CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Performance Analysis of Scheduling Schemes for Cloud Computing Resources

by

Ammara Sajjad

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing

Department of Computer Science

2020

This thesis work is wholeheartedly dedicated to my beloved parents, sister, brothers, and my best friends who have been my source of inspiration and gave me strength when I thought of giving up, who continually provide their moral, emotional, and financial support. A special feeling of gratitude to my kind supervisor and loving parents for their love, endless support and encouragement.

CERTIFICATE OF APPROVAL

Performance Analysis of Scheduling Schemes for Cloud Computing Resources

by

Ammara Sajjad

(MCS173025)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Muhammad Aleem	FAST, Islamabad
(b)	Internal Examiner	Dr. Masroor Ahmed	CUST, Islamabad
(c)	Supervisor	Dr. Muhammad Abdul Qadir	CUST, Islamabad

Dr. Muhammad Abdul Qadir
Thesis Supervisor
June, 2020

Dr. Nayyer Masood
Head
Dept. of Computer Science
June, 2020

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
June, 2020

Author's Declaration

I, **Ammara Sajjad** hereby state that my MS thesis titled "**Performance Analysis of Scheduling Schemes for Cloud Computing Resources**" 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/abroad.

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

(Ammara Sajjad)
Registration No: MCS173025

Plagiarism Undertaking

I solemnly declare that research work presented in this thesis titled "**Performance Analysis of Scheduling Schemes for Cloud Computing Resources**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been dully 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 MS Degree, the University reserves the right to withdraw/revoke my MS 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 work.

(Ammara Sajjad)
Registration No: MCS173025

Acknowledgements

In the Name of Allah, the Most Merciful, the Most Compassionate all praise be to Allah, the Lord of the worlds; and prayers and peace be upon Mohammad His servant and messenger. First and foremost, I must acknowledge my limitless thanks to Allah, the Ever-Magnificent; the Ever-Thankful, for His help and blessings.I am totally sure that this work would have never become truth, without His guidance.

I am grateful to some people, who worked hard with me from the beginning till the completion of the present research particularly my supervisor **Dr. Muhammad Abdul Qadir**, who has been always generous during all phases of the research.

I would like to take this opportunity to say warm thanks to all my beloved friends, who have been so supportive along the way of doing my thesis.

Last but not least, I would like to express my wholehearted thanks to my family for their generous support they provided me throughout my entire life and particularly through the process of pursuing the master degree. Because of their unconditional love and prayers, I have the chance to complete this thesis.

(Ammara Sajjad)
Registration No: MCS173025

Abstract

Scheduling jobs to available resources (VMs) in the cloud is a crucial process in order to maximize the performance (minimized makespan, higher throughput, and increased resource utilization) with a minimum cost. Users of a cloud system need a fair comparison of different scheduling schemes in order to make decisions for scheduling their jobs. After carefully evaluating the existing comparison between different scheduling schemes, it was observed that the performance evaluation by the developers of scheduling schemes have been done by using either a dataset which favors their scheme or by using partial definitions of the performance parameters, like, throughput or resource utilization. This demands an independent evaluation of the scheduling schemes by using fair datasets, and with thorough and complete definition of the performance parameters. In this research, we evaluate the performance of eleven popular static scheduling schemes with thorough and complete definitions of the performance parameters which covers all the aspects including size of tasks and VMs of non-uniform computational power. These schemes are evaluated by using both a third-party dataset Google Cloud Jobs dataset (GoCJ) and our own prepared dataset named as Random Data Set (RanDS) to evaluate the worst-case scenario of scheduling schemes. In this analysis, we assume a job pool of independent tasks which lead to a non-pre-emptive scheduling, in which the tasks are scheduled statically till its completion. The analysis is a simulation-based. The simulations have been conducted on a renowned cloud simulator, CloudSim, using GoCJ dataset and RanDS. As per our results, RALBA has the lowest makespan, highest throughput, and resource utilization while RS showed the highest makespan, lowest throughput, and resource utilization by using the GoCJ dataset. With RanDS dataset, RALBA, RASA, and Max-Min showed the same amount of lowest makespan, highest throughput, and resource utilization. One significant result observed that the maximum resource utilization was 98% on the GoCJ dataset and 36.4% by using RanDS for the best performing scheme, which proves that still there is room to devise a scheme with improved resource utilization.

Contents

List of Figures

List of Tables

Abbreviations

AMS	Advanced MaxSufferage
ARUR	Average Resource Utilization Ratio
ELBMM	Enhanced Load Balanced Min-Min
FCFS	First Come First Serve
FIFO	First In First Out
GoCJ	Google Cloud Jobs
GA	Genetic Algorithm
GSA	Genetic Simulated Annealing
HCSP	Heterogeneous Computing Scheduling Problems
HPC2N	High Performance Computing center North
IaaS	Infrastructure as a Service
KPB	K-Percent Best
LIGO	Laser Interferometer Gravitational Wave Observatory
LBMM	Load Balancing Min-Min
LBIMM	Load Balancing Improved Min-Min
LBTSA	Load Balancing objective Task Scheduling Algorithm
MCT	Minimum Completion Time
MET	Minimum Execution Time
MT	Meta-Task
MIs	Million Instructions
MIPS	Million Instructions Per Second
MOTSA	Multi-Objective Task Scheduling Algorithm
MPTSA	Multilevel Priority-Based objective Task Scheduling Algorithm
OLB	Opportunistic Load Balancing

PMs	Physical Machines
PSSLB	Proactive Simulation-Based Scheduling and Load Balancing
PSSELB	Proactive Simulation-Based Scheduling and Enhanced Load sBalancing
PA-LBIMM	Priority-Aware Load Balancing Improved Min-Min
QoS	Quality of Service
RS	Random Selection
RR	Round Robin
RALBA	Resource-Aware Load Balancing Algorithm
RASA	Resource-Aware Scheduling Algorithm
RanDS	Random Dataset
RM	Resource Manager
SJF	Shortest Job First
SA	Switching Algorithm
SLA	Service Level Agreement
TASA	Task-Aware Scheduling Algorithm
TS-GA	Tournament Selection Genetic Algorithm
VM	Virtual Machine
VMM	Virtual Machine Moniter

Symbols

CLS	Set of all cloudlets (to be scheduled)
$Cloudleti_i.MI$	Size of $Cloudlet_i$ in MI
T_p	Total Computational Power
U_p	Utilized Computational Power
I_p	Idle Computational Power
VMS	Set of VMs in a data-center
$vmCrMap$	Set of sorted VMs with its computing ratio
$vmCrMap_j$	Computation Ratio of VM_j in vmCrMap
$RPCloudlet_j$	Set of remaining possible cloudlets that can be assigned to VM_j
$maxPCloudletVM_j$	Largest cloudlet in $RPCloudlet_j$ that is assigned to VM_j
$VMShare$	Set of sorted VMs with its computing share
$VMShare_j$	Computing share of VM_j (in MI) in VMShare
$CloudletCT_{ij}$	Expected completion time of $Cloudlet_i$ on VM_j
$VMCT_j$	Completion time of VM_j
$CloudletEFT_i$	Earliest finish time of $Cloudlet_i$
$Fill\_Scheduler$	Sub-scheduler that assigns $Cloudlet_i$ to VM_j based on $VMShare_j$
$Spill\_Scheduler$	Sub-scheduler that assigns $Cloudlet_i$ to VM_j based on $CloudletEFT_i$
$VM_j.MIPS$	Computing power of VM_j in terms of Million Instructions Per Second (MIPS)

Chapter 1

Introduction

1.1 Background

"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"[10].The main intention of cloud computing is to provide virtualization of resources to be accessed remotely [11]. Most prominent cloud computing platforms are Google App Engine [12], GoGrid, Aneka [11], Microsoft Azure [13], and, Amazon EC2 [14] for managing, monitoring, provisioning resources, and application services. In cloud computing, the virtualization of a service means the combination of registered computing resources collected in a virtual environment called Virtual Machines (VMs). The Physical Machine (PM) resources are available to the cloud user through virtualization in the form of VMs. A VM emulates a specific computer system and executes tasks provided by the user. The concept of scheduling in cloud computing refers to the method of assigning a set of jobs to a set of VMs or allocating VMs to the available resources (PMs) to meet user demands [15]. Schedulers which perform scheduling activity, are often implemented to optimally utilize the resources in a load-balanced way, allow multiple users to effectively share system resources

[16]. The problem of scheduling tasks to computing resources has a significant impact on system performance [17]. With an optimized scheduling scheme, one can enjoy the benefits of minimized makespan and maximized throughput. Thus, the important scheduling objective is to optimize resource utilization.

1.2 Task Scheduling Model

Scheduling is a non-deterministic polynomial-time hard (NP-hard) problem [18]. Normally, it is difficult to develop an optimal scheduler for producing optimal solutions in a reasonable time, because there are many VMs in a cloud and many user tasks to be scheduled with different scheduling objectives [15]. Thus, applying meta heuristics is important which gives near-optimal solutions within a reasonable amount of time [18]. The tasks (cloudlets) are assumed to be independent, scheduled statically till its completion without interruption (non-preemptive), without any inter-task communication or migration, with the same priority, and no deadlines. The VMs are assumed to be heterogeneous with varying computational power.

A cloud contains a set of heterogeneous VMs which are responsible for the execution of tasks, represented as $V = VM_1, VM_2, VM_3, ..., VM_m$, where m denotes the total heterogeneous VMs and a particular VM may be shown as VM_j. The set of n tasks are presented as $T = T_1, T_2, T_3, ..., T_n$, and a specific task can be represented as T_i. Each task T_i is characterized by e_{ij} where e_{ij} is the execution time of the task T_i on VM_j and is provided in a utilization matrix C. The column number in matrix C indicates the number of VMs and the number of rows shows the number of tasks. In other words, the matrix order is $n \times m$ and the elements of matrix C are real numbers, which specify the maximum completion time of the VMs used for the tasks.

The problem of task scheduling can now be formally defined as: In accordance with T and V, create a schedule that allocates different VM in V for each task in T, so that the utilization of the tasks at any VM does not exceed than the

utilization bound (makespan) of that VM. The total time of all the scheduled task on a VM_j is termed as the utilization of that VM named as UVM_j and if we multiply the computational power of that VM with UVM_j that can be termed as Computational Power Utilization named as PU_j. The difference between the maximum makespan and UVM_j, multiplied with the computational power of VM_j is termed as Idle Computational Power named as I_p of that VM. The Sum of the idle computational power of all the VMs can be termed as the idle computational power of the scheduler. The goal of a good scheduling scheme is to minimize idle computational power.

Scheduling in cloud computing can be done at two different levels: [15] VM-level and Host-level as shown in Figure 1.1

(i) At the Host-level, VM scheduler is used for the allocation of virtual resources on physical machines in the cloud datacenters known as VM scheduling.

(ii) At the VM-level, tasks are mapped for execution to the allocated VMs using task scheduler submitted by cloud users on virtual resources known as task scheduling.

Assume the cloud platform is supported by PMs: $PM_1, PM_2, PM_3, ..., PM_p$, each of which hosts a set of VMs: $V = VM_1, VM_2, VM_3, ..., VM_m$, through the corresponding VMM and each PM is managed by RM who tracks the resource utilization of PMs and gathers runtime statistics from every PM, including PM resource utilization, availability, and status of the VM. There are N independent tasks; $T_1, T_2, T_3, ..., T_n$, to be scheduled as shown in Figure 1.1. The user submits their tasks in the first stage, after which the Task Scheduler collects input from RMs to ensure an overall efficiency of the resources. The Task scheduler, which takes into account the user requests received from the service provider and information received from the RMs, initiates a task algorithm for the scheduling of tasks for VMs and VM schedulers to schedule VMs for PMs.

This study focusses on Cloudlet-VM scheduling. The task scheduling is the process of mapping cloudlets on several available computing resources (VMs) [19],[20].

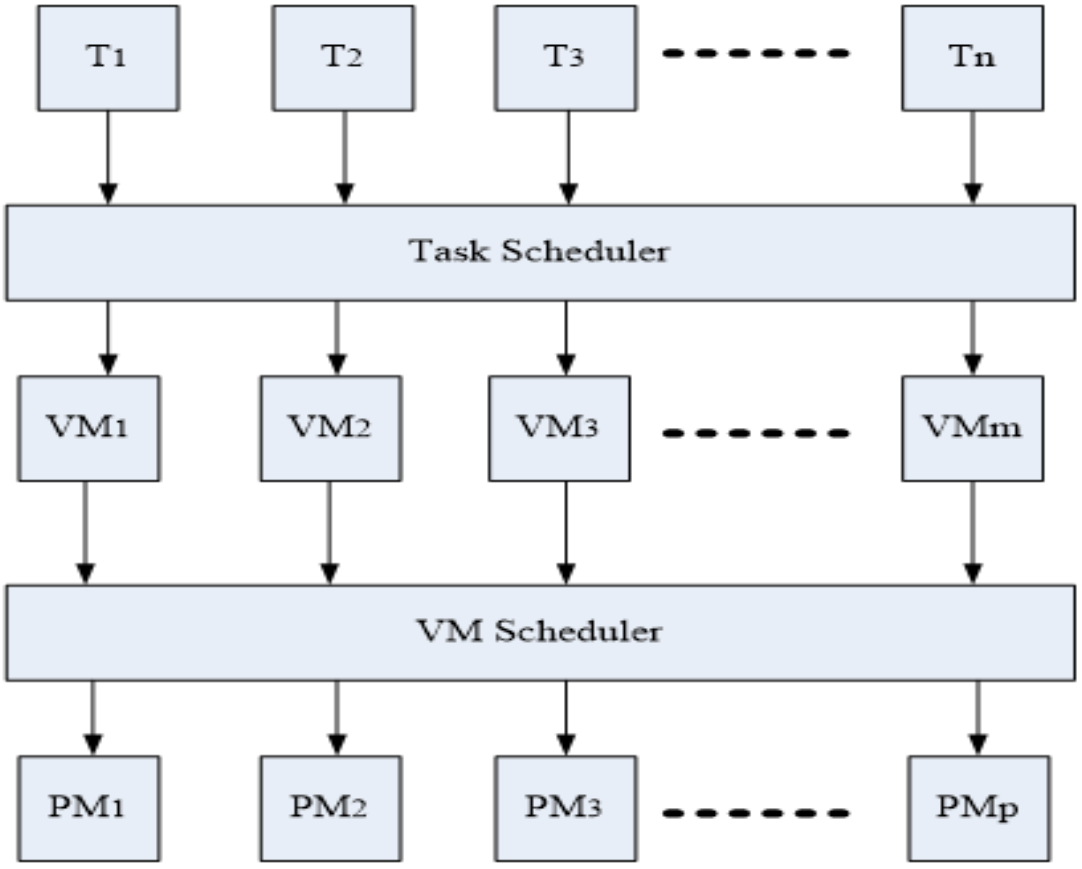

FIGURE 1.1: Scheduling of Tasks and VMs in cloud computing

Effective task scheduling mainly reduces the completion time of the task, increases throughput, and maximizes resource utilization. Generally, the scheduling schemes are categorized as static and dynamic. Static scheduling schemes provide a complete map of the task or job before it is executed, while a dynamic scheduling technique usually relies on runtime parameters to schedule tasks [21].

