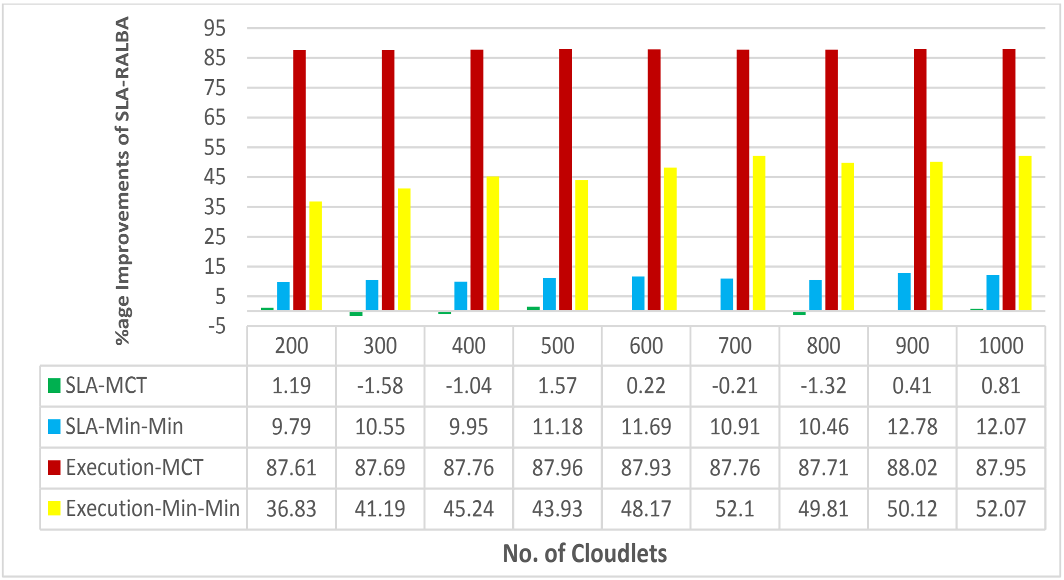
percentage improvement of SLA-RALBA in terms of makespan against SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min heuristics for the given instances of GoCJ dataset. On average the SLA-RALBA attains lower makespan than SLA-MCT (27.89% lower), SLA-Min-Min (45.69% lower), Profit-MCT (81.93% lower), and Profit-Min-Min (81.93% lower) heuristics.

Fig. 5.15 presents gain cost based results of SLA-RALBA as compared to SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min for the instances



	200	300	400	500	600	700	800	900	1000
🟧 SLA-RALBA	119060	180979	207478	282094	356245	381284	434775	513858	557906
🟩 SLA-MCT	120498	178164	225450	286597	357024	380503	429095	515987	562440
🟦 SLA-Min-Min	131978	202325	252974	317618	403392	427956	485569	589179	634482
🟥 Execution-MCT	960756	1470758	1860584	2342258	2951380	3115068	3538362	4289210	4629043
🟨 Execution-Min-Min	188477	307727	416028	503154	687299	796058	866247	1030159	1163987

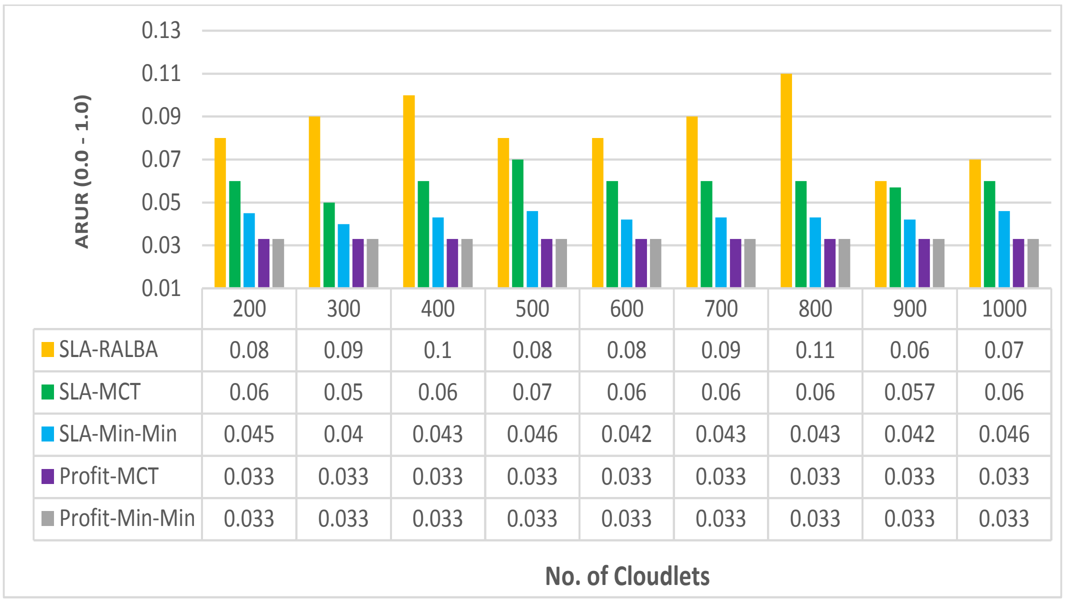
FIGURE 5.15: Gain Results - GoCJ Instances.



	200	300	400	500	600	700	800	900	1000
🟩 SLA-MCT	1.19	-1.58	-1.04	1.57	0.22	-0.21	-1.32	0.41	0.81
🟦 SLA-Min-Min	9.79	10.55	9.95	11.18	11.69	10.91	10.46	12.78	12.07
🟥 Execution-MCT	87.61	87.69	87.76	87.96	87.93	87.76	87.71	88.02	87.95
🟨 Execution-Min-Min	36.83	41.19	45.24	43.93	48.17	52.1	49.81	50.12	52.07

FIGURE 5.16: %age Gain Improvements of SLA-RALBA – GoCJ Instances .

of GoCJ dataset with 200, 300, 400, 500, 600, 700, 800, 900, and 1000 jobs, respectively. Fig. 5.16 shows the percentage improvements of SLA-RALBA in terms of gain cost against SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min heuristics. On average the SLA-RALBA achieves lower gain cost as compared to SLA-MCT (0.01% lower), SLA-Min-Min (11.04% lower), Execution-MCT (87.82% lower), and Execution-Min-Min (46.61% lower) heuristics, respectively.



	200	300	400	500	600	700	800	900	1000
🟨 SLA-RALBA	0.08	0.09	0.1	0.08	0.08	0.09	0.11	0.06	0.07
🟩 SLA-MCT	0.06	0.05	0.06	0.07	0.06	0.06	0.06	0.057	0.06
🟦 SLA-Min-Min	0.045	0.04	0.043	0.046	0.042	0.043	0.043	0.042	0.046
🟪 Profit-MCT	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033
⬜ Profit-Min-Min	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033	0.033

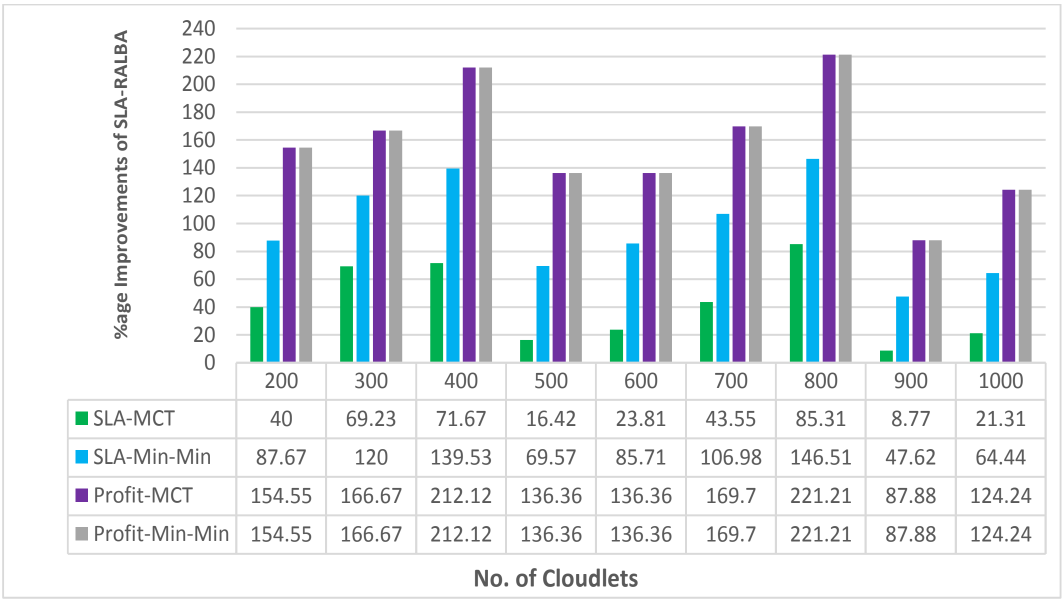
FIGURE 5.17: ARUR Results - GoCJ Instances.