In this thesis, three performance measures are used to compare different scheduling schemes such as: makespan, throughput, and Resource Utilization.

Makespan is the maximum time for the completion of all the cloudlets (tasks) in a workload by the available resources [5].
Throughput is referred as the total size of all cloudlets (MIs) completed per unit time.

Resource Utilization is a measure of busyness of resources during a scheduling.It is accomplished by reducing the idle time of a resource.

1.3 A Motivational Scheduling Scenario

Consider we have 5 VMs with different computational power (i.e. 100, 500, 1000, 1500, and 2000 MIPS) and 10 cloudlets with different length (i.e. 9.3, 11.1, 11.3, 11.3, 12.5, 13.5, 33.75, 52.5, 71.25, and 90 MIs). We have tried manually all possible combinations of scheduling then build an ideal scheduler for this scenario. And then these 10 cloudlets are scheduled on 5 VMs by using RALBA and RASA algorithms through simulation because these two algorithms have outperformed than other algorithms that's why we have compared these two algorithms with an ideal scheduling scenario as shown in Figure 1.2. An ideal scheduler has achieved 665.25 sec makespan, 4758 MIs/sec throughput, and 93.2% resource utilization. While, RALBA and RASA have achieved 712.5 and 706 sec makespan, 4442 and 4483 MIs/sec throughput, 87.1 and 87.9 % resource utilization. So, an ideal scheduler is 7% better than RALBA and 6% better than RASA as shown in Figure 1.2. This indicates that there is a need to critically evaluate the existing state of the art scheduling schemes and to come up with a scheduling scheme which is closer to an ideal scheduler and also its complexity should not be exponential.

1.4 Scope

We critically evaluate the performance measures as makespan, throughput, and resource utilization and then eleven prominent static scheduling schemes including; FCFS [8], RS [7], SJF [9], OLB [6], MCT [1], Min-Min [2], Max-Min [2], TASA [4], RASA [3], Sufferage [2], and RALBA [5] are evaluated with thorough and complete definitions of three performance measures as makespan, throughput, and resource utilization which covers all aspects including computation power of VMs and size of tasks. These schemes are evaluated by using both a third-party dataset Google

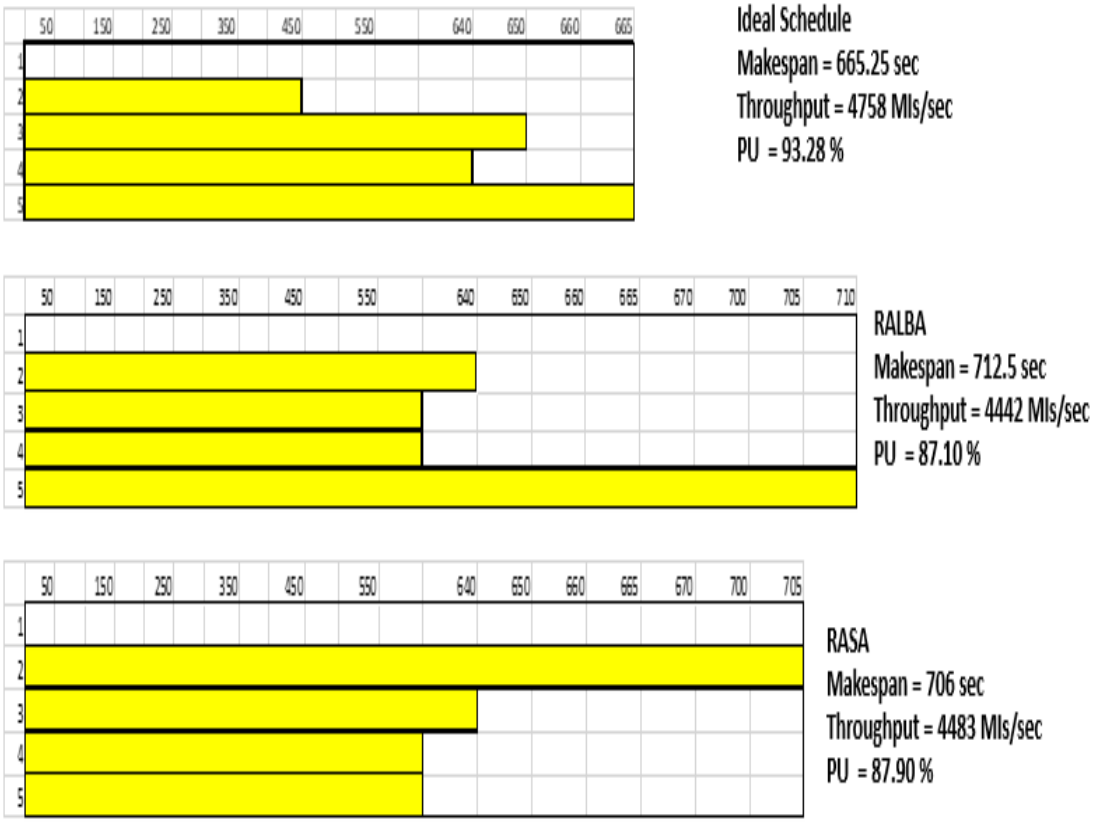

FIGURE 1.2: Comparison with an Ideal Scheduler

Cloud Jobs dataset (GoCJ) and our own prepared dataset named as Random Data Set (RanDS) to evaluate the worst-case scenario of scheduling schemes.

1.5 Problem Statement

After carefully evaluating the existing comparison between different scheduling schemes, it was observed that the performance evaluation by the developers of scheduling schemes have been done by using either self-created synthetic dataset instead of any third-party popular dataset or by using partial definitions of the performance measures, like throughput or resource utilization. It has also been observed that the scheduling schemes have not been evaluated with the worst-case scenario. And also, the comparison has not been done with an ideal scheduler.

1.6 Research Questions

We have formulated the following research questions relying on the problem statement describe above:

1. What are the thorough and complete definitions of different performance measures for a scheduling scheme?
2. What is the resource utilization and throughput of each scheduling scheme by using thorough and complete definitions of these performance measures?
3. Can we find or formulate a dataset very close to the real-life scenario to see worst-case performance against the parameters for which the developers of the scheme has claimed an excellent performance?
4. How a comparison with an ideal scheduling scenarios for a smaller dataset can be prepared?

1.7 Research Methodology

This section contains a brief description of the proposed methodology. The methodology consists of the following steps:

Step 1: Literature Review

In step 1, the literature review is conducted and categorized into two categories. In category 1, the working, strengths, weaknesses, and time complexities of task scheduling schemes are described in detail. In category 2, related survey papers and related studies based on selected scheduling schemes and scheduling objectives are discussed.

Step 2: Review of Evaluation Metrics

In step 2, performance metrics are reviewed. It was found that the definition of throughput does not truly reflect when we have the different size of tasks and similarly, the definition of resource utilization doest not truly reflect when we have

different computational power of VMs. After reviewing performance metrics from the literature, two new performance metrics are thoroughly defined which covers all aspects including size of the tasks and computational power of VMs.

Step 3: Dataset Collection

In this research, we evaluate the performance of eleven popular scheduling schemes by using both a third-party dataset named as Google Cloud Jobs dataset (GoCJ) and our own prepared dataset named as Random Data Set (RanDS) which is acquired from an internet source and modified based on certain parameters. The details of these datasets are described in Chapter 4.

Step 4: Simulation and Results

This study is simulation-based. In step 4, the CloudSim simulator is integrated with eclipse IDE for simulation. The scheduling schemes are implemented in java using CloudSim toolkit. Then the simulation results of all scheduling schemes are evaluated on the basis of makespan, throughput, and resource utilization that are discussed in Chapter 5

Step 5: Analysis and Discussion

In step 5, comparative analysis of eleven selected scheduling schemes is conducted in different scenarios on the basis of three performance measures as makespan, throughput, and resource utilization. The detailed discussion is given in Chapter 6.

Step 6: Conclusion and Future Direction

In step 6, the conclusion is drawn by keeping in mind the experimental results and comparative analysis. Some potential future directions are highlighted that could assist the scientific community for the effective task scheduling mechanism.

1.8 Thesis Organization

The remainder of the study is formulated as follows:

Chapter 2- provides the literature review in which the working of selected scheduling schemes and related work are discussed.

Chapter 3- provides the thorough definitions of performance measures (i.e. throughput and resource utilization) which covers all the aspects.

Chapter 4- illustrates the dataset and workload compositions.

Chapter 5- demonstrates the experimental setup and simulation results.

Chapter 6- presents the comparative discussion between state-of-the-art scheduling schemes.

Chapter 7- concludes this thesis along with future directions.

Chapter 2

Literature Review

The literature review is categorized into two categories; category one describes the working of static task scheduling schemes which we analyze on the basis of makespan, throughput, and resource utilization. In category two related survey papers and studies which are based on selected scheduling schemes are discussed.

2.1 Task Scheduling Schemes

In this thesis, we empirically analyze eleven prominent static task scheduling schemes in cloud computing which are: MCT [1], Min-Min [2], Max- Min [2], FCFS [8], SJF [9], RS [7], OLB [6], Sufferage [2], RASA [3], TASA [4], and RALBA [5]. The working of these algorithms is described as below:

2.1.1 MCT Algorithm [1]

Let r_j represents the ready time for a virtual machine m_j that will become ready for the execution of a task, and e_{ij} presents the execution time of task i on machine m_j. MCT algorithm first determines the expected completion time c_{ij} using the r_j and e_{ij} values at each scheduling interval. By searching the ith row of C matrix (consisting of the c_{ij} values) for each task t_i, determines the machine which

provides the earliest completion time. The task t_k is determined and allocated to the corresponding machine with the minimum expected time of completion. The task t_k is deleted from the meta-task which has just been mapped. At the end, the matrix C and vector r will be modified, and the above-mentioned procedure will be repeated for tasks not yet allocated to a machine. MCT examines the current load of VMs to identify a suitable VM for task scheduling [1]. The MCT algorithm has to search for all the available VMs at each scheduling interval, to determine the most appropriate VM for task scheduling that causes significant scheduling overhead. Therefore, it takes $O(M.N)$ time to map a given workload [5].

2.1.2 Min-Min Algorithm [2]

Let r_j represents the ready time for a virtual machine m_j that will become ready for the execution of a task and e_{ij} presents the execution time of i on machine m_j. Min-Min scheduling scheme first determines the expected completion time c_{ij} using the r_j and e_{ij} values at each scheduling interval and selects the minimum one. By searching the ith row of the C matrix (consisting of the c_{ij} values), determines the machine expected to has an earliest completion time for each task t_i. The task t_k having the minimum expected completion time is determined and allocated to the appropriate machine. The task t_k is deleted from the meta-task that has just been mapped. At the end, the matrix C and vector r will be will be modified, and the process described above will be repeated for tasks that were not yet allocated to a machine. If tasks t_i and t_k compete for a similar virtual machine m_j, then Min-Min allocates machine m_j to the task which has less ready time on m_j.

From Line (1) to Line (3) of Figure **??** the initialization of the c matrix takes $O(M.N)$ time: internal for loop runs M times (number of VMs) and external for loop runs N times (cloudlets). The Min-Min algorithm's **do** loop is repeated N times and every iteration takes $O(M.N))$ time, so it takes $O(M.N^2)$ time to schedule a given workload [2]. Min-Min algorithm initially schedules shorter jobs and then executing longer jobs [2][22][3]. Therefore, Min-Min mostly overloads the faster machines with a greater number of small jobs, while less but larger jobs

are allocated to the slower machines. Therefore, the larger jobs mapped on slower machines often cause a higher makespan for the execution of the workload [6].

2.1.3 Max-Min Algorithm [2]

The Max-Min scheduling scheme is same as Min-Min but the selection criteria of Max-Min varies from Min-Min. It varies from the Min-Min (as shown in Figure ??) in such a way, once a machine is identified which gives the earliest completion time for each task, the task t_k is determined with the earliest maximum completion time and then allocated to the most suitable virtual machine. That is, "minimum" will be changed to "maximum" in line (6) of Figure ??. The time complexity of Max-Min is the same as the complexity of Min-Min [2]. In such situations, the Max-min algorithm performs relatively better than the Min-Min algorithm, when the number of shorter tasks exceeds longer tasks [23]. For example, if the workload contains one long task only, then the Max-Min will run a number of shorter tasks simultaneously with the long ones. However, Min-Min first executes short tasks and then completes the longer tasks that leads to poor makespan compared to the Max-Min [24].

2.1.4 Sufferage Algorithm [2]

The Sufferage algorithm (shown in Appendix C) is built on the assumption that better mappings can be created through the assignment of the virtual machine to a task that would "suffer" most if it is not allocated to that particular machine based on the completion time. The Sufferage algorithm calculates sufferage value for every task. For the measurement of the sufferage value, the difference between the second-best minimum completion time and the earliest minimum completion time is calculated for every task in each scheduling interval. The task with the highest sufferage value is consequently allocated to the VM having the shortest completion time for that task.

The initialization process of Sufferage algorithm is the same as the Min-Min or Max-Min algorithms from lines (1) to (3), in Figure C1. All machines are initially labeled as unallocated. In every iteration of the **for** loop from Lines (6) to (14), a task t_k is arbitrarily selected from the meta-task. Find the virtual machine m_j for task t_k, which provides the earliest completion time and allocate m_j to t_k tentatively, if m_j is unallocated. Eliminate task t_k from meta-task and mark m_j as assigned. If, however, a task t_i has previously been allocated to a machine m_j, then from t_i and t_k, select the task which has maximum sufferage value, assign m_j to the selected task, and delete the selected task from the meta-task. In the implementation of the **for**loop, the unchosen task will never be considered again, it is taken into account for the next **do** loop iteration starting from Line (4). At the completion of **for** loop, the ready time of each machine is updated and next **do** loop iteration is executed until all tasks are scheduled. The first execution of the **for** loop takes $O(M.N)$ time as observed from the pseudo-code provided in Appendix C. The number of tasks assignment performed in one execution of the **for** loop depending upon the total number of VMs. Only one task will be allocated in each execution of the **for** loop, in the worst case. To schedule the entire meta-task, the outer **do** loop executes N iterations. Hence, the time $T(N)$ taken to schedule a meta-task of size N for the worst case will be [2]

$$T(N) = N.M + (N-1)M + (N-2)M + \cdots + M$$

$$T(N) = O(N^2 M) \tag{2.1}$$

There is an equal number of tasks and virtual machines in the best case, and there will be no contention between the tasks. Then, in the first execution of the **for** loop all the tasks are allocated, so $O(N.M)$ time will be taken by the Sufferage scheme in the best case scenario [2].