Fig. 5.17 presents the ARUR based results of SLA-RALBA as compared to SLA-MCT, SLA-Min-Min, and Profit-MCT and Profit-Min-Min for the instances of GoCJ dataset with 200, 300, 400, 500, 600, 700, 800, 900, and 1000 jobs, respectively.

Fig. 5.18 shows the percentage improvement of SLA-RALBA in terms of ARUR against SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min heuristics. On average, the SLA-RALBA attains higher resource utilization as compared to SLA-MCT (42.23% higher), SLA-Min-Min (96.34% higher), Profit-MCT (156.57% higher), and Profit-Min-Min (156.57% higher) heuristics. On the other hand, the Execution-MCT and Execution-Min-Min attain the highest resource utilization as compared to SLA-RABA. In contrary, the Execution-MCT and Execution-Min-Min attain 1965% and 982% higher gain cost as compared to SLA-RALBA, because these don't consider the SLA parameters in scheduling decisions.

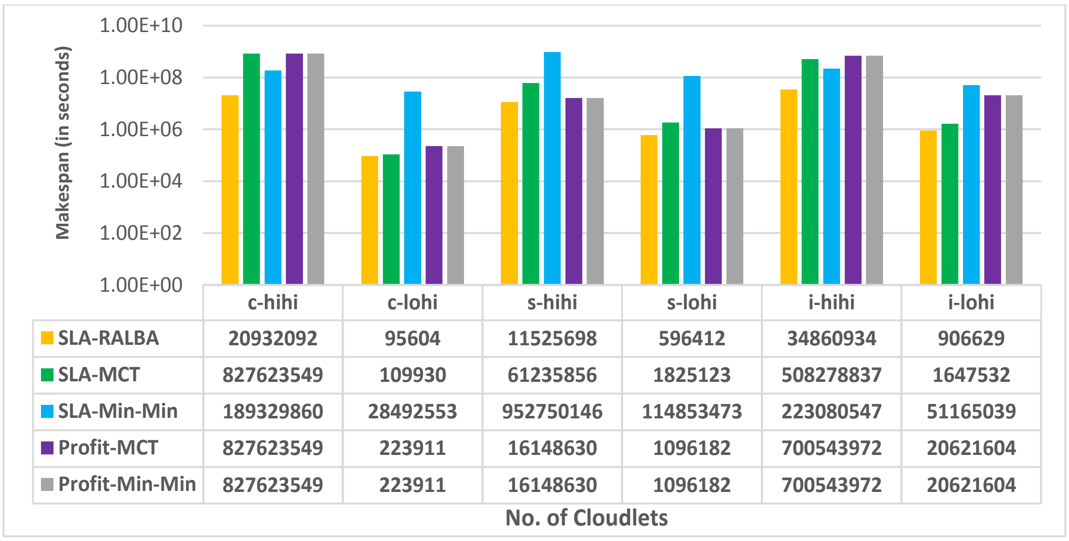
5.3.2 Results using HCSP Dataset

Fig. 5.19 presents the makespan based results for the six instances of 512x16 HCSP dataset, namely u-c-hihi, u-c-lohi, u-s-hihi, u-s-lohi, u-i-hihi, and u-i-lohi.



	200	300	400	500	600	700	800	900	1000
🟩 SLA-MCT	40	69.23	71.67	16.42	23.81	43.55	85.31	8.77	21.31
🟦 SLA-Min-Min	87.67	120	139.53	69.57	85.71	106.98	146.51	47.62	64.44
🟪 Profit-MCT	154.55	166.67	212.12	136.36	136.36	169.7	221.21	87.88	124.24
◼ Profit-Min-Min	154.55	166.67	212.12	136.36	136.36	169.7	221.21	87.88	124.24

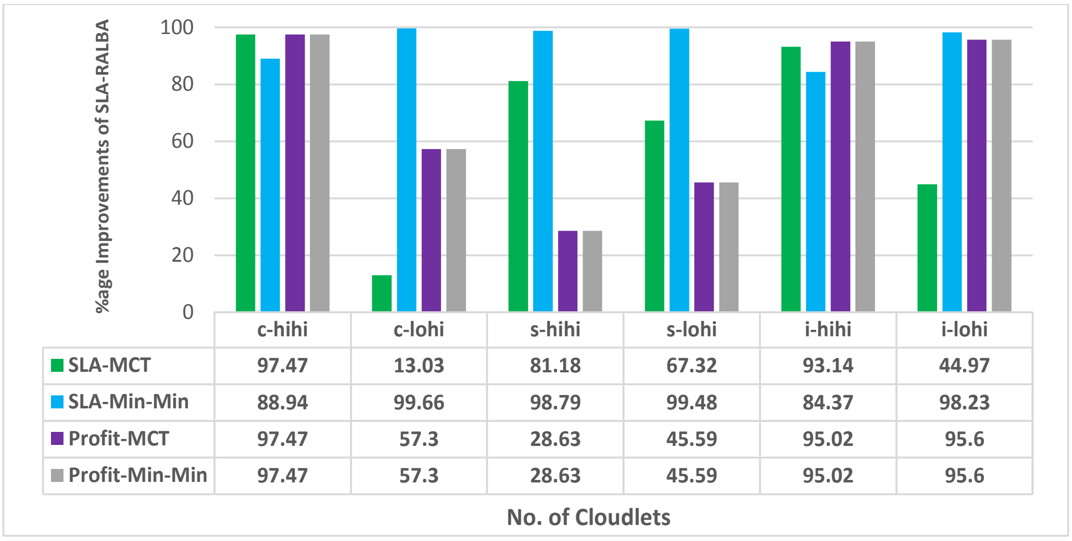
FIGURE 5.18: %age ARUR Improvements of SLA-RALBA – GoCJ Instances .



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟨 SLA-RALBA	20932092	95604	11525698	596412	34860934	906629
🟩 SLA-MCT	827623549	109930	61235856	1825123	508278837	1647532
🟦 SLA-Min-Min	189329860	28492553	952750146	114853473	223080547	51165039
🟪 Profit-MCT	827623549	223911	16148630	1096182	700543972	20621604
◼ Profit-Min-Min	827623549	223911	16148630	1096182	700543972	20621604

FIGURE 5.19: Makespan Results - HCSP Instances.

Fig.5.20 shows the percentage improvement of SLA-RALBA in terms of makespan against SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min heuristics for the given instances of HCSP dataset. On average the SLA-RALBA attains lower makespan than SLA-MCT (66.17% lower), SLA-Min-Min (94.91% lower), Profit-MCT (69.94% lower), and Profit-Min-Min (69.94% lower) heuristics. Fig. 5.21 presents gain cost based results of SLA-RALBA as compared to SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min for the given instances of

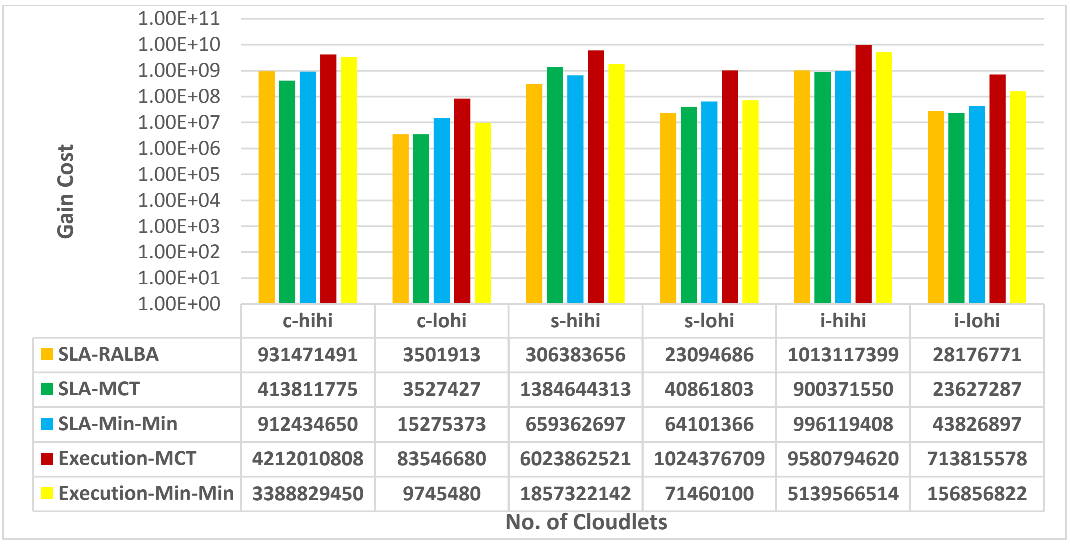


	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟩 SLA-MCT	97.47	13.03	81.18	67.32	93.14	44.97
🟦 SLA-Min-Min	88.94	99.66	98.79	99.48	84.37	98.23
🟪 Profit-MCT	97.47	57.3	28.63	45.59	95.02	95.6
⬜ Profit-Min-Min	97.47	57.3	28.63	45.59	95.02	95.6

FIGURE 5.20: %age Makespan Improvements of SLA-RALBA – HCSP Instances .

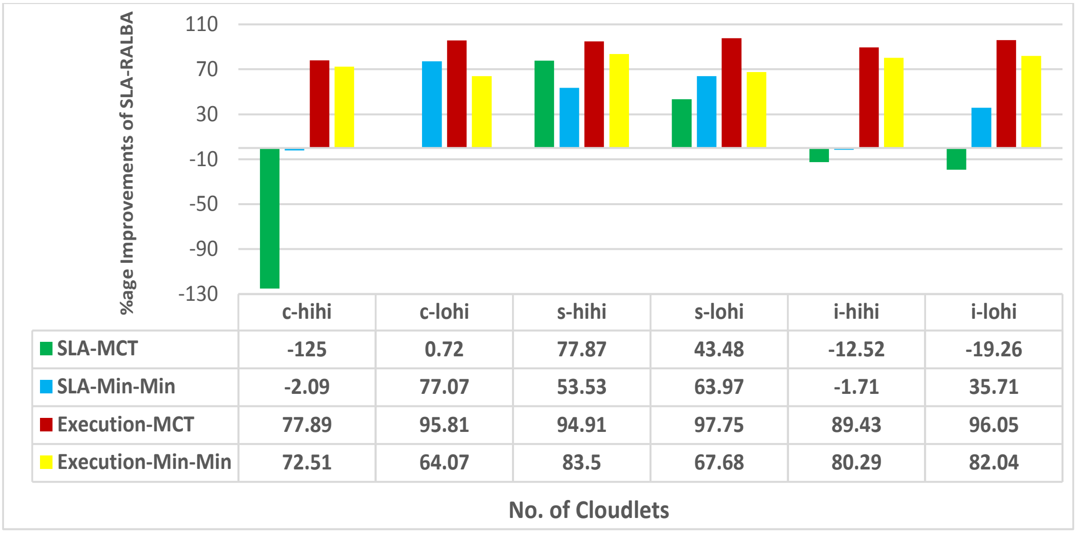
512x16 HCSP dataset. Likewise, Fig. 5.22 shows the percentage improvements of SLA-RALBA in terms of gain cost against SLA-MCT, SLA-Min-Min, Execution-MCT, and Execution-Min-Min heuristics. On average the SLA-RALBA achieves lower gain cost as compared to SLA-Min-Min (37.74% lower), Execution-MCT (91.97% lower), and Execution-Min-Min (75.02% lower) heuristics, respectively. However, SLA-MCT achieves 5.79% lower gain cost on average (due to much better performance of SLA-MCT in terms of gain cost only using u-c-hihi instance of 512x16 HCSP dataset) as compared to the proposed SLA-RALBA.

Fig. 5.23 presents the ARUR based results of SLA-RALBA as compared to SLA-MCT, SLA-Min-Min, and Profit-MCT and Profit-Min-Min for given instances of 512x16 HCSP dataset. Fig. 5.24 shows the percentage improvement of SLA-RALBA in terms of ARUR against SLA-MCT, SLA-Min-Min, Profit-MCT, and Profit-Min-Min algorithms. On average, the SLA-RALBA attains higher resource utilization as compared to SLA-MCT (253.95% higher), SLA-Min-Min (698.15% higher), Profit-MCT (871.39% higher), and Profit-Min-Min (871.39% higher) algorithms. On the other hand, the Execution-MCT and Execution-Min-Min heuristics attains 35.23% and 4.54% highest resource utilization on average as compared to SLA-RABA. In contrary, the Execution-MCT and Execution-Min-Min algorithms



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟧 SLA-RALBA	931471491	3501913	306383656	23094686	1013117399	28176771
🟩 SLA-MCT	413811775	3527427	1384644313	40861803	900371550	23627287
🟦 SLA-Min-Min	912434650	15275373	659362697	64101366	996119408	43826897
🟥 Execution-MCT	4212010808	83546680	6023862521	1024376709	9580794620	713815578
🟨 Execution-Min-Min	3388829450	9745480	1857322142	71460100	5139566514	156856822

FIGURE 5.21: Gain Results - HCSP Instances.



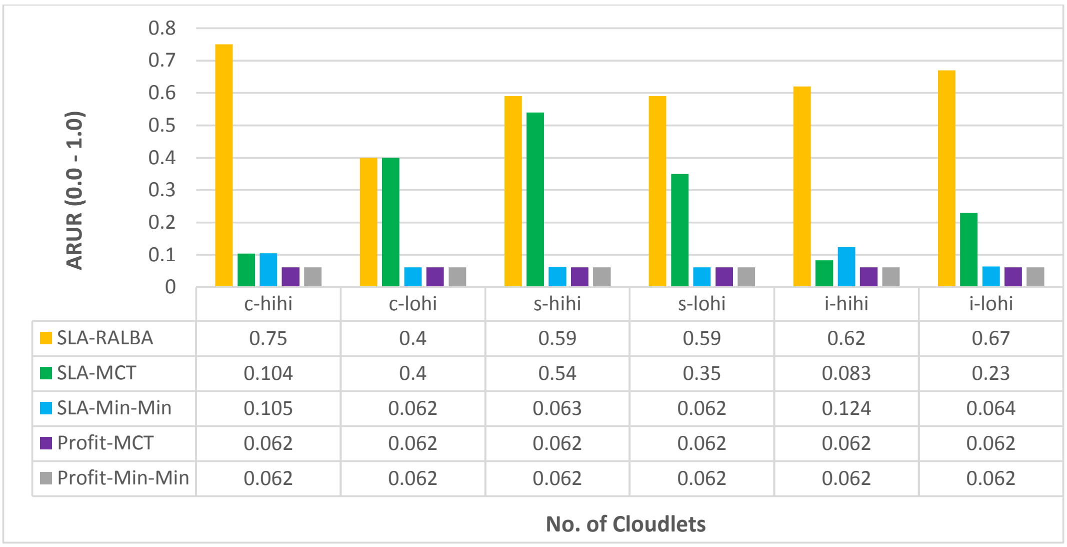
	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟩 SLA-MCT	-125	0.72	77.87	43.48	-12.52	-19.26
🟦 SLA-Min-Min	-2.09	77.07	53.53	63.97	-1.71	35.71
🟥 Execution-MCT	77.89	95.81	94.91	97.75	89.43	96.05
🟨 Execution-Min-Min	72.51	64.07	83.5	67.68	80.29	82.04

FIGURE 5.22: %age Gain Improvements of SLA-RALBA – HCSP Instances .

attains higher gain cost as compared to SLA-RALBA as shown in Fig. 5.22.