2.1.5 RASA [3]

RASA is presented in Appendix D, let r_j represents the ready time for a resource R_j to execute a task and E_{ij} presents the execution time to run task i on a resource R_j. RASA first determines the expected completion time C_{ij} using the R_j and E_{ij} values at each scheduling interval. If there are even numbers of resources, Max-Min technique is used for allocating the first task, on the other hand, the Min-Min technique is used. One of the two techniques is used alternatively, for the assignment of remaining tasks to the most suitable resources. For example, if the Max-Min strategy allocates the first task to a resource, then the next task will be allocated through the Min-Min. The machine that provides the earliest completion time is determined by searching the ith row of C matrix (consists of the C_{ij} values) for each task t_i. The task t_k with the maximum expected completion time is determined and allocated to the most appropriate resource. The task t_k is deleted from the meta-task that has been just mapped. At the end, the matrix C and vector r will be modified, and the above-mentioned procedure will be repeated alternatively for those tasks which were not yet allocated to a machine. Mostly, when considering smaller and larger jobs in alternative scheduling steps, RASA results in a lower makespan [3]. However, when the number of larger jobs is higher in the workload, RASA penalizes smaller jobs [25].

In Figure D1, the initialization of the C matrix takes $O(M.N)$ time: internal **for** loop runs M times (number of VMs) and external **for** loop runs N times (cloudlets). The **do** loop of the RASA algorithm is executed N times and every iteration takes $O(M.N)$ time. Therefore, the algorithm takes $O(M.N^2)$ time [3].

2.1.6 TASA [4]

In Appendix E, let r_j denotes the ready time for a resource R_j that becomes ready to execute a task and E_{ij} represents the execution time to run task i on a resource R_j. TASA first determines the expected completion time C_{ij} using the R_j and E_{ij} values at each scheduling interval. If there are even numbers of

resources then sufferage strategy is adopted to schedule tasks, otherwise the Min-Min technique is applied. When resources are even, then the machine which gives the earliest expected completion time is determined by scanning the i^th row of C matrix (consists of the C_{ij} values) for each task t_i. To measure the sufferage value, the earliest minimum completion time and the second-best minimum completion time is calculated for each job in each scheduling interval. The task t_k with the maximum sufferage value is determined and allocated to the corresponding VM which provides the earliest completion time. A task t_k is deleted from the meta-task that has been just mapped. At the end, the matrix C and vector r will be modified, and the same process will be repeated alternatively for those tasks that were not yet allocated to a machine. For most cases, TASA provides better makespan relative to other scheduling schemes including Min-Min, OLB, and Max-Min [4].

In Figure E1, the initialization of the C matrix takes $O(M.N)$ time: internal **for** loop runs M times (number of VMs) and external **for** loop runs N times (cloudlets). The **do** loop of the TASA algorithm is executed N times and each iteration takes $O(M.N)$ time. Therefore, the algorithm takes $O(M.N^2)$ time [4].

2.1.7 RALBA [5]

RALBA is a resource-aware load balancing algorithm. It consists of two sub modules: Spill scheduler and Fill scheduler as shown in Figure F1. Taking into account the computational share of virtual machines, Fill Scheduler performs Cloudle to VM allocation. Having $Largest\_VMShare$, Fill scheduler selects VM_j and determines $maxPCloudletVM_j$ for VM_j. The candidate cloudlet will be allocated to the VM and $VMShare_j$ of VM_j will be updated after the cloudlet allocation. The candidate cloudle is deleted from the cloudlet list on every scheduling decision and the computing share of specific VM is modified. Afterwards, RALBA moves to the second scheduler, Spill scheduler, to assign the remaining cloudlets. Spill scheduler performs allocation of $Cloudlets$ to VMs on the basis of EFT of $Cloudlets$. The $maxCloudlet$ is chosen and assigned to a particular virtual machine which

has EFT for that $maxCloudlet$. After assigning $Cloudlet$ to VM, the finish time of a specific VM will be modified. This mapping process is repeated until the scheduling of all the cloudlets has been done. The time complexity of the spill scheduler is $O(M.N)$ when N cloudlets are allocated by the spill scheduler. If n is the number of cloudlets allocated by Fill scheduler then Spill scheduler must schedule the remaining $N - n$ cloudlets. Hence, the time complexity of RALBA is $O(M^2n + M.N - n)$ [5].

2.1.8 OLB [6]

OLB assigns each task to a VM in an arbitrary order that will be ready next, regardless of the expected execution time of the task on that VM. If at the same time several machines are ready, one machine will be randomly selected [6]. To keep all VMs as busy as possible is the main scheduling objective of OLB scheduling scheme. However, due to the non-consideration of the current workload, it may result in poor makespan. In the worst case, all M machines may need to be examined by the scheduler to discover the machine that will be ready next. Therefore, OLB takes $O(M)$ time [6].

2.1.9 RS [7]

RS assigns cloudlets to VMs randomly, without taking into account the existing load of the VMs [26]. However, the biased scheduling process used by the RS may result in the selection of an overloaded VM that could result in low resource utilization, load imbalance and poor makespan [5]. RS has a simple implementation, low scheduling overhead and less time complexity relative to other scheduling schemes [7].

2.1.10 FCFS [8]

FCFS is the easiest First In First Out (FIFO) scheduling algorithm. This implies that the FCFS algorithm serves those tasks that arrive at the queue first. Once a task is finished, it is assigned to the next task in the queue. Then it eliminates the executing task from the list. All the tasks behind the queue must wait for a long time before finishing the long task in this scheme [8]. Although it is easy to implement, but it is poor in performance because it's average waiting time is higher than other scheduling schemes [23].

2.1.11 SJF [9]

In the shortest job first algorithm, the tasks are arranged according to their execution time in the ascending order and then executed in the same order. The tasks with the least execution time are executed first completely and then the scheduler is assigned to other tasks. The intuition behind SJF is to reduce the completion time [9]. In SJF, knowing or estimating the processing time of each task is a major problem. It can cause starvation for longer tasks if there are a large number of shorter tasks. SJF can improve throughput by ensuring that shorter tasks are executed first.

The brief overview of scheduling schemes is described in Table 2.1 along with their strengths, weaknesses, and time complexities.

TABLE 2.1: Strengths, Weaknesses, and Complexities of Scheduling Techniques

Algorithms	**Strengths**	**Weaknesses**	**Complexities**
RS [7]	Low Complexity than other algorithms	Poor resource utilization	$O(N)$
FCFS [8]	Avoid too much idle time of VM	Poor resource utilization	$O(N)$
SJF [9]	Relatively better makespan and throughput than RS and FCFS	Poor resource utilization	$O(N)$
OLB [6]	Low Complexity	Load-imbalance	$O(M)$
MCT [1]	Better makespan than RS,FCFS, SJF and OLB	Faster machines overloaded with more tasks Load-imbalance	$O(M.N)$
Min–Min [2]	Favors small sized tasks	Low Resource Utilization	$O(M.N^2))$
Max–Min [2]	Favors larger tasks	Load-imbalance for larger size tasks	$O(M.N^2)$
Sufferage [2]	Better makespan than RS, OLB, SJF, FCFS, MCT, Min–Min, Max–Min and TASA	Scheduling overhead due to the calculation of Sufferage-value	$O(M.N^2)$
RASA [3]	Fair mapping of smaller and longer tasks	Load imbalance	$O(M.N^2)$
TASA[4]	Better makespan and throughput than MCT, Min–Min, and Max–Min, Favors smaller tasks	Load-imbalance	$O(M.N^2)$

Algorithms	**Strengths**	**Weaknesses**	**Complexities**
RALBA [5]	Better makespan and throughput than other discussed schemes, Improved resource utilization	Load imbalance	$O(M^2.N)$

2.2 Survey Papers

A brief review of task scheduling schemes is provided in this section. Many researchers have proposed solutions to the problem of scheduling and allocation of resources. However, still, there is room to devise a scheme with improved resource utilization. Related work in the literature was examined, to perform comparative analysis.

2.2.1 Survey No 1

Hussain et al. [5] performed a comparative analysis of 9 static scheduling schemes. The scheduling schemes are evaluated using the CloudSim simulator on the basis of makespan, throughput and resource utilization by using 9 instances of HCSP and GoCJ datasets. But the method of calculating throughput and resource utilization in [5] partial, as they referred throughput as "number of tasks executed per second" this definition of throughput is correct for the tasks of nearly equal size. However, if the task size differs significantly, throughput can be defined as "length of tasks (MIs) executed per second". Empirical evaluation indicates that the ideal solution for scheduling independent tasks on machines is not based only on throughput, average resource utilization ratio and execution time; instead, load balancing at the machine level must also be taken into account in order to fully utilize the computing power in the cloud system [16]. Among all algorithms, RALBA gives

better results in terms of resource utilization [5]. On the other hand, the worst-case performance of RALBA against the parameters is not formulated in [5].

2.2.2 Survey No 2

Madni et al. [1] have compared task scheduling schemes including FCFS, Max-Min, Min-Min, Sufferage MCT, and, MET in the IaaS cloud computing system. The scheduling schemes are evaluated using the CloudSim simulator on the basis of cost, throughput, resource utilization, and makespan using the HPC2N dataset. In [1], partial definitions of throughput and resource utilization are used to evaluate the performance of scheduling schemes. However, results of [1] show that Min-Min outperforms all other schemes while MET always gives better performance in terms of resource utilization in the scheduling of the tasks in IaaS cloud computing [1]. But the worst-case performance of scheduling schemes against the parameters is also not formulated in [1].

2.2.3 Survey No 3

Ali and Alam [27] reviewed and compared task scheduling schemes on the basis of throughput, resource utilization, makespan, cost, QoS, and energy consumption. Most of the schemes discussed in [27] focused on energy consumption and service cost. The schemes compared in [27] are not analyzed using any third-party popular dataset, not using thorough and complete definitions of throughput and resource utilization and also, the methodology for analysis is not discussed.

2.2.4 Survey No 4

Maipan et al. [28] have proposed Max-Average algorithm and reviewed algorithms for task scheduling, including Max-Min, Min-Min, MET, MCT, and Max-Average using 4 instances of HCSP dataset. These algorithms were compared based on average resource utilization and makespan. Simulation results revealed that the

Max-Average performs better than other scheduling algorithms but the worst-case performance of Max-Average against the parameters is not formulated.

2.2.5 Survey No 5

Gopinath and Vasudevan [29] have analyzed two scheduling schemes including Max-Min and Min-Min. Using the Cloudsim simulator, the performance of both schemes were observed. Based on the results, Max-Min outperforms Min-Min in terms of makespan. However, they did not evaluate their analyzed schemes on different performance metrics to compare the results they only considered the makespan. Instead of using any benchmark dataset they created and use their own dataset.

2.2.6 Survey No 6

Patel et al. [30] proposed ELBMM scheduling algorithm using Min-Min in the first phase of its scheduling to obtain makespan. ELBMM is compared with LBMM and Min-Min on the basis of makespan only. However, they did not analyze these algorithms using any benchmark dataset and also, they did not perform any simulation.

2.2.7 Survey No 7

Mohialdeen [7] conducted a comparative analysis of four scheduling algorithms such as RS, RR, MCT, and, OLB. The algorithms were compared on the basis of cost, throughput, and makespan for the LIGO real dataset using the Cloudsim simulation tool. Experimental results have shown that no single scheduling scheme provides better performance for different types of quality services. The reason is that scheduling schemes require to be chosen depending on its ability to ensure

good quality of services with reasonable cost and reasonable equity by distributing equitably the available resources among all the jobs and address the users' constraints.

2.2.8 Survey No 8

Chen et al. [31] proposed LBIMM and PA-LBIMM algorithms . In the first step of LBIMM, Min-Min scheduling scheme is used to get makespan for the identification of the resource receiving most tasks (most heavy load resource). With respect to the makespan, an effort is being made to move tasks to lighter resources which creates load balancing between different resources. LBIMM and PA-LBIMM are simulated using Matlab. They compared LBIMM and PA-LBIMM with a Min-Min algorithm based on makespan and resource utilization. Partial definition of resource utilization is used to evaluate the performance of scheduling schemes. However, they performed experiments using a self-created dataset instead of using any benchmark dataset.

2.2.9 Survey No 9

Braun et al. [32] carried out a study of the relative performance of eleven scheduling algorithms on the basis of makespan using the HCSP dataset. They created their own simulation software to implement and analyze the algorithms. They created the HCSP dataset to provide a simulation base for the research community to analyze the performance of scheduling schemes. The results of their simulations show that the Min-Min produces minimum makespan compared to other schemes.

2.2.10 Survey No 10

Elzeki et al. [33] proposed a modified version of the Max-Min scheduling scheme. The proposed technique works on the expected execution time rather than the completion time. Improved Max-Min assigned the task to a VM having maximum

execution time. The experiments are compared with the RASA and Max-Min algorithms on the basis of makespan. However, they performed experiments using a dataset created by themselves rather than using any third-party popular dataset.

2.2.11 Survey No 11

Upendra and Purvi [34] proposed a modified version of the improved Max-Min scheduling scheme. Improved Max-Min assigned the task to a VM with maximal execution time, but the enhanced Max-Min assigned the task to a VM which has average execution time or the execution time nearest greater than the average execution time producing minimum completion time. The experiments are simulated in the CloudSim and compared with improved Max-Min on the basis of makespan. Enhanced Max-Min attained improved makespan and load balance of resources than improved Max-Min scheme. However, experiments are performed using a self-created dataset rather than using any third-party popular dataset. Also, the worst-case scenario of proposed scheduling schemes is not formulated.

2.2.12 Survey No 12

Muthucumaru et al. [2] studied dynamic mapping heuristics for independent tasks employing heterogeneous distributed computing systems. Batch and immediate mode schemes have been considered in [2]. The authors introduced three new schemes, two for immediate mode and one for batch mode. K-percent best (KPB) and Switching Algorithm (SA) have been proposed for immediate mode and the Sufferage algorithm has been proposed for batch mode. To perform a comparison of these schemes with some existing ones, simulation studies were performed on the basis of makespan. In the immediate mode, the KPB outperformed the other schemes and sufferage performed best in the batch mode.

2.2.13 Survey No 13

Hussain et al. [6] empirically analyzed ten static scheduling schemes using the Cloudsim simulation tool. The experiments were conducted using three workloads: two synthetics and one benchmark GoCJ workload against makespan, throughput, and resource utilization. However, partial definitions of metrics were used to evaluate the performance of scheduling schemes. The outcomes have revealed that sufferage and TASA perform best, based on makespan and resource utilization than other schemes. However, OLB produces poor makespan and low resource utilization.

2.2.14 Survey No 14

Safwat et al. [35] proposed a scheduling scheme TS-GA which is based on the Genetic Algorithm for assigning and executing independent tasks. The objective of the proposed technique was to minimalize the completion time, maximize resource-utilization, and decrease the execution cost- of tasks. The performance of the proposed scheme was evaluated using CloudSim toolkit and results were compared with three schemes; the default GA, Round-Robin, and the improved TS-GA algorithms against makespan, resource utilization, cost, speedup, and efficiency. The experimental results revealed that the TS-GA algorithm performs better than other scheduling schemes. However, experiments are performed using self-created dataset rather than any benchmark dataset. Also, the worst-case scenario of the proposed scheduling scheme is not formulated.

2.2.15 Survey No 15

Maipan et al. [36] proposed an Extended Min-Min Algorithm which assigns jobs on the basis of the difference between the maximum and minimum execution time. The performance of the proposed scheme was compared with RR, FCFS, Min-Min,

and Max-Min using Inspiral and Montage datasets against makespan. The simulation was done on Cloudsim and results show that the proposed algorithm gives minimum makespan as compared to other algorithms. However, they considered the only makespan to analyze the performance of the proposed scheme and also the worst-case scenario of the proposed scheduling scheme is not formulated.

2.2.16 Survey No 16

Chiang et al. [37] proposed a novel technique called Advanced MaxSufferage (AMS) to improve the performance of scheduling schemes. The proposed scheme was compared with the sufferage and MaxSufferage algorithm using the HCSP dataset on the basis of makespan and load balancing. Based on the results, it is concluded that AMS has better makespan than sufferage and MaxSufferage. Also, AMS get better load balancing result than previous algorithms in a heterogeneous environment. On the other hand, the worst-case scenario of the proposed scheduling scheme is not formulated.

2.2.17 Survey No 17

Kokilavani et al. [38] introduced Load Balanced Min-Min (LBMM) scheduling scheme to bypass the conventional Min–Min boundaries. The proposed scheme contains two-steps. The traditional Min-Min scheme is executed in the first step to get the makespan and the tasks are rescheduled in the second step to effectively utilize the unused resources. The LBMM is compared with Min-Min in C++ on the basis of resource utilization and makespan. The results depict that the proposed scheme improves resource utilization and decreases the makespan. However, experiments are performed using a self-created dataset rather than using any third-party popular dataset. Also, the worst-case scenario of the proposed scheduling scheme is not formulated.

2.2.18 Survey No 18

Maipan-uku et. al. [39] reviewed four different immediate and batch mode schemes, including Max-Min, Min-Min, MET, and, MCT. The scheduling schemes are evaluated on the basis of resource utilization and makespan. The results revealed that in an immediate mode, MCT is better than MET. In batch mode, Max-Min outperforms Min-Min based on both resource utilization and makespan. The Max-Min decreases the makespan and improves resource utilization as compared to both immediate and batch mode schemes. However, algorithms are compared theoretically instead of simulation.

2.2.19 Survey No 19

Li et al. [40] carried out a comparative analysis of seven scheduling schemes (given in Table 2.2) based on makespan, average resource utilization, average slowdown, and average offloading time. Experimental results show that mapping tasks simply based on the estimated execution time or average processing time are not appropriate. Conversely, the expected completion time must be taken into account. Also, the MCTComm algorithm offers the lowest average slowdown and shortest average wait times and, SufferageComm steadily outperforms in terms of both the average slowdown and average waiting time. While, on the basis of average resource utilization, MaxMinComm outperforms the other schemes.

Table 2.2 provides a summary of empirical studies in the form of performance metrics, scheduling techniques, used definitions and dataset, and simulator/ evaluation strategy.

TABLE 2.2: Summary of Literature Review

Ref	Perf. Measures			Definitons Used	Scheduling Techniques	Dataset Used	Simulator/ Evaluation Strategy
	Makespan	Throughput	Resource Utilization				
[5]	Yes	Yes	Yes	Partial	MCT, MET, OLB, Min-Min, Max-Min, Sufferage, TASA, RASA, RALBA	HCSP, GoCJ	CloudSim
[1]	Yes	Yes	Yes	Partial	FCFS, MCT, MET, Min-Min, Max-Min, Sufferage	HCP2N	Cloudsim
[31]	Yes	No	Yes	Partial	LBIMM, PALBIMM, Min-Min	Self-created synthetic	Matlab
[33]	Yes	No	No	Partial	Improved Max-Min, Max-Min, RASA	Self-created synthetic	Java
[30]	Yes	No	No	Partial	ELBMM, LBMM, Min-Min	Self-created synthetic	Theoretical
[26]	Yes	No	No	Partial	OLB, MET, MCT, Min-Min, Max-Min, Duplex, GA, SA, GSA, Tabu, A*	HCSP	Self-created S/W

Ref	Perf. Measures			Definitions Used	Scheduling Techniques	Dataset Used	Simulator/ Evaluation Strategy
	Makespan	Throughput	Resource Utilization				
[2]	Yes	No	No	-	KPB, SA, Sufferage, MCT, Min-Min, Max-Min	HCSP	SmartNet
[8]	Yes	Yes	No	Partial	RS, RR, OLB, MCT	LIGO	Cloudsim
[29]	Yes	No	No	-	Min-Min, Max-Min	Self-created synthetic	Cloudsim
[27]	Yes	Yes	Yes	Partial	MOTSA, MPTSA, LBTSA	Self-created synthetic	Not Known
[28]	Yes	No	Yes	-	Min-Min, Max-Min, MET, MCT, Max-Average	HCSP	simulation
[34]	Yes	No	No	-	Enhanced Max-Min, Improved Max-Min	Self-created synthetic	Cloudsim
[6]	Yes	Yes	Yes	Partial	OLB, MCT, Min-Min, Max-Min, RASA, TASA, Sufferage, LBIMM, PSSLB, PSSELB	GoCJ, Synthetic	Cloudsim

Ref	Perf. Measures			Definitions Used	Scheduling Techniques	Dataset Used	Simulator/ Evaluation Strategy
	Makespan	Throughput	Resource Utilization				
[35]	Yes	No	Yes	Partial	TS-GA, RR, GA	Self-created synthetic	Cloudsim
[36]	Yes	No	No	-	Extended Min-Min, RR, FCFS, Min-Min, Max-Min	Inspiral, Montage	Cloudsim
[37]	Yes	No	Yes	Partial	AMS, Sufferrage, Max-Sufferrage	HCSP	Cloudsim
[38]	Yes	No	Yes	Partial	LBMM, Min-Min	Self-created synthetic	Simulation
[39]	Yes	No	Yes	Partial	MET, MCT, Min-Min, Max-Min	Self-created synthetic	Theoretical
[40]	Yes	No	Yes	Partial	MET, MinHop, METComm, MCTComm, MinMinComm, MaxMinComm, SufferageComm	Self-created synthetic	Matlab

After carefully evaluating the existing comparison between different scheduling schemes, it was observed that the performance evaluation has been done by using either self-created synthetic dataset instead of any third-party popular dataset or by using partial definitions of the performance measures, like throughput and resource utilization. It was also observed that the scheduling schemes have not been evaluated with worst-case scenario dataset, and not with an ideal scheduler on a smaller scale scenario.

Chapter 3

Thorough Definitions of Performance Measures

In this chapter, we will highlight the problem in defining the performance measures i.e. throughput and resource utilization that are used in the evaluation of the schemes covered in previous chapter literature review. We will provide thorough and complete definitions of throughput and resource utilization which covers all aspects including size of the tasks and VMs with non-even computational power.

3.1 Makespan

Makespan is the maximum time for the completion of all the cloudlets (tasks) in a workload by the available resources [5].The makespan to execute n cloudlets on m virtual machines is expressed as follows [5]

$$Makespan = \max_{\forall j=1,2,3,\cdots,m} (VM\_CT)_j \tag{3.1}$$

VM_CT_j is the time to complete all tasks on j^{th} VM, and VM_CT_j is computed as follows:

$$VM\_CT_j = \sum_{i=1}^{n_j} \frac{Cloudlet_i.MI}{VM_j.MIPS} \tag{3.2}$$

where $Cloudlet_i.MI$ represents the size of $Cloudlet_i$ in terms of Million Instructions (MIs), $VM_j.MIPS$ is the computing power of VM_j in terms of Million Instructions Per Second (MIPS) and n_j denotes the total number of cloudlets assigned to VM_j.

3.2 Throughput

Throughput is referred as the number of cloudlets (jobs) completed per unit time [5].

$$Throughput = \frac{n}{Makespan} \tag{3.3}$$

where n denotes the number of jobs or cloudlets and $Makespan$ is the maximum time for the completion of all the cloudlets. This definition is for the tasks of nearly equal size. However, if the task size differs significantly, then this definition of throughput becomes meaningless so we should incorporate the size of the tasks. Let us consider an example to understand the problem of traditional throughput that is used in literature. For this we have two scenarios:

Scenario A: In this scenario, we have 15 cloudlets having different size (MIs) and three VMs of 5000 MIPS each. The total size of cloudlets is 12000 MIs and Makespan for this scenario is 1.21 second. According to the Equation 3.3 the throughput is 12.39 tasks per second.

Scenario B: In this scenario, we have 3 cloudlets of same size (5000 MIs each) and three VMs of 5000 MIPS each. The total size of cloudlets is 15000 MIs and

Makespan for this scenario is 1.1 second. According to the 3.3 the throughput is 2.72 tasks per second.

From these two scenarios, scenario A is better according to the definition of traditional throughput because in scenario A 12.39 tasks are executed per second while 2.72 tasks are executed in scenario B. But in actual scenario B is better than scenario A, because in scenario B, 15000 MIs are processed per second while in scenario A, only 12000 MIs are processed per second. To incorporate the size of the tasks we have proposed a modified throughput which covers all aspects including size of the tasks and it can be defined as follows:

$$Modified\_Throughput = \frac{TotalSizeinMIs}{Makespan} \tag{3.4}$$

where $TotalSizeinMIs$ is total length of all cloudlets and $Makespan$ is the maximum completion time to execute all the cloudlets.
According to modified throughput, the throughput of scenario A is 9917.35 MIs per second and throughput of scenario B is 13636.36 MIs per second. As in scenario B, the greater number of MIs are executed so this scenario is better than scenario A.

3.3 Resource Utilization

In the literature, resource utilization is computed by using the Average Resource Utilization Ratio (ARUR). ARUR is the ratio of average makespan to the maximum makespan of the cloud system [5],[41] and is calculated as follows:

$$ARUR = \frac{\frac{\sum_{j=1}^{m}(VM\_CT)_j}{m}}{Makespan} \tag{3.5}$$

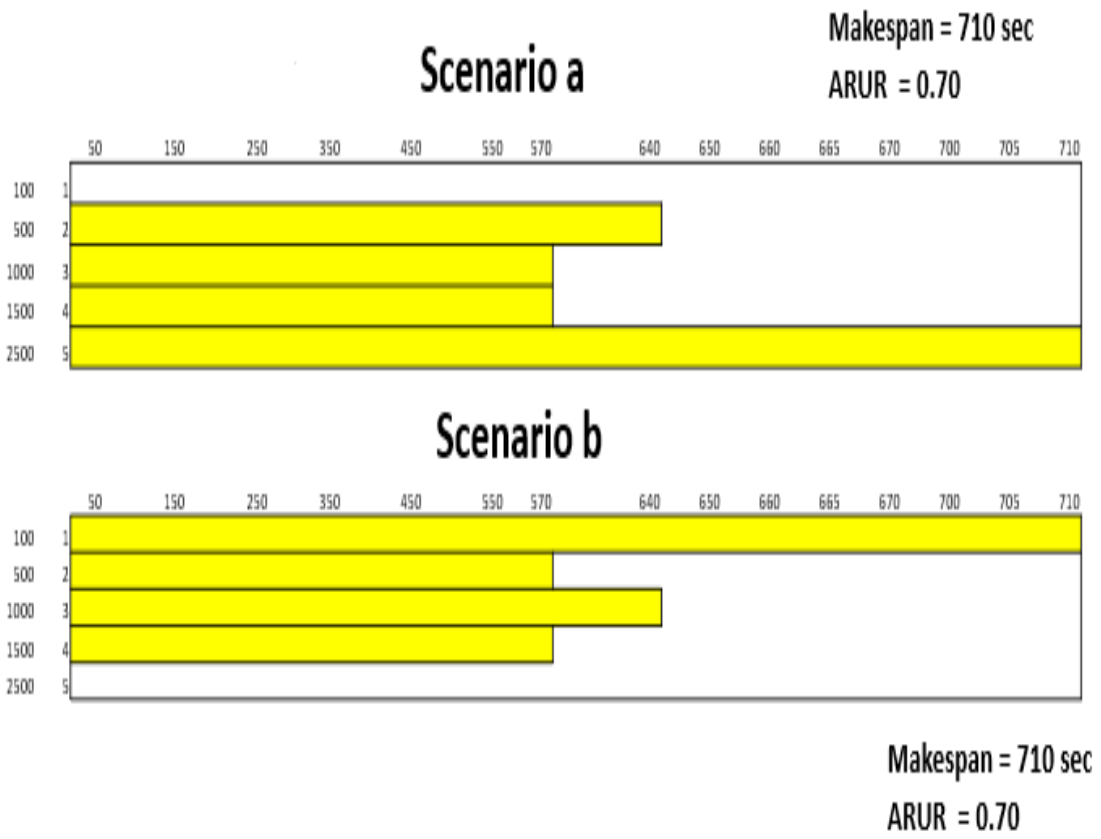

FIGURE 3.1: Resource Utilization by using ARUR

Where $\frac{\sum_{j=1}^{m}(VM\_CT)_j}{m}$ is the average completion time of all the VMs, and Makespan is the maximum time for the completion of all the system workload. ARUR is useful when we have the same computational power of VMs but when the computational power of VMs differ then ARUR does not truly reflect that which VM is actually used. Suppose we have two scenarios; in scenario a slow VM is idle whose computational power is 100 MIPS while in scenario b, fast VM is idle whose computational power is 2500 MIPS as given in Figure 3.1. Here, the question is that which scenario is better? According to ARUR, both scenarios are same because by using ARUR method both scenarios have same resource utilization i.e. 70 % as shown in Figure 3.1. But in actual the scenario a is better in which slow VM is idle.

To overcome this problem, we have proposed a Computational Power Utilization (P_U) to compute resource utilization. P_U is a performance metric which shows the overall utilization of the system, and is expressed in Equation 3.6 as follows:

$$P_U = \frac{U_P}{T_P} \times 100 \tag{3.6}$$

where U_P is Utilized Computational Power and T_P is Total Computational Power of all VMs. U_P and T_P are computed as in Equation 3.7 and 3.8

$$U_P = \sum_{j=1}^{m} \frac{(VM\_CT_j \times MIPS_j)_j}{Makespan} \tag{3.7}$$

$$T_P = U_P + I_P \tag{3.8}$$

Where I_P is Idle computational power and is computed as:

$$I_P = \sum_{j=1}^{m} \frac{(Idle\_Time_j \times MIPS_j)_j}{Makespan} \tag{3.9}$$

and Idle Time can be computed by using the following formula:

$$Idle\_Time = \sum_{j=1}^{m} (max\_Makespan - Makespan_j)_j \tag{3.10}$$

where m denotes the total number of VMs and $Makespan_j$ is the makespan on j^{th} VM.