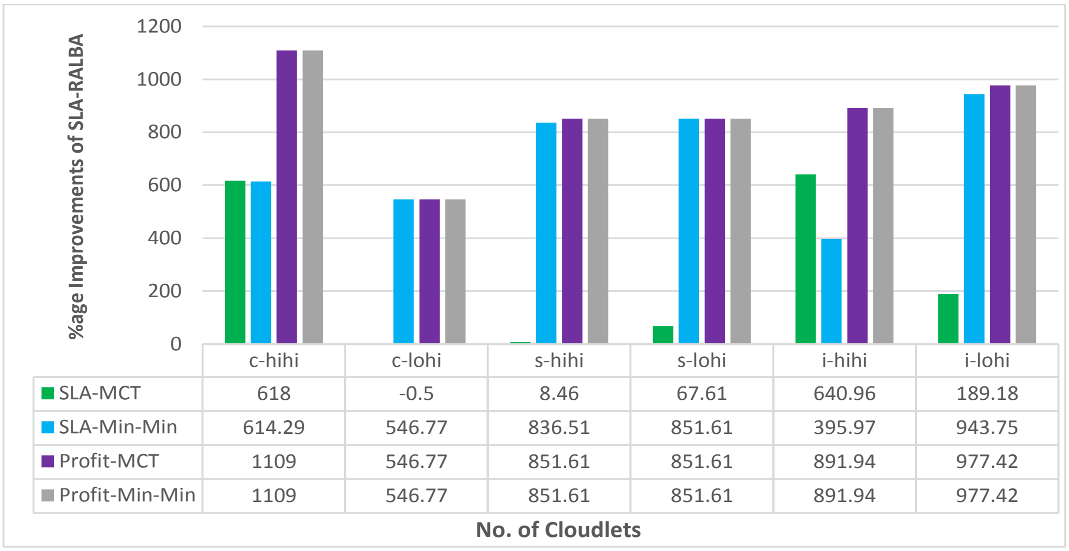
5.3.3 Results Discussion

A substantial improvement is witnessed in the efficacy of SLA-RALBA in terms of resource utilization against existing SLA-aware and SLA-spare scheduling algorithms as presented in Fig. 5.17 – 5.18 and Fig. 5.23 – 5.24. Similarly, a prominent enhancement in the makespan results are also achieved except against



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟨 SLA-RALBA	0.75	0.4	0.59	0.59	0.62	0.67
🟩 SLA-MCT	0.104	0.4	0.54	0.35	0.083	0.23
🟦 SLA-Min-Min	0.105	0.062	0.063	0.062	0.124	0.064
🟪 Profit-MCT	0.062	0.062	0.062	0.062	0.062	0.062
⬜ Profit-Min-Min	0.062	0.062	0.062	0.062	0.062	0.062

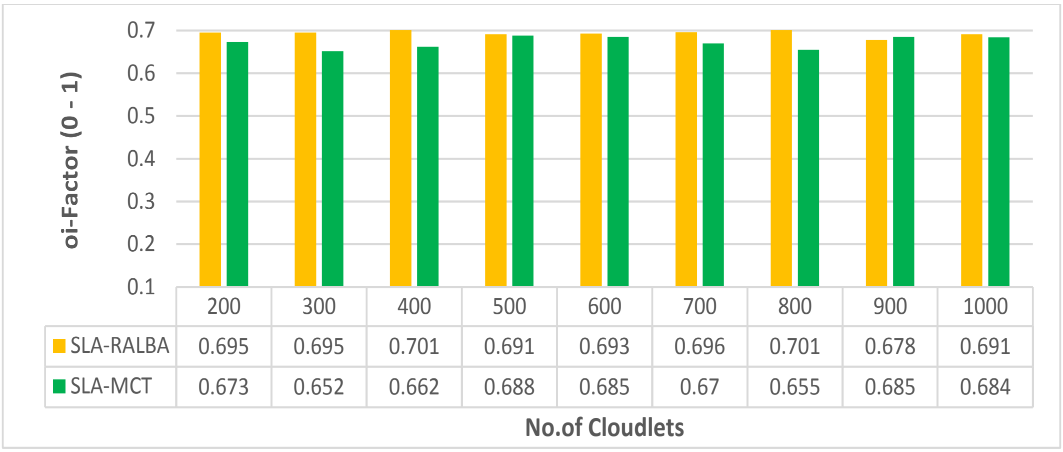
FIGURE 5.23: ARUR Results - HCSP Instances.



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟩 SLA-MCT	618	-0.5	8.46	67.61	640.96	189.18
🟦 SLA-Min-Min	614.29	546.77	836.51	851.61	395.97	943.75
🟪 Profit-MCT	1109	546.77	851.61	851.61	891.94	977.42
⬜ Profit-Min-Min	1109	546.77	851.61	851.61	891.94	977.42

FIGURE 5.24: %age ARUR Improvements of SLA-RALBA – HCSP Instances .

the makespan-based results of SLA-MCT for a couple of GoCJ instances (i.e., 500 and 900 Cloudlets instances, shown in Fig. 5.14). The SLA-RALBA attained a significant reduction in expected gain against existing algorithms except SLA-MCT for some GoCJ and HCSP instances (i.e., in case of c-hihi, i-hihi, and i-lohi instances of HCSP, and GoCJ instances with 300, 400, and 800 Cloudlets, respectively). However, SLA-RALBA outperformed the given scheduling heuristics (including SLA-MCT) by offering a well-balanced among makespan, expected gain and resource utilization on Cloud using all instances of HCSP and GoCJ datasets.



	200	300	400	500	600	700	800	900	1000
🟨 SLA-RALBA	0.695	0.695	0.701	0.691	0.693	0.696	0.701	0.678	0.691
🟩 SLA-MCT	0.673	0.652	0.662	0.688	0.685	0.67	0.655	0.685	0.684

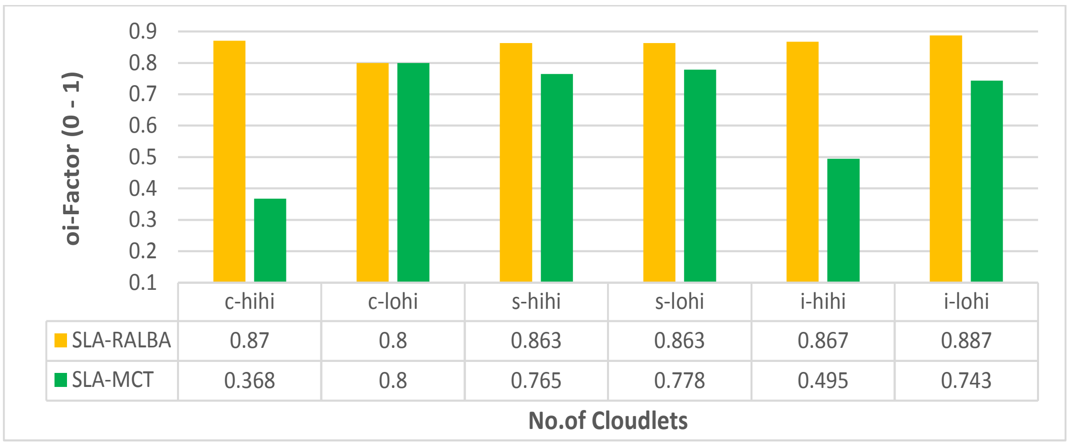
FIGURE 5.25: oi-Factor of SLA-RALBA and SLA-MCT - GoCJ Instances.

To prove the improved performance of SLA-RALBA against the SLA-MCT, the oi-Factor is calculated and compared as follows. The max-min normalization is adopted to normalize the values of makespan and expected gain (achieved by SLA-RALBA and SLA-MCT in the experimental results) to a number ranged of 0 – 1. [121, 122]. The normalized values of makespan and expected gain is represented as n-Makespan and n-EGain. The lower values (between 0.0 and 1.0) of n-Makespan and n-EGain, for a scheduling algorithm, show it to be the better one. However, the higher value of ARUR identifies an scheduling algorithm to be the better one. Therefore, n-Makespan' and n-EGain' is found by subtracting the values of n-Makespan and n-EGain from 1, respectively. The oi-Factor of an algorithm is calculated by averaging its values of n-Makespan', n-EGain' and ARUR. Fig. 5.25 and Fig. 5.26 show the overall improvement of SLA-RALBA against SLA-MCT using GoCJ and HCSP datasets, which proves a promising balance of SLA-RALBA among makespan, expected gain and resource utilization.

5.4 Summary

Resource-Aware Scheduling

In-depth analysis of the current state-of-the-art scheduling heuristics results in inefficient resource utilization, reduced makespan, and low throughput due to the



	c-hihi	c-lohi	s-hihi	s-lohi	i-hihi	i-lohi
🟨 SLA-RALBA	0.87	0.8	0.863	0.863	0.867	0.887
🟩 SLA-MCT	0.368	0.8	0.765	0.778	0.495	0.743

FIGURE 5.26: oi-Factor of SLA-RALBA and SLA-MCT - HCSP Instances.

imbalanced mapping of Cloud jobs. The RALBA is a reource-aware technique instead of a fair scheduling algorithm that considers the computing powers of machines in the computing environment and the computation requirements of jobs in the dataset for scheduling decisions. This Chapter presents the performance analysis of a novel Cloud scheduling heuristic RALBA that ensures improved resource utilization with minimal makespan and increased throughput. The performance of RALBA has been compared with the state-of-the-art scheduling heuristics in terms of makespan, resource utilization, and throughput. The detailed analysis of experimental results has shown that RALBA has successively attained lower makespan, higher throughput, and improved resource utilization as compared to the existing Cloud scheduling heuristics (i.e., TASA, Sufferage, Max–Min, RASA, Min–Min, MCT, RS, and RR).

VM-Level Load Balancing

This Chapter has focused on the empirical investigation of 09 state-of-the-art scheduling heuristics, OLB, MCT, Min-Min, Max-Min, RASA, Sufferage, TASA, PSSELB, and RALBA in a renowned CloudSim simulation environment. To scrutinize the potential of each heuristic under varying workload compositions and computing capabilities have been assessed by employing two different benchmark scientific workloads (i.e., HCSP instances and GoCJ datasets). These workloads have been configured using different heterogeneity of jobs and machines in the computing environment. The RALBA heuristic has attained the highest resource utilization among nine scheduling heuristics for all the instances of benchmark datasets (i.e., 93.4% resource utilization for the c-lohi HCSP instances and 99.9% resource utilization for ihilo, s-hilo, and c-hilo HCSP instances). However, for few HCSP instances, Max-Min heuristic has also achieved the high resource utilization with a slighter difference in ARUR value than RALBA. Although Max-Min has achieved slightly reduced resource utilization than RALBA in these three cases (i.e., i-hilo, s-hilo, and c-hilo HCSP instances). However, RALBA has caused the highest resource utilization in consolidation with more machine-level load balancing and attained improved results of makespan and throughput as compared to OLB, MCT, Min-Min, Max-Min, RASA, Sufferage, TASA, and PSSELB heuristics. Moreover, RALBA has achieved significantly improved resource utilization as compared to Max-Main for two instances of HCSP dataset (i.e. Max-Min achieves 70.9, and 62.0% resource utilization for s-lohi and c-lohi HCSP instances, while RALBA achieves 99.8, and 93.4% resource utilization for s-lohi and c-lohi HCSP instance, respectively) and two instances of GoCJ dataset (i.e. Max-Min achieves 43.4%, and 80.5% resource utilization for GoCJ workload with 256 and 512 Cloudlets, while RALBA has achieved 99.3, and 99.4% resources utilization for the same GoCJ workload, respectively).

SLA and Resource-Aware Scheduling

A comprehensive investigation of the current SLA-based heuristics fallouts in a reduced resource utilization and increased execution cost due to the improper allocation of jobs to VM and the straggling jobs in scheduling. The proposed novel technique SLA-RALBA ensures enhanced resource utilization with reduced execution time and cost. The performance of SLA-RALBA has been compared using benchmark HCSP and GoCJ datasets with the scheduling algorithms in terms of execution time, cost and resource utilization. The exhaustive examination of pragmatic results has shown that SLA-RALBA has presented an improved balance among the resource utilization, execution time, and cost as compared to the existing algorithms (i.e., Execution-MCT, Execution-Min-Min, Profit-MCT, Profit-Min-Min, SLA-MCT, and SLA-Min-Min).

In future, we intend to consider other important QoS aspects to enhance the functionalities of SLA-RALBA; especially to deal with the communication cost involved in the scheduling on Cloud.

SLA-RALBA as compared to RALBA technique: RALBA is a resource-aware scheduler, while SLA-RALBA is resource-aware as well as SLA-aware scheduler in Cloud computing. Therefore, the SLA-RALBA has to consider the SLA-type of user jobs for satisfying the SLA requirements in job scheduling. If a non SLA-based workload is submitted, then both RALBA and SLA-RALBA will behave in the same manner and will exhibits the same performance in terms of resource utilization, makespan and throughput. On the other hand, if SLA-based workload is submitted, then RALBA will produce a drastic increase in the resource utilization as compared to SLA-RALBA (as RALBA doesn't considers the SLA types of jobs). However, RALBA will not satisfy the SLA requirements of jobs (SLA-3 jobs may receive expensive execution cost, and SLA-1 jobs maybe executed in longer completion time) in the workload, and this is not desired by CSPs and Cloud users. While, SLA-RALBA will provide the scheduling that will satisfy the SLA-requirements of all jobs in workload but will produce much reduced resource utilization as compared to RALBA technique. Further, It is evident in the experimental results (presented in section 5.3) that SLA-RALBA still produces drastic increase in resource utilization as compared to existing SLA-aware scheduling algorithms.

Chapter 6

Conclusions and Future Work

This chapter concludes the thesis by presenting the summary of main contributions related to proposed benchmark GoCJ dataset, resource-aware load balancing, and cost-efficient resource-aware load balancing techniques. Moreover, this Chapter provides an insight into promising open-issues for future work in the area of Cloud computing.

6.1 Contributions

Following is the summary of major contributions of the thesis. We present two novel Cloud scheduling algorithms: 1) RALBA, and 2) SLA-RALBA. The RALBA ensures improved resource utilization in a datacenter with reduced mankespan, and higher throughput. Likewise, the SLA-RALBA scheduler provides enhanced resource utilization with reduced makespan and execution cost.

Existing state-of-the-art algorithms are either based on completion time or execution time of jobs on each machines in a computing system. Empirical analysis shows that these scheduling techniques produce very poor resource utilization in Cloud datacenters. However, the proposed RALBA algorithm presents the idea of job scheduling based on computation share of each computing machine. The computation share based scheduling methodology introduces computation-awareness both at machines and HPC applications level, which produces a significant increase in the resource utilization. In addition, a scalable computing environment will not affect the performance of RALBA because the increase or decrease in the number of computing machines will not affect the computation share of each machine. This will ensure a load-balanced job scheduling with improved resource utilization.