Now consider an above example in which we have two scenarios; in scenario a slow VM is idle whose computational power is 100 MIPS while in scenario b, fast VM is idle whose computational power is 2500 MIPS. Here, the question is that which scenario is better? According to PU, the scenario a has achieved 97.2 % resource utilization when the slow VM was idle, while when fast VM was idle in scenario b

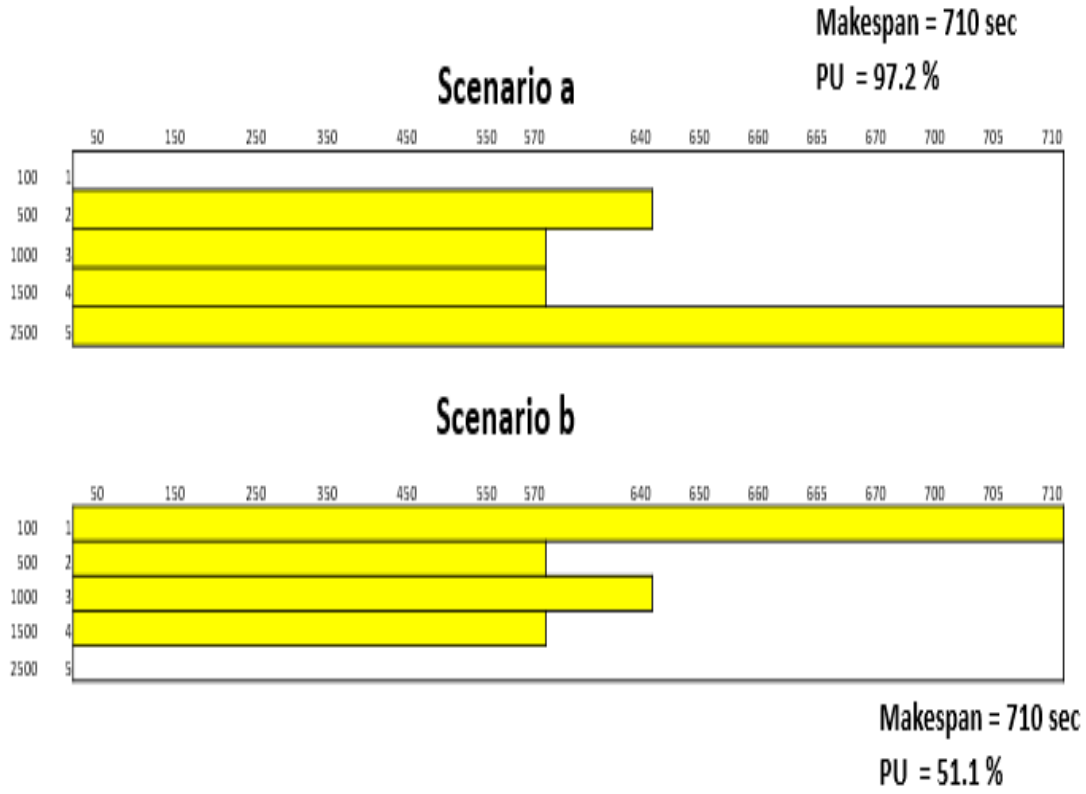

FIGURE 3.2: Resource Utilization by using thorough definition

then the resource utilization was 51.5 % as shown in Figure 3.2 , so scenario a is better than scenario b according to P_U method and also in actual the scenario a is better in which slow VM is idle.

In this thesis, eleven prominent scheduling schemes are evaluated by using thorough definitions of performance measures which covers all the aspects having task of different size and VMs with non-even computational power.

Chapter 4

Dataset and Workload Compositions

4.1 Dataset

In this research, we evaluate the performance of eleven popular static scheduling schemes by using both a third-party popular dataset, Google Cloud Jobs dataset (GoCJ) [42] and our prepared dataset named as Random Data Set (RanDS). The details of these datasets are described as follows:

4.1.1 GoCJ Dataset

GoCJ is based on the realistic Google cluster traces. The GoCJ dataset is stored in a Mendeley Data repository which consists of 21 text files [42]. 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 MIs. In GoCJ, the cloudlet sizes are distributed as: small (15,000-55,000 MI), medium (59,000-99,000 MI), large (101,000-135,000 MI), extra-large (150,000-337,500 MI), and huge (525,000-900,000 MI) (as shown in Figure 4.1). Using the distribution of cloudlet sizes in GoCJ, we have extracted the following statistics about the sizes of jobs from each text file as shown in Table 4.1.

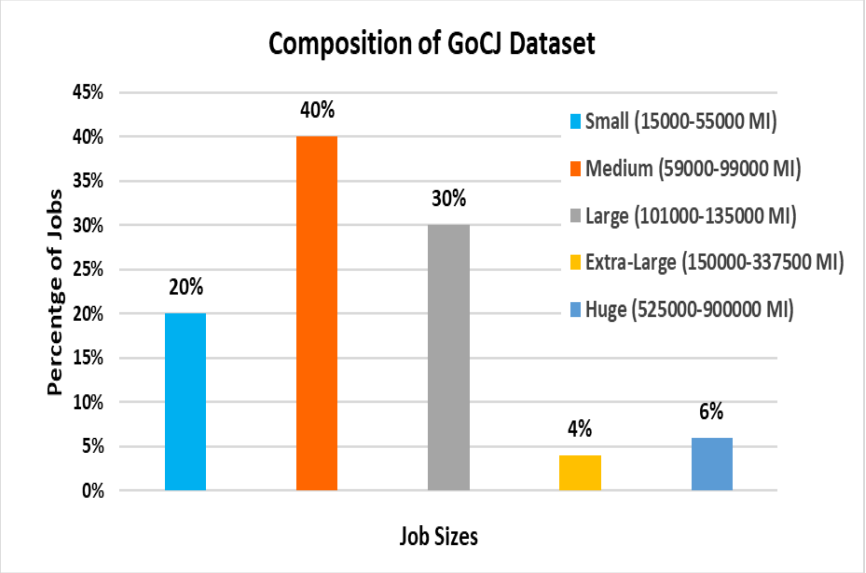

FIGURE 4.1: Composition of the GoCJ Dataset

TABLE 4.1: Statistics of the Cloudlets for GoCJ Dataset

Dataset File Name	Number of Cloudlets	Dataset File Name	Number of Cloudlets
GoCJ_Dataset_100.txt	Small Jobs = 19 Medium Jobs = 39 Large Jobs = 31 Extra Large Jobs= 3 Huge Jobs = 8	GoCJ_Dataset_350.txt	Small Jobs = 60 Medium Jobs = 131 Large Jobs = 119 Extra Large Jobs= 14 Huge Jobs = 26
GoCJ_Dataset_150.txt	Small Jobs = 22 Medium Jobs = 58 Large Jobs = 51 Extra Large Jobs= 6 Huge Jobs = 13	GoCJ_Dataset_400.txt	Small Jobs = 55 Medium Jobs =165 Large Jobs = 139 Extra Large Jobs= 18 Huge Jobs = 23
GoCJ_Dataset_200.txt	Small Jobs = 42 Medium Jobs = 75 Large Jobs = 63 Extra Large Jobs= 4 Huge Jobs = 16	GoCJ_Dataset_450.txt	Small Jobs = 85 Medium Jobs = 184 Large Jobs = 141 Extra Large Jobs=17 Huge Jobs = 23
GoCJ_Dataset_250.txt	Small Jobs = 58 Medium Jobs = 93 Large Jobs = 79 Extra Large Jobs= 7 Huge Jobs = 13	GoCJ_Dataset_500.txt	Small Jobs = 96 Medium Jobs = 191 Large Jobs = 155 Extra Large Jobs= 25 Huge Jobs = 33
GoCJ_Dataset_300.txt	Small Jobs = 61 Medium Jobs = 114 Large Jobs = 95 Extra Large Jobs= 7 Huge Jobs = 23	GoCJ_Dataset_550.txt	Small Jobs = 98 Medium Jobs = 226 Large Jobs = 176 Extra Large Jobs= 17 Huge Jobs = 33

Dataset File Name	**Number of Cloudlets**	**Dataset File Name**	**Number of Cloudlets**
GoCJ_Dataset_600.txt	Small Jobs = 114 Medium Jobs = 229 Large Jobs = 190 Extra Large Jobs=25 Huge Jobs = 42	GoCJ_Dataset_850.txt	Small Jobs = 148 Medium Jobs = 363 Large Jobs = 259 Extra Large Jobs= 33 Huge Jobs = 47
GoCJ_Dataset_650.txt	Small Jobs = 118 Medium Jobs = 243 Large Jobs = 210 Extra Large Jobs= 33 Huge Jobs = 46	GoCJ_Dataset_900.txt	Small Jobs = 156 Medium Jobs = 344 Large Jobs = 298 Extra Large Jobs= 44 Huge Jobs = 58
GoCJ_Dataset_700.txt	Small Jobs = 116 Medium Jobs = 295 Large Jobs =222 Extra Large Jobs= 29 Huge Jobs = 38	GoCJ_Dataset_950.txt	Small Jobs = 179 Medium Jobs = 347 Large Jobs = 337 Extra Large Jobs= 39 Huge Jobs = 48
GoCJ_Dataset_750.txt	Small Jobs = 145 Medium Jobs = 284 Large Jobs = 240 Extra Large Jobs= 28 Huge Jobs = 53	GoCJ_Dataset_1000.txt	Small Jobs = 162 Medium Jobs = 423 Large Jobs = 322 Extra Large Jobs= 33 Huge Jobs = 60
GoCJ_Dataset_800.txt	Small Jobs = 160 Medium Jobs = 315 Large Jobs = 252 Extra Large Jobs= 27 Huge Jobs = 46	-	-

4.1.2 RanDS

RanDS is our own prepared dataset which is used to test the worst-case scenario of scheduling schemes. Initially, a random dataset was downloaded from the internet which has 20 text files. Each text file consists of cloudlets with different sizes but the length of each cloudlet (MIs) in the random dataset was very small as compare to the GoCJ dataset. In the file containing 500 cloudlets, the length of the smallest cloudlet was only 2MIs and the length of the largest cloudlet was 44930 MIs. And for 1000 cloudlets, the length of the smallest cloudlet was only 118 MIs which is very small as compared to the smallest cloudlet of the GoCJ dataset. So, two files are chosen from the random dataset (i.e., with 500 and 1000 number of cloudlets) and the size of each cloudlet is increased by adding 1000 MIs. Now in updated RanDS for 500 number of cloudlets, the length of the smallest cloudlet is 2000 MIs and the length of the largest cloudlet is 44930000 MIs. Similarly, for 1000 cloudlets the length of the smallest cloudlet is 1118 MIs and the length of the

largest cloudlet is 13001000 MIs, respectively. The sizes of some of the cloudlets for RanDS is given in Table 4.2 as a sample.

TABLE 4.2: Statistics of the Cloudlets for RanDS

Length of 500 Cloudlets (MIs) in Random Dataset	**Length of 500 Cloudlets (MIs) in RanDS**	**Length of 1000 Cloudlets (MIs) in Random Dataset**	**Length of 1000 Cloudlets (MIs) in RanDS**
73	73000	5800	6800
78	78000	219	1219
24	24000	7334	8334
155	155000	2000	3000
120	120000	3098	4098
216	216000	118	1118
873	873000	1802000	1803000
1070	1070000	20933	21933
947	947000	20933	21933
1152	1152000	135678	136678
822	822000	1782225	1783225
877	877000	13000000	13001000
1189	1189000	11422	12422
815	815000	31622	32622
843	843000	30443	31443
824	824000	42755	43755
943	943000	40081	41081
1112	1112000	32274	33274
39228	39228000	37394	38394
44930	44930000	32555	33555
36363	36363000	37901	38901
41105	41105000	41616	42616
41733	41733000	33780	34780
30979	30979000	44474	45474

Chapter 5

Simulation and Results

5.1 Experimental Setup

Use of real cloud, for evaluation of scheduling and resource allocation policies, is often limited and a challenging problem. In real infrastructures, an extremely difficult task is the reproduction of reliable results. In cloud infrastructure reconfiguration of many experiments is a costly and time-consuming task. Moreover, experiments cannot be performed in a repeatable, reliable, and scalable manner in a real cloud environment. Therefore, an ideal alternative 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). So, for empirical evaluation, we use a well-known simulator; CloudSim [43] (version 3.0.3). It is an open-source framework for modeling and analyzing the performance of cloud services. A user job/task is represented as cloudlet in CloudSim and the job's size (computational requirement) is measured in terms of MIs.

The experiments are conducted on a machine equipped with Intel Core i3-4010U Quad-core processor (having 1.70 GHz clock speed) and 4.00 GB of main memory. Liu and Cho [44] describe the workloads and computing machines on a Google cluster and found that 93% of Google cluster machines are fairly homogenous and only

6% of machines with higher computing capabilities. We construct an experimental setup for empirical evaluation using the characteristics of the real computing machines (found in the analysis of Liu and Cho [44]). All the experiments are performed by using 30 VMs, hosted on 10 host machines within one data-center. Table 5.1 shows the configuration details for the simulation environment used. Figure 5.1 presents the overall statistics of the VMs and their computing power in terms of MIPS. As shown in Figure 5.1, the slowest and fastest VMs have the computing power of 100 and 4000 MIPS, respectively.