SLA-aware scheduling algorithms ensure the QoS by considering both the execution time and cost of jobs. On the other hand, these algorithms drastically reduce the resource utilization in Cloud datacenters. Therefore, the idea of computation share based scheduling is inculcated in a new proposed SLA-RALBA algorithm that significantly increased the resource utilization while providing an improved balance between execution time and cost. The performance of SLA-RALBA is empirically examined using two benchmark datasets against the exiting SLA-aware algorithms.

These proposed schedulers (i.e., RALBA and SLA-RALBA) effectuate the CSP to minimize the delay time in initializing the execution of next batch of workload, maximize the revenue generation of CSP, and minimize the energy consumption in Cloud datacenter by improving the resource utilization. In addition, a new GoCJ dataset is proposed that can be used by the Cloud research community for job scheduling, resource-allocation policies, and benchmark-based performance analysis in distributed and Cloud computing.

6.2 Future Directions

In this thesis, a novel resource-aware load balancing technique, a benchmark dataset for distributed and Cloud computing, and a novel cost- and resource-efficient load balancing technique with improved resource utilization are presented. There are several potential research directions which could be explored and investigated. These research directions are as follows:

RALBA is a batch-dynamic load balancing mechanism to schedule non-preemptive, compute-intensive, and independent jobs on Cloud. The functionality of RALBA will be enhanced to deal with unexpectedly slow computing resources and to cope with the sudden resource failures and a fault-tolerant scheduling mechanism.

RALBA scheduler represents a static mechanism for intra-batch jobs scheduling. However, RALBA mechanism will be enhanced for dynamic scheduling. In the future, we intend to propose and design the dynamic mechanism based on RALBA (i.e., Dynamic-RALBA) with a single Fill-Scheduler instead of two sub-schedulers (i.e., Fill-Scheduler and Spill-Scheduler). The Fill-Scheduler of Dynamic-RALBA will be based on the idea of employing a mechanism to revise the computing share of VMs for the completing jobs on those VMs.

The proposed SLA-RALBA provides an improved balance among resource utilization, execution time, and cost as compared to existing state-of-the-art heuristics. In the future, the SLA-RALBA mechanism could be amended to consider other important QoS aspects specially to deal with the communication cost involved. In addition, a fault-tolerant version of the SLA-RALBA could possibly be proposed and developed.

GoCJ benchmark dataset is based on the jobs trend observed for a specific Cloud datacenter (i.e., Google cluster traces). The modified version of GoCJ benchmark dataset can be enhanced with a generalized original dataset (to be input to the MC simulation) based on a diversified analysis of multiple real Cloud infrastructures (i.e., Google cluster traces [101], Facebook Hadoop [100] workload, and Yahoo cluster traces [102] etc.). In addition, the GoCJ can be equipped with SLA- and deadline-aware job parameters based on a comprehensive literature review that would be useful in the performance analysis of SLA-aware and constraints-oriented resource-allocation and scheduling policies in Cloud.

Bibliography

[1] 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.

[2] D. F. Parkhill, *Challenge of the computer utility.* Addison-Wesley, 1966.

[3] 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.

[4] N. Draft, "Working definition of cloud computing v15," accessed on 15-02-2019. [Online]. Available: csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc

[5] M. A. R. Sossa, "Resource provisioning and scheduling algorithms for scientific workflows in cloud computing environments," Ph.D. dissertation, University of Melbourne, Department of Computing and Information Systems, 2016.

[6] 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.

[7] "Saleforce crm," accessed on 15-01-2019. [Online]. Available: http://www.salesforce.com/platform

[8] "Dropbox software," accessed on 15-12-2018. [Online]. Available: https://www.dropbox.com/

[9] "Google drive: Free cloud storage for personal use," accessed on 15-12-2018. [Online]. Available: https://www.google.com/drive/

[10] "Facebook, inc," accessed on 15-12-2018. [Online]. Available: https://www.facebook.com/

[11] "Google app engine," accessed on 15-01-2019. [Online]. Available: http://code.google.com/appengine

[12] "Windows azure," accessed on 15-01-2019. [Online]. Available: www.microsoft.com/azure

[13] "Amazon elastic cloud computing," accessed on 15-12-2018. [Online]. Available: https://aws.amazon.com/ec2/

[14] "Openstack: Open source software for creating private and public clouds." accessed on 15-12-2018. [Online]. Available: https://www.openstack.org/

[15] "Cloud hosting, cloud computing and hybrid infrastructure from gogrid," accessed on 15-01-2019. [Online]. Available: http://www.gogrid.com

[16] "Flexiscale cloud comp and hosting," accessed on 15-01-2019. [Online]. Available: www.fleiscale.com

[17] L. Cherkasova, D. Gupta, A. Vahdat *et al.*, "When virtual is harder than real: Resource allocation challenges in virtual machine based it environments," *Hewlett Packard Laboratories, Technical Report HPL-2007-25*, pp. 1–9, 2007.

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

[19] X. Li, Y. Mao, X. Xiao, and Y. Zhuang, "An improved max-min task-scheduling algorithm for elastic cloud," in *2014 International Symposium on Computer, Consumer and Control.* IEEE, 2014, pp. 340–343.

[20] K. R. Babu and P. Samuel, "Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud," in *Innovations in bio-inspired computing and applications*. Springer, 2016, pp. 67–78.

[21] 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.

[22] 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.

[23] T. Mathew, K. C. Sekaran, and J. Jose, "Study and analysis of various task scheduling algorithms in the cloud computing environment," in *Int. conf. on adv. in computing, com. and informatics*. IEEE, 2014, pp. 658–664.

[24] T. D. Braun, H. J. Siegel, N. Beck *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed computing*, vol. 61, no. 6, pp. 810–837, 2001.

[25] S. H. H. Madni, M. S. A. Latiff, M. Abdullahi, M. J. Usman *et al.*, "Performance comparison of heuristic algorithms for task scheduling in iaas cloud computing environment," *PloS one*, vol. 12, no. 5, pp. 1–26, 2017.

[26] L. Hongyou, W. Jiangyong, P. Jian, *et al.*, "Energy-aware scheduling scheme using workload-aware consolidation technique in cloud data centres," *China Communications*, vol. 10, no. 12, pp. 114–124, 2013.

[27] X. Yu and X. Yu, "A new grid computation-based min-min algorithm," in *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, vol. 1. IEEE, 2009, pp. 43–45.

[28] S. K. Panda, P. Agrawal, *et al.*, "Skewness-based min-min max-min heuristic for grid task scheduling," in *Proceedings of Fourth Int. Conference on Adv. Comp. and Comm. Techn.*. IEEE Computer Society, 2014, pp. 282–289.

[29] S. Parsa and R. Entezari-Maleki, “Rasa: a new grid task scheduling algorithm,” *International Journal of Digital Content Technology and its Applications*, vol. 3, no. 4, pp. 91–99, 2009.

[30] O. Elzeki, M. Reshad, *et al.*, “Improved max-min algorithm in cloud computing,” *Int. Journal of Comp. App.*, vol. 50, no. 12, pp. 22–27, 2012.

[31] E. K. Tabak, *et al.*, “Improving the performance of independent task assignment heuristics minmin, maxmin and sufferage,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1244–1256, 2014.

[32] J. Maipan-uku, A. Muhammed, A. Abdullah, and M. Hussin, “Max-average: An extended max-min scheduling algorithm for grid computing environtment,” *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 8, no. 6, pp. 43–47, 2016.

[33] A. Aditya, *et al.*, “A comparative study of different static and dynamic load balancing algorithm in cloud computing with special emphasis on time factor,” *Int. J. Curr. Eng. Technol*, vol. 5, no. 3, pp. 2277–4106, 2015.

[34] I. A. Mohialdeen, “Comparative study of scheduling al-gorithms in cloud computing environment,” *Journal of Computer Science*, vol. 9, no. 2, pp. 252–263, 2013.

[35] M. Maheswaran, S. Ali, H. J. Siegel, *et al.*, “Dynamic mapping of a class of independent tasks onto heterogeneous computing systems,” *Journal of parallel and distributed computing*, vol. 59, no. 2, pp. 107–131, 1999.

[36] L. Lenzini, E. Mingozzi, and G. Stea, “Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers,” *IEEE/ACM Transactions on Networking (TON)*, vol. 12, no. 4, pp. 681–693, 2004.

[37] H. Chen, F. Wang, N. Helian, and G. Akanmu, “User-priority guided min-min scheduling algorithm for load balancing in cloud computing,” in *National conference on parallel computing technologies (PARCOMPTECH) (PARCOMPTECH)*. IEEE, 2013, pp. 1–8.

[38] B. Li, Y. Pei, H. Wu, and B. Shen, “Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds,” *The Journal of Supercomputing*, vol. 71, no. 8, pp. 3009–3036, 2015.

[39] S. Taherian Dehkordi and V. Khatibi Bardsiri, “Tasa: a new task scheduling algorithm in cloud computing,” *Journal of Adv. in Comp. Eng. and Tech.*, vol. 1, no. 4, pp. 25–32, 2015.

[40] F. Wu, Q. Wu, and Y. Tan, “Workflow scheduling in cloud: a survey,” *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3373–3418, 2015.

[41] I. De Falco, U. Scafuri, and E. Tarantino, “Two new fast heuristics for mapping parallel applications on cloud computing,” *Future Generation Computer Systems*, vol. 100, no. 37, pp. 1–13, 2014.

[42] N. Soltani, B. Soleimani, and B. Barekatain, “Heuristic algorithms for task scheduling in cloud computing: a survey,” *International Journal of Computer Network and Information Security*, vol. 9, no. 8, p. 16, 2017.

[43] K. Gupta and V. Katiyar, “Survey of resource provisioning heuristics in cloud and their parameters,” *International Journal of Computational Intelligence Research*, vol. 13, no. 5, pp. 1283–1300, 2017.

[44] 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.

[45] S. K. Panda and P. K. Jana, “Sla-based task scheduling algorithms for heterogeneous multi-cloud environment,” *The Journal of Supercomputing*, vol. 73, no. 6, pp. 2730–2762, 2017.

[46] X. Ye, S. Liu, Y. Yin, Y. Jin, “User-oriented many-objective cloud workflow scheduling based on an improved knee point driven evolutionaryalgorithm,” *Knowledge-Based Systems*, vol. 135, no. C, pp. 113–124, 2017.

[47] J. A. J. Sujana, T. Revathi, *et al.*, "Smart pso-based secured scheduling approaches for scientific workflows in cloud computing," *Soft Computing*, vol. 23, no. 5, pp. 1745–1765, 2019.

[48] M. R. Bouzidi, A. Soltani, A. Bouhank, and M. Daoudi, "New search based methods to solve workflow scheduling problem in cloud computing," in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT).* IEEE, 2018, pp. 647–652.

[49] H.-L. Chan, J. W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. Wong, "Optimizing throughput and energy in online deadline scheduling," *ACM Transactions on Algorithms (TALG)*, vol. 6, no. 1, p. 10, 2009.

[50] 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.

[51] X. Ye, Y. Yin, and L. Lan, "Energy-efficient many-objective virtual machine placement optimization in a cloud computing environment," *IEEE Access*, vol. 5, pp. 16 006–16 020, 2017.

[52] 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. 75 033–75 047, 2018.

[53] M. Duggan, J. Duggan, E. Howley, and E. Barrett, "An autonomous network aware vm migration strategy in cloud data centres," in *Cloud and Autonomic Computing (ICCAC), 2016 International Conference on.* IEEE, 2016, pp. 24–32.

[54] W. Dargie, "Estimation of the cost of vm migration," in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on.* Citeseer, 2014, pp. 1–8.

[55] G. Zhang, X. Zhu, W. Bao, D. Tan, H. Yan, and J. Chen, "A novel server consolidation method based on local storage integrated with resource demand

prediction," in *2018 15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)*. IEEE, 2018, pp. 113–120.

[56] G. Zhang, X. Zhu, W. Bao, H. Yan, and D. Tan, "Local storage-based consolidation with resource demand prediction and live migration in clouds," *IEEE Access*, vol. 6, pp. 26 854–26 865, 2018.

[57] A. Zareie, M. Pedram, M. Kelarestaghi, and A. Kosari, "An approach to mapping scientific workflow in cloud computing data centers to minimize costs of workflow execution," *Int. J. Comp. Tech. Appl*, vol. 2, no. 5, pp. 1627–1640, 2011.

[58] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE transactions on Cloud Computing*, vol. 2, no. 2, pp. 222–235, 2014.

[59] A. Khiyaita, H. El Bakkali, M. Zbakh, and D. El Kettani, "Load balancing cloud computing: state of art," in *Network Security and Systems (JNS2), 2012 National Days of*. IEEE, 2012, pp. 106–109.

[60] H. Khazaei, J. Misic, and V. B. Misic, "Performance analysis of cloud computing centers using m/g/m/m+ r queuing systems," *IEEE Transactions on parallel and distributed systems*, vol. 23, no. 5, pp. 936–943, 2011.

[61] 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.

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

[63] S. Makonin, *et al.*, "Rae: The rainforest automation energy dataset for smart grid meter data analysis," *Data*, vol. 3, no. 1, p. 8, 2018.

[64] A. G. Delavar and Y. Aryan, "Hsga: a hybrid heuristic algorithm for workflow scheduling in cloud systems," *Cluster computing*, vol. 17, no. 1, pp. 129–137, 2014.

[65] B. Hayes, “Cloud computing,” *Communications of the ACM*, vol. 51, no. 7, pp. 9–11, 2008.

[66] B. P. Rimal, E. Choi, and I. Lumb, “A taxonomy and survey of cloud computing systems,” in *2009 Fifth International Joint Conference on INC, IMS and IDC.* IEEE, 2009, pp. 44–51.

[67] A. Beloglazov and R. Buyya, “Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1366–1379, 2013.

[68] C. S. Yeo and R. Buyya, “Service level agreement based allocation of cluster resources: Handling penalty to enhance utility,” in *2005 IEEE International Conference on Cluster Computing.* IEEE, 2005, pp. 1–10.

[69] F. Durao, J. F. S. Carvalho, A. Fonseka, and V. C. Garcia, “A systematic review on cloud computing,” *The Journal of Supercomputing*, vol. 68, no. 3, pp. 1321–1346, 2014.

[70] S. Groot, “Research on efficient resource utilization in data intensive distributed systems,” Ph.D. dissertation, University of Tokyo, 2013.

[71] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, and R. Katz, “Multi-task learning for straggler avoiding predictive job scheduling,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 3692–3728, 2016.

[72] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, “Effective straggler mitigation: Attack of the clones,” in *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 2013, pp. 185–198.

[73] R. W. Ahmad, A. Gani, S. H. A. Hamid, M. Shiraz, A. Yousafzai, and F. Xia, “A survey on virtual machine migration and server consolidation frameworks for cloud data centers,” *Journal of network and computer applications*, vol. 52, no. C, pp. 11–25, 2015.

[74] H. Jin, W. Gao, S. Wu, X. Shi, X. Wu, and F. Zhou, "Optimizing the live migration of virtual machine by cpu scheduling," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1088–1096, 2011.

[75] F. Ramezani, J. Lu, and F. K. Hussain, "Task-based system load balancing in cloud computing using particle swarm optimization," *International journal of parallel programming*, vol. 42, no. 5, pp. 739–754, 2014.

[76] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the migration of virtual computers," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 377–390, 2002.