TABLE 5.1: Configuration of the Simulation Environment

Simulator/Version	Cloudsim version 3.0.3
Computing power of cloudhost machines	4 Dual-core (4000 MIPS), 6 Quad-core (4000 MIPS)
Total cloudhost machines	10
Host machine memory	16,384 MBs (each)
Total VMs	30 heterogeneous VMs
Total Cloudlets	100,150,200,250,300,350,400,450,500,550, 600,650,700,750,800,850,900,950,1000

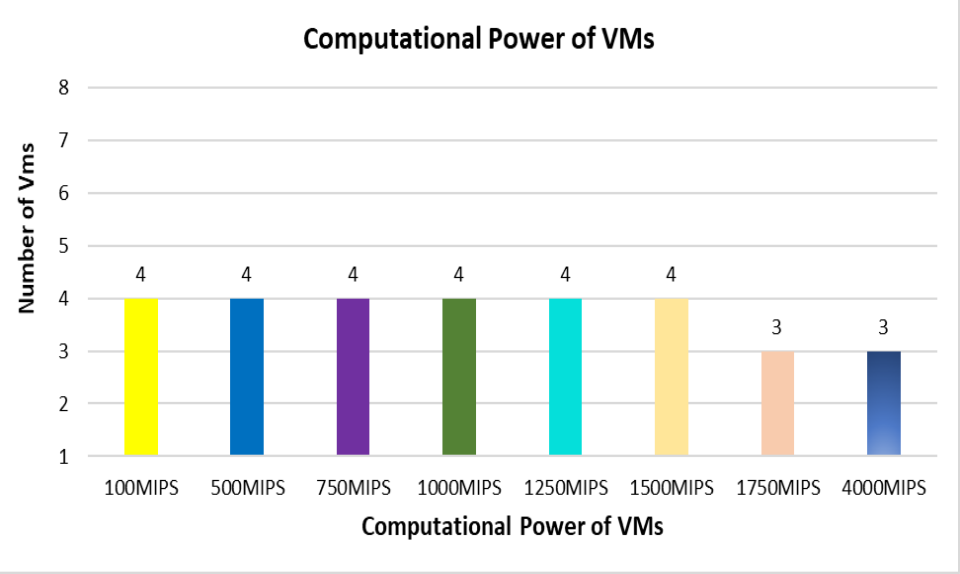

FIGURE 5.1: Computational Power of VMs

Later on, we performed an analysis of best performing scheme by changing the overall statistics of the VMs and their computing power (MIPS). This experiment

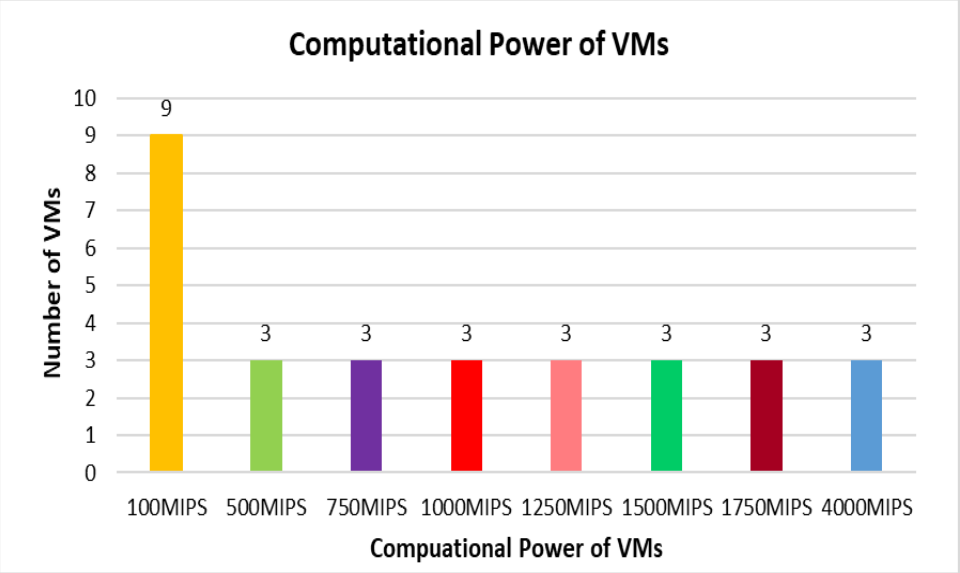

FIGURE 5.2: Computational Power of VMs (for scenario A)

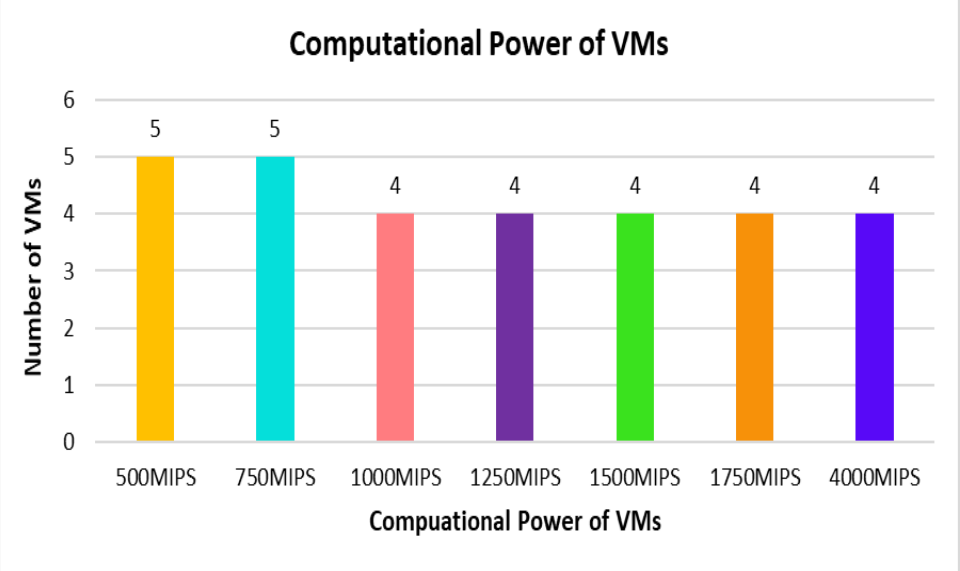

FIGURE 5.3: Computational Power of VMs (for scenario B)

is conducted to analyze the impact of outperforming scheme on resource utilization. The number of cloudlets used in this experiment are 100, 500, and 1000 from GoCJ. This experiment is comprised of two scenarios:

Scenario A: In this scenario, the number of VMs with slowest computational power (100 MIPS) is increased as shown in Figure 5.2.

Scenario B: In this scenario, the computational power for slowest VMs is changed from 100 MIPS to 500 MIPS. Details are shown in Figure 5.3.

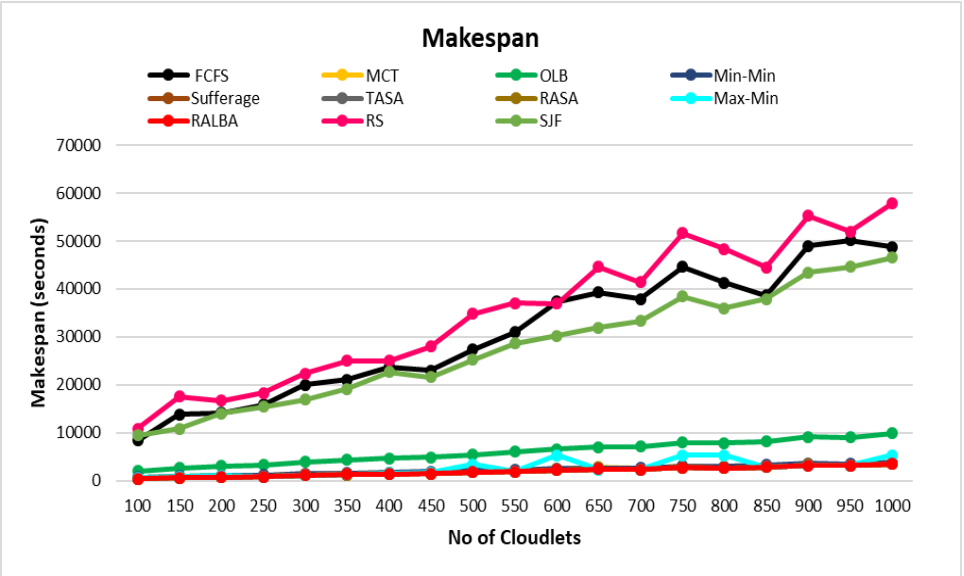

FIGURE 5.4: Makespan Results Using GoCJ Dataset

5.2 Experimental Results

This section explains the simulation results obtained after running the task scheduling schemes on CloudSim. We consider eleven prominent static scheduling schemes including: FCFS [30], RS [8], SJF [7], OLB [9] , MCT [33], Min-Min [6], Max-Min [6], TASA [2], RASA [4], Sufferage [6], and RALBA [17]). An in-depth empirical study is conducted to better understand the scheduling mechanisms in terms of makespan, throughput, and resource utilization. Each experiment is conducted 10 times and the analysis are performed on average values.

5.3 Makespan-Based Results

We use term makespan to represent the completion of all execution of the cloudlets in a workload. The scheduling scheme whose makespan is minimum is considered best. Figure 5.4 demonstrates the makespan results of eleven scheduling schemes for GoCJ benchmark workload. The x-axis shows the number of cloudlets (jobs) and y-axis shows the makespan (measured in seconds). As shown in Figure 5.4, makespan increases in most cases when we increase the number of cloudlets. For more clarity, the average makespan of all the schemes using different number of

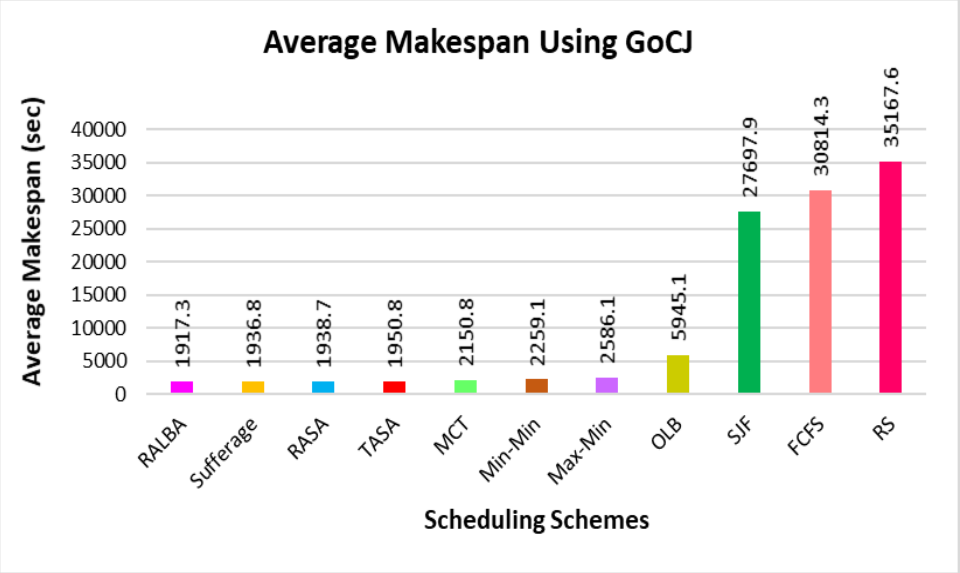

FIGURE 5.5: Average Makespan Using GoCJ Dataset

cloudlets is calculated as follows:

$$Avg\_Makespan = \sum_{i=1}^{N} \frac{Makespan_i}{N} \tag{5.1}$$

where N represents the number of experiments conducted for each scheduling scheme and $Makespan_i$ represents the makespan of ith experiment. Each experiment is repeated using a varying number of cloudlets (*i.e.*, $cloudlets100 - 1000$, as given in Table 5.1). The average makespan results for all scheduling schemes are presented in Figure 5.5. The x-axis presents scheduling schemes and y-axis shows average makespan of each scheduling scheme (measured in seconds). As shown in Figure 5.5, for the execution of GoCJ dataset, our results revealed that the RALBA, Sufferage, and RASA attains lower makespan as compared to the other scheduling schemes. However, there is a minor difference between Sufferage, and RASA with respect to makespan for GoCJ workload. On the other hand, FCFS, and SJF achieves largest makespan while RS performs worse than all other schemes.

Figure 5.6 and 5.7 show the makespan results of 11 scheduling schemes using RanDS for 500 and 1000 cloudlets. RALBA, RASA, and Max-Min achieved the

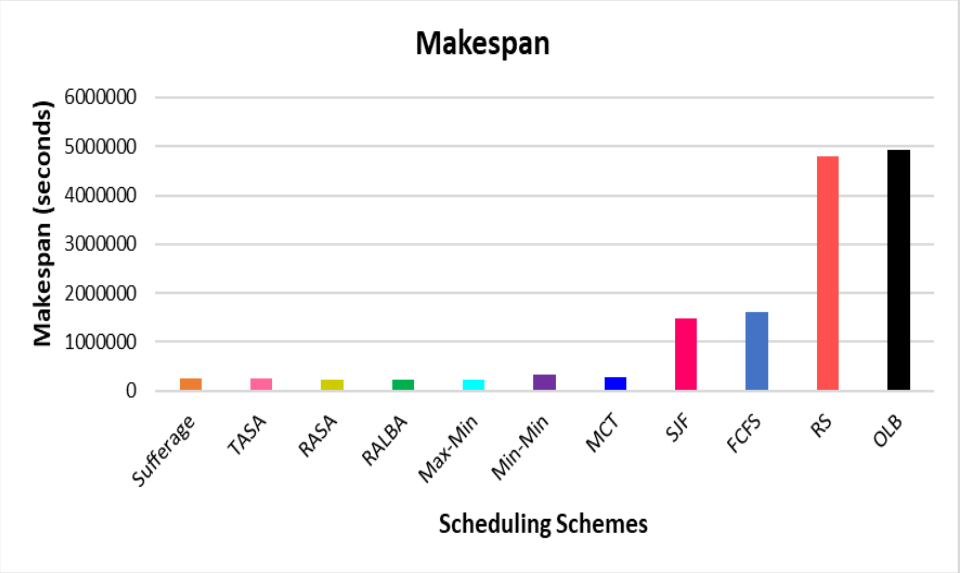

FIGURE 5.6: Makespan for 500 Cloudlets Using RanDS

same makespan on RanDS. After RALBA and RASA, Sufferage and TASA algorithms gives the minimum makespan when compared to rest of the schemes using RanDS. On the other hand, OLB has achieved highest makespan for 500 cloudlets-based scheduling and for 1000 number of cloudlets RS has perfomed worst. Figure 5.8 shows the average makespan result for RanDS. As shown in Figure 5.8, RALBA, RASA and Max-Min achieved the same makespan on average and outperforms all other schemes while OLB has performed worst. The reason is that, OLB keeps all machines as busy as possible regardless of considering the job's execution time on that particular machine. As a result, OLB scheduling scheme mostly results in poor makespan.

5.4 Throughput-Based Results

In this study, we have computed throughput in two different ways; one is by using traditional method of calculating throughput which was used in the previous chapter literature review as: throughput is referred as the number of cloudlets executed per second. But this method of defining throughput is partial because the length of cloudlets varies in the dataset, so, we defined throughput as: number of million instructions (total length of cloudlets) executed per unit time which we

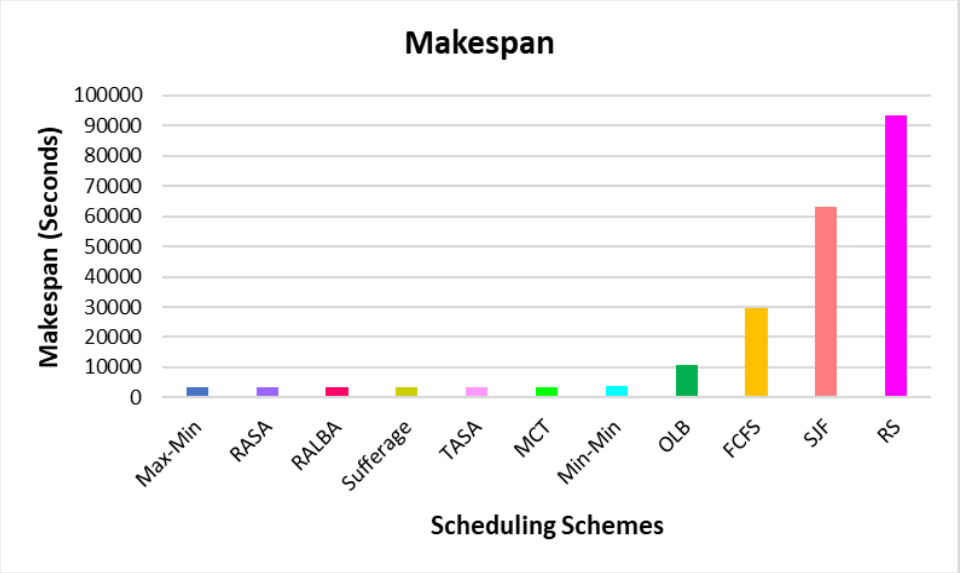

FIGURE 5.7: Makespan for 1000 Cloudlets Using RanDS

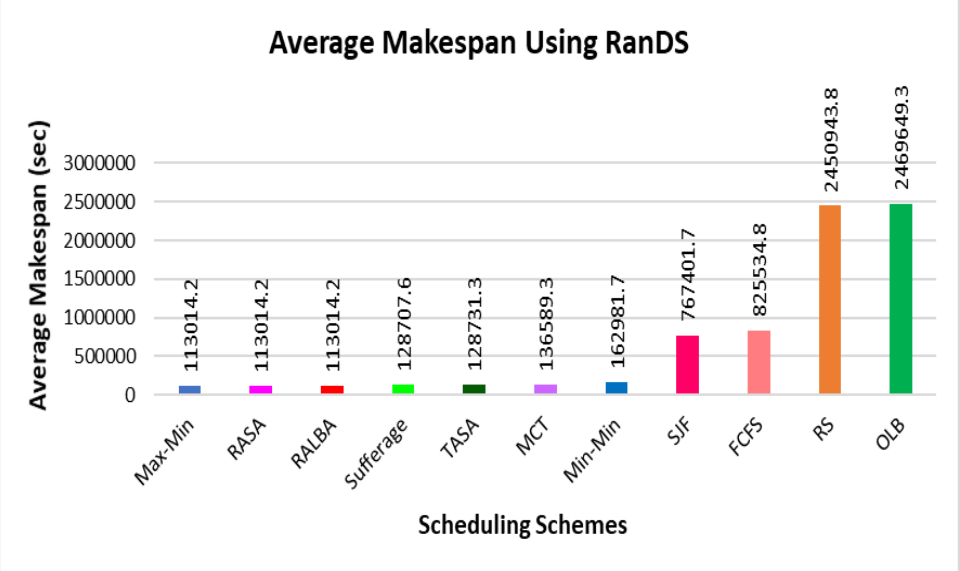

FIGURE 5.8: Average Makespan Using RanDS Dataset

called modified throughput as discussed in Chapter 1 in equation **??**. A scheduling scheme producing higher throughput is assumed a better performing scheme. Figure 5.9 presents the results of modified throughput and Figure 5.10 presents the traditional throughput results for the execution of GoCJ workload. As shown in Figure 5.9, RALBA has executed greater number of million instructions per unit time for different number of cloudlets using GoCJ dataset. After RALBA, the second-best scheduling scheme is RASA on the basis of both modified and traditional throughput. For more clarity in the simulation results, the average

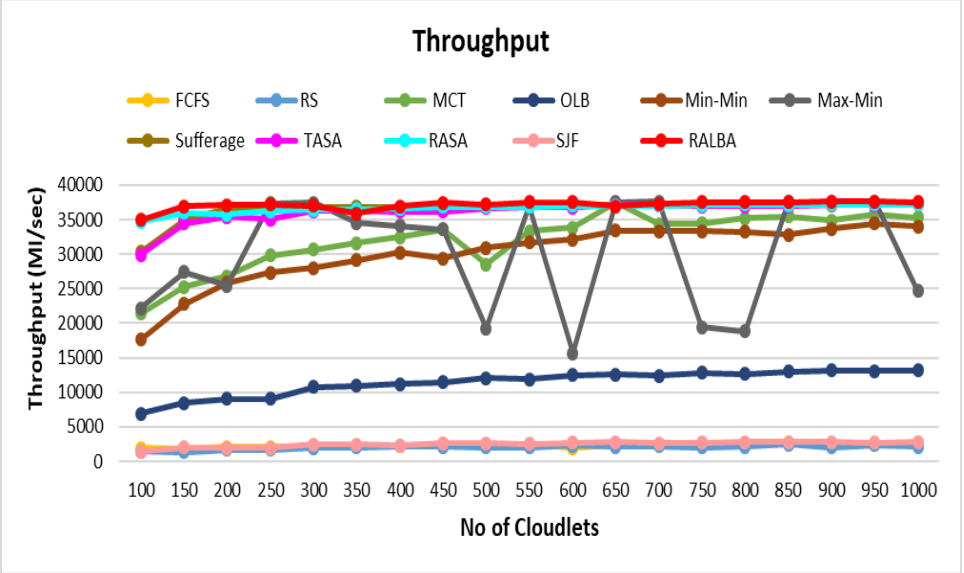

FIGURE 5.9: Modified Throughput Using GoCJ Dataset

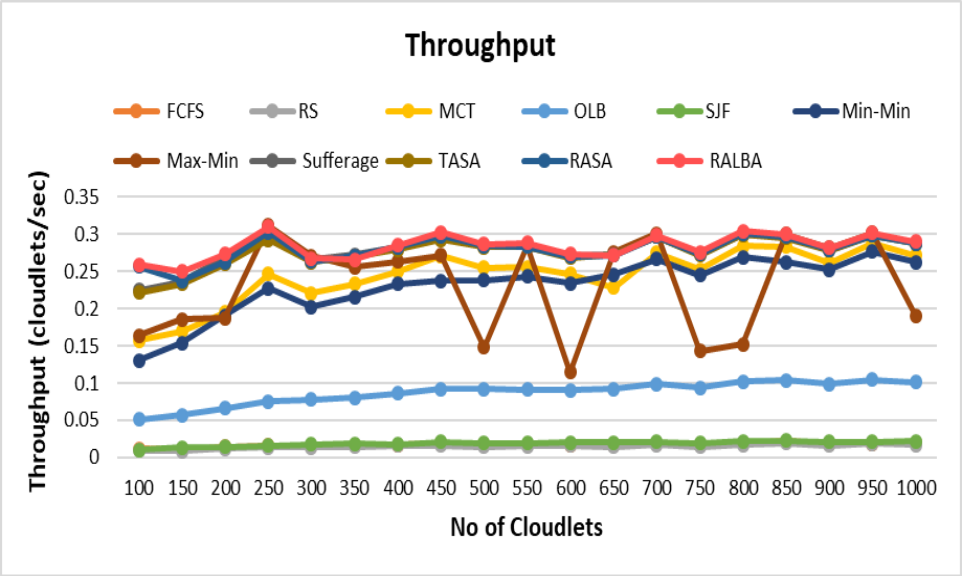

FIGURE 5.10: Throughput Using GoCJ Dataset

throughput is calculated as:

$$Avg\_Throughput = \sum_{i=1}^{N} \frac{Throughput_i}{N} \tag{5.2}$$

where N represents the number of experiments conducted for each scheduling algorithm and $Throughput_i$ represents the throughput of ith experiment. Each experiment is repeated using a varying number of cloudlets (*i.e.*, $cloudlets100 - 1000$, as

given in Table 5.1). Figure 5.11 presents the average modified throughput for all scheduling schemes using GoCJ dataset. The x-axis shows the scheduling schemes and y-axis shows average modified throughput (MI/second). According to the modified throughput, RALBA executes 37125.39 million instructions per second on average, while 0.2832 jobs per second according to traditional throughput as shown in Figure 5.12. Likewise, average makespan results, RALBA and RASA achieved the highest throughput. Similarly, RS, FCFS, and SJF techniques have the least throughput as shown in Figure 5.11 and 5.12.

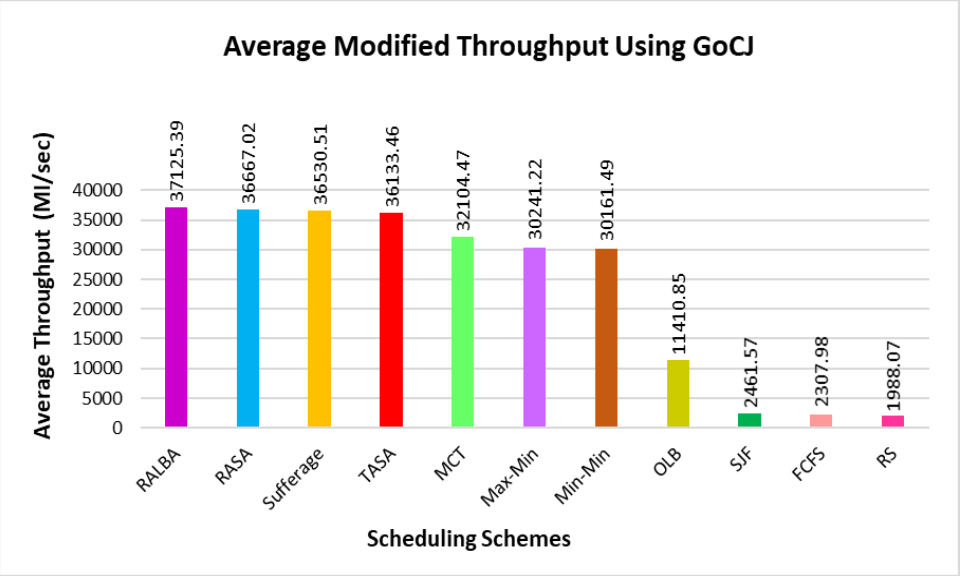

FIGURE 5.11: Average Modified Throughput Using GoCJ Dataset

Figure 5.13 and 5.14 show the throughput results of 11 scheduling schemes using RanDS for 500 cloudlets. As shown in Figures 5.13 and 5.14, Max-Min, RALBA and RASA have achieved same throughput, similarly TASA and Sufferage attained same throughput using RanDS, while RS has least throughput than all other schemes.

Figure 5.15 and 5.16 present throughput for 1000 number of cloudlets using RanDS. It is observed that the RALBA, RASA, TASA, Sufferage, and Max-Min produce the same throughput using RanDS but the performance of these scheduling schemes become poor by increasing the number of cloudlets (i.e., from 500 to

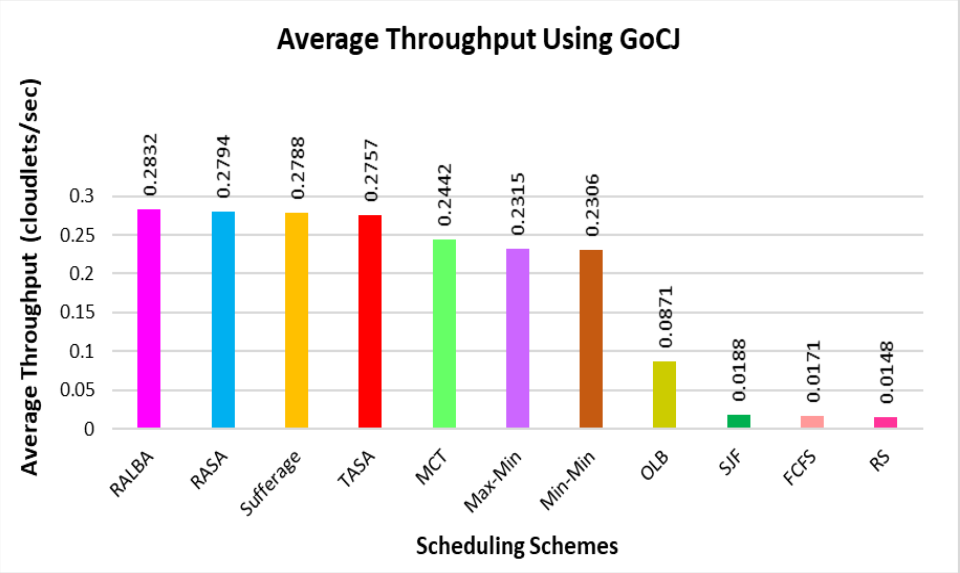

FIGURE 5.12: Average Throughput Using GoCJ Dataset

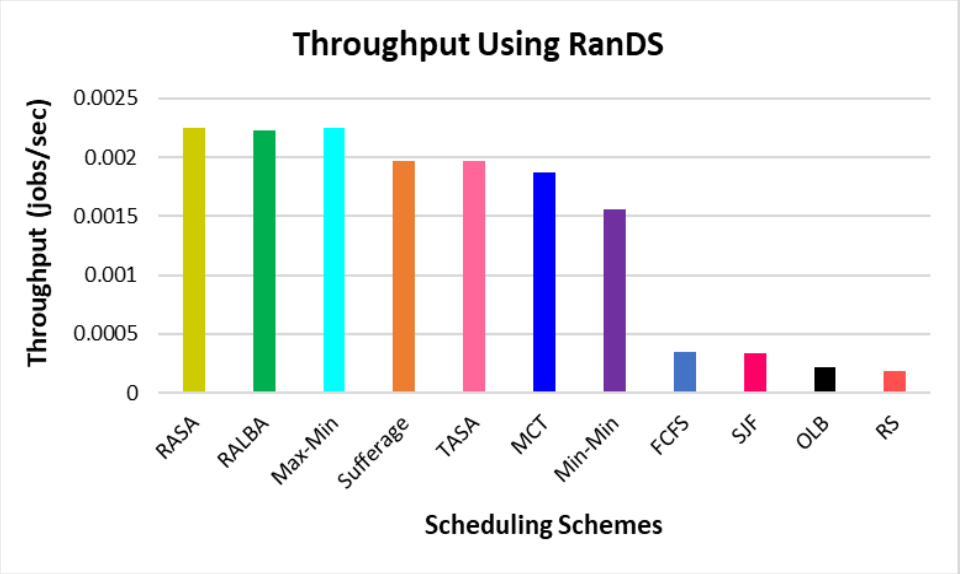

FIGURE 5.13: Throughput for 500 Cloudlets Using RanDS

1000). However, SJF and FCFS have least modified throughput while RS performed worse than all other scheduling schemes on average using RanDS as given in Figure 5.17.