[77] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg, "Live wide-area migration of virtual machines including local persistent state," in *Proceedings of the 3rd international conference on Virtual execution environments.* ACM, 2007, pp. 169–179.

[78] M. Nelson, B.-H. Lim, *et al.*, "Fast transparent migration for virtual machines." in *USENIX Annual technical conference*, 2005, pp. 391–394.

[79] A. Tchernykh, L. Lozano, *et al.*, "Online bi-objective scheduling for iaas clouds ensuring quality of service," *Journal of Grid Computing*, vol. 14, no. 1, pp. 5–22, 2016.

[80] P. P. Ray, "The green grid saga-a green initiative to data centers: a review," *I. Journal of Comp. Sc. and Eng.*, vol. 1, no. 4, pp. 335–339, 2012.

[81] D. Fernández-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, 1989.

[82] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of ACM symposium on theory of computing.* ACM, 1971, pp. 151–158.

[83] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems," *SIAM Journal on computing*, vol. 18, no. 1, pp. 186–208, 1989.

[84] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM (JACM)*, vol. 24, no. 2, pp. 280–289, 1977.

[85] S. K. Garg, A. N. Toosi, *et al.*, "Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter," *Journal of Network and Computer Applications*, vol. 45, pp. 108–120, 2014.

[86] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[87] A. A. S. Farrag, S. A. Mahmoud, and M. El Sayed, "Intelligent cloud algorithms for load balancing problems: A survey," in *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS). IEEE*, 2015, p. 210216.

[88] Z. Liu and S. Cho, "Characterizing machines and workloads on a google cluster," in *International Conference on Parallel Processing Workshops (ICPPW).* IEEE, 2012, pp. 397–403.

[89] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in google cloud to derive realistic resource utilization models," in *International Symposium on Service Oriented System Engineering (SOSE).* IEEE, 2013, pp. 49–60.

[90] 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.

[91] C. Reiss, A. Tumanov, *et al.*, "Towards understanding heterogeneous clouds at scale: Google trace analysis," *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, vol. 84, 2012.

[92] 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. IEEE Computer Society, 2010, pp. 94–103.

[93] C. Liu, C. Liu, Y. Shang, *et al.*, "An adaptive prediction approach based on workload pattern discrimination in the cloud," *Journal of Network and Computer Applications*, vol. 100, no. 80, pp. 35–44, 2017.

[94] A. Hussain and M. Aleem, "Gocj: Google cloud jobs dataset for distributed and cloud computing infrastructures." Mendeley, Sep 2018, accessed on 30-04-2019. [Online]. Available: https://data.mendeley.com/datasets/b7bp6xhrcd/1

[95] "Hcsp: Heterogeneous computing scheduling problems," accessed on 20-03-2019. [Online]. Available: https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/HCSP_inst.htm

[96] A. Hussain, M. Aleem, M. A. Iqbal, and M. A. Islam, "Sla-ralba: cost-efficient and resource-aware load balancing algorithm for cloud computing," *The Journal of Supercomputing*, vol. 75, no. 10, pp. 6777–6803, 2019.

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

[98] S. Ali, H. J. Siegel, *et al.*, "Task execution time modeling for heterogeneous computing systems," in proceedings of *Heterogeneous Computing Workshop,(HCW 2000).* IEEE, 2000, pp. 185–199.

[99] "Grid workload archiver tudelft (gwa-t) traces." accessed on 15-03-2019. [Online]. Available: http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains

[100] "Facebook hadoop workload." accessed on 20-03-2019. [Online]. Available: https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository

[101] "Google cluster traces." accessed on 20-03-2019. [Online]. Available: https://github.com/google/cluster-data

[102] "Yahoo cluster traces." accessed on 20-03-2019. [Online]. Available: https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&guccounter=1

[103] "Opencloud hadoop workload." accessed on 15-03-2019. [Online]. Available: http://ftp.pdl.cmu.edu/pub/datasets/hla/

[104] "Eucalyptus iaas cloudworkload." accessed on 15-03-2019. [Online]. Available: https://www.cs.ucsb.edu/~rich/workload/

[105] T. C. Hung and N. X. Phi, "Study the effect of parameters to load balancing in cloud computing," *International Journal of Computer Networks and Communications (IJCNC)*, vol. 8, no. 3, pp. 33–45, 2016.

[106] G. Sharma and P. Banga, "Task aware switcher scheduling for batch mode mapping in computational grid environment," *International Journal*, vol. 3, no. 6, 2013.

[107] N. A. Mehdi, A. Mamat, H. Ibrahim, and S. K. Subramaniam, "Impatient task mapping in elastic cloud using genetic algorithm," *Journal of Computer Science*, vol. 7, no. 6, p. 877, 2011.

[108] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar, "Cost-efficient and application sla-aware client side request scheduling in an infrastructure-as-a-service cloud," in *2012 IEEE Fifth International Conference on Cloud Computing.* IEEE, 2012, pp. 213–220.

[109] M. Alrokayan, A. V. Dastjerdi, and R. Buyya, "Sla-aware provisioning and scheduling of cloud resources for big data analytics," in *Cloud Computing in Emerging Markets (CCEM), 2014 IEEE International Conference on.* IEEE, 2014, pp. 1–8.

[110] R. F. Freund, M. Gherrity *et al.*, "Scheduling resources in multi-user, heterogeneous, computing environments with smartnet," in *Proceedings Heterogeneous Computing Workshop (HCW'98).* IEEE, 1998, pp. 184–199.

[111] S. S. Chauhan and R. Joshi, "Qos guided heuristic algorithms for grid task scheduling," *Int. Journal of Computer App.*, vol. 2, no. 9, pp. 24–31, 2010.

[112] N. Mehdi, A. Mamat, H. Ibrahim, and S. Symban, "Virtual machines cooperation for impatient jobs under cloud paradigm," *International Journal of Information and Communication Engineering*, vol. 7, no. 1, pp. 13–19, 2011.

[113] S. Behzad, R. Fotohi, and M. Effatparvar, "Queue based job scheduling algorithm for cloud computing," *International Research Journal of Applied and Basic Sciences*, vol. 4, no. 11, pp. 3785–3790, 2013.

[114] L. Tripathy and R. R. Patra, "Scheduling in cloud computing," *Int. Journal on Cloud Computing Ser. and Arch. (IJCCSA)*, vol. 4, no. 5, pp. 21–7, 2014.

[115] S. Raychaudhuri, "Introduction to monte carlo simulation," in *Simulation Conference, WSC 2008. Winter.* IEEE, 2008, pp. 91–100.

[116] Y. Cheng and G. Xu, "A novel task provisioning approach fusing reinforcement learning for big data," *IEEE Access*, vol. 7, pp. 143 699–143 709, 2019.

[117] U. Ahmed, M. Aleem, *et al.*, "Ralb-hc: A resource-aware load balancer for heterogeneous cluster," *Concurrency and Computation: Practice and Experience*, p. e5606.

[118] M. Ibrahim, S. Nabi, *et al.*, "Toward a task and resource aware task scheduling in cloud computing: An experimental comparative evaluation," *International Journal of Networked and Distributed Computing*, 2020.

[119] R. Gulbaz, "Task scheduling optimization in cloud computing," M.S. dissertation, CUST, Islamabad, Department of Computer Science, 2020.

[120] A. Sajjad, "Performance analysis of scheduling schemes for cloud computing resources," M.S. dissertation, CUST, Islamabad, Department of Computer Science, 2020.

[121] L. Al Shalabi, Z. Shaaban, and B. Kasasbeh, "Data mining: A preprocessing engine," *Journal of Computer Science*, vol. 2, no. 9, pp. 735–739, 2006.

[122] X.-Y. Shao, Z.-H. Wang *et al.*, "Integrating data mining and rough set for customer group-based discovery of product configuration rules," *International Journal of Production Research*, vol. 44, no. 14, pp. 2789–2811, 2006.