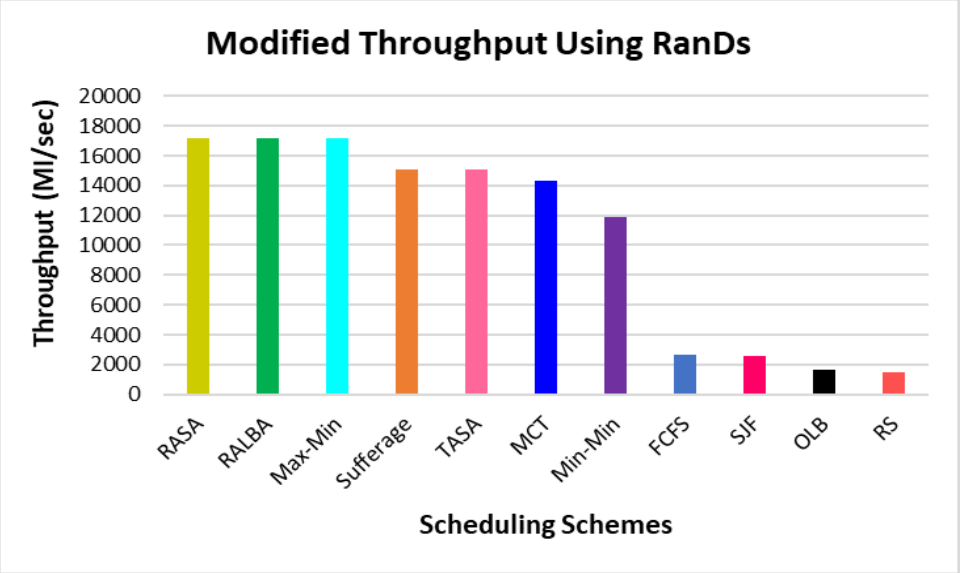

FIGURE 5.14: Modified Throughput for 500 Cloudlets Using RanDS

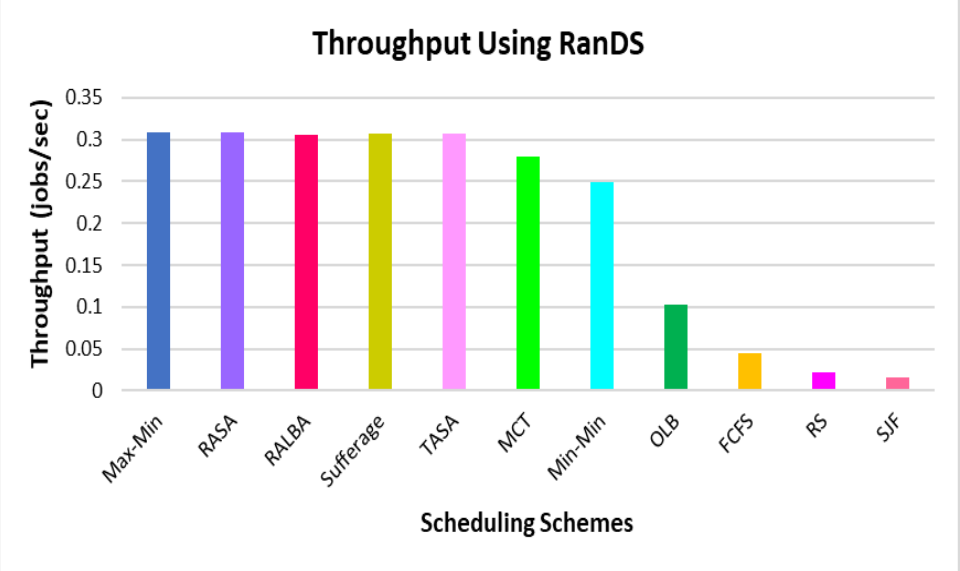

FIGURE 5.15: Throughput for 1000 Cloudlets Using RanDS

5.5 Resource Utilization-Based Results

This parameter indicates the efficiency of an algorithm while keeping the available resources busy during the scheduling. The optimization of resources is achieved by reducing the idle time of the resource. In this work, resource utilization is computed in two ways: one is by using **ARUR** and second method is by using $\boldsymbol{P_U}$.

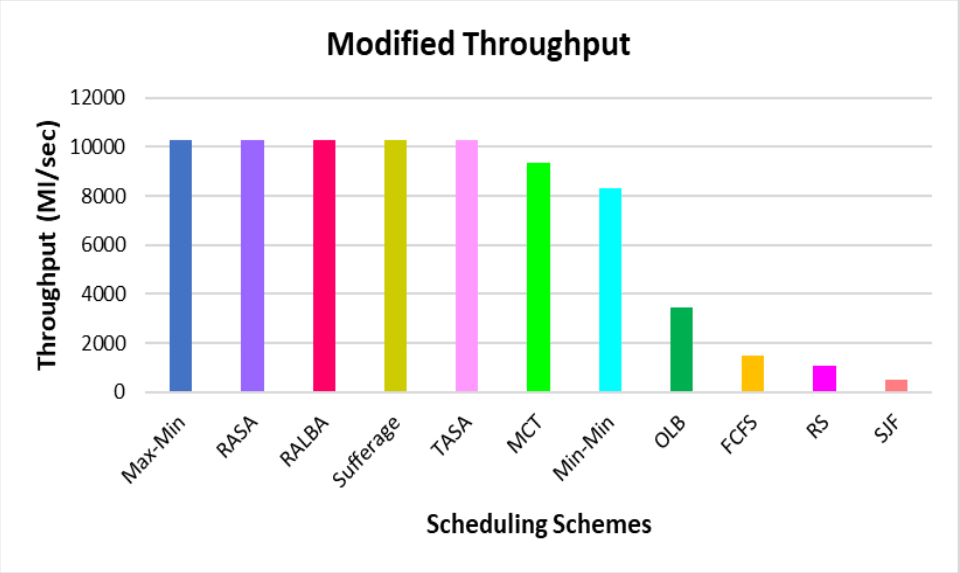

FIGURE 5.16: Modified Throughput for 1000 Cloudlets Using RanDS

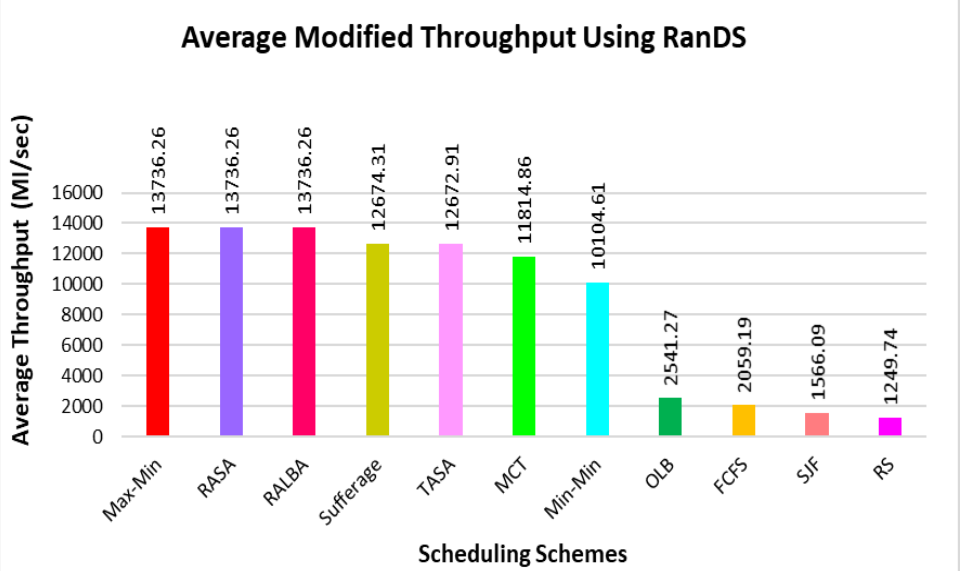

FIGURE 5.17: Average Modified Throughput Using RanDS

5.6 ARUR-Based Results

Figure 5.18 shows the ARUR-based experimental results of eleven schemes for GoCJ benchmark workload. The ARUR value remains between 0 and 1, where the value near to 1 indicates exemplary resource utilization (i.e., closest to 100% resource utilization). As shown in Figure 5.18, RALBA gives relatively highest resource utilization than other scheduling schemes. Max-Min has much variation in resource utilization ratio with the increasing number of cloudlets. The reason

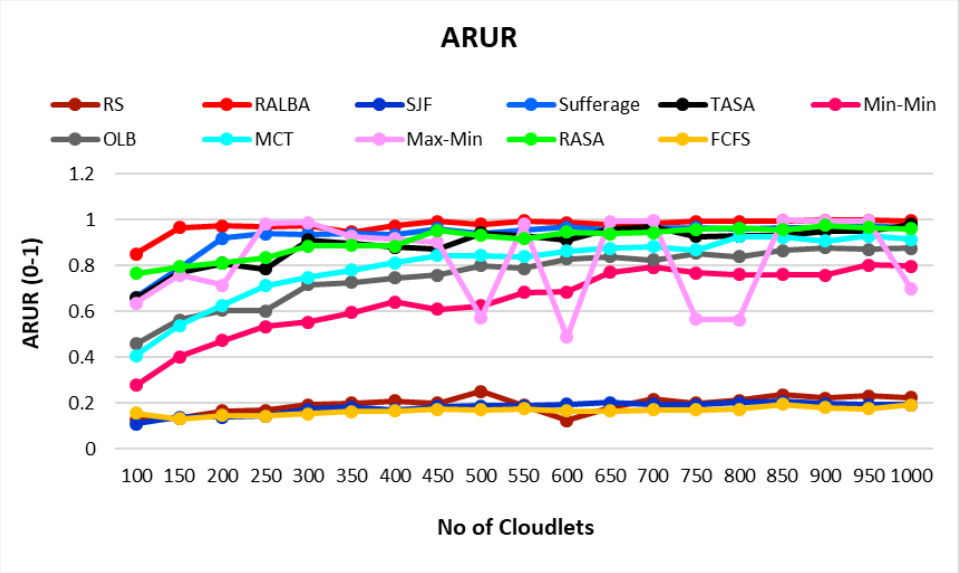

FIGURE 5.18: ARUR Using GoCJ

is that Max-Min schedules long and short tasks simultaneously, so sometimes it utilizes almost 100% resources and sometimes its utilization undergoes below 50% as given in Figure 5.18. However, FCFS and SJF produced almost similar results and performs worst in terms of average resource utilization ratio.

For more clarity Mean ARUR value for each scheduling is described based on the following equation:

$$Mean\_ARUR = \sum_{i=1}^{N} \frac{ARUR_i}{N} \tag{5.3}$$

where N represents the number of experiments conducted for each scheduling algorithm and $ARUR_i$ represents ARUR of ith experiment.

Figure 5.19 presents the Mean ARUR results for all schemes. RALBA and sufferage algorithms produce the highest resource utilization, (i.e,. 97.5% and 92.9%) as compared to other scheduling schemes. The FCFS scheduling algorithm produces the least resource utilization among all scheduling schemes using GoCJ dataset.

Figure 5.20 and 5.21 present the ARUR results of 11 scheduling schemes using RanDS for 500 and 1000 cloudlets. As shown in Figure 5.20 and 5.21 for

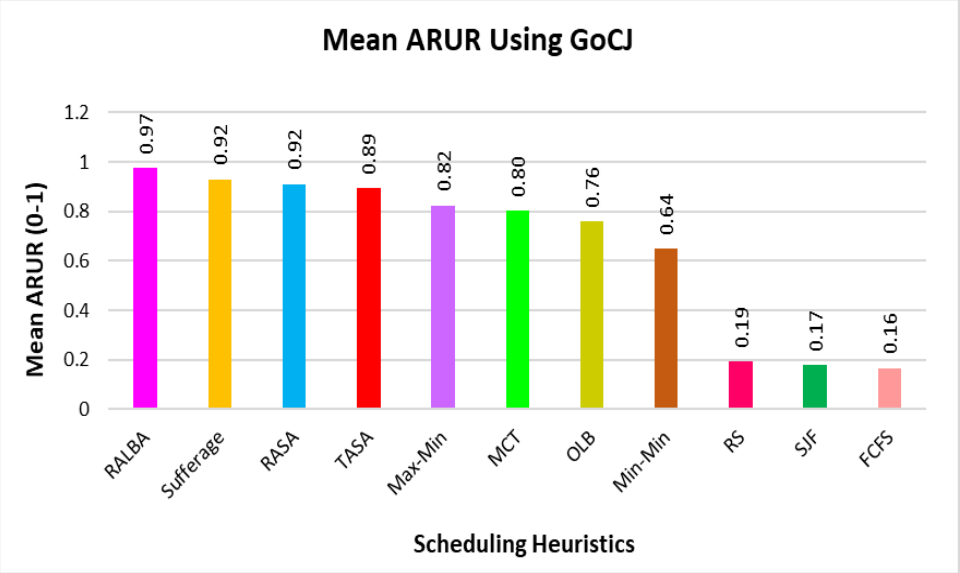

FIGURE 5.19: Mean ARUR Using GoCJ

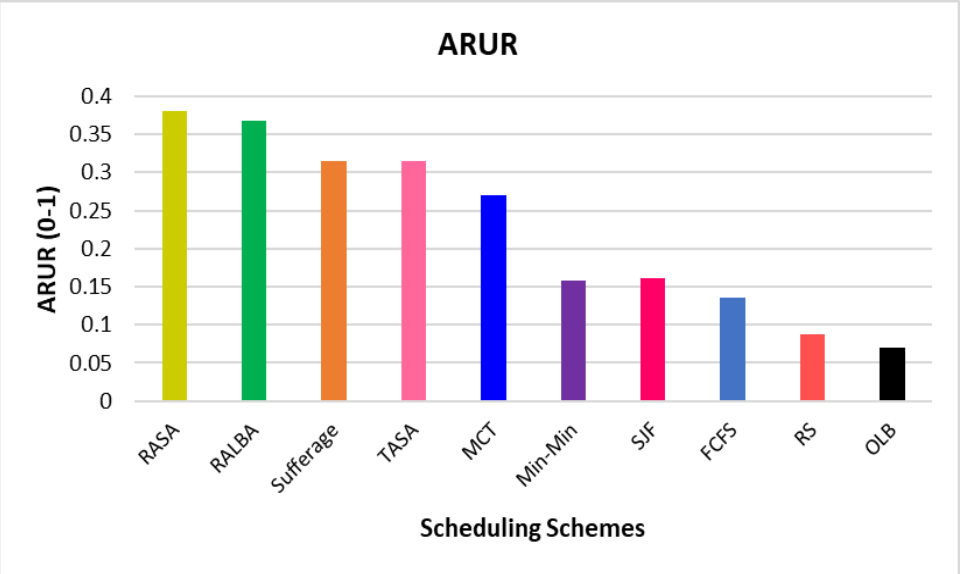

FIGURE 5.20: ARUR for 500 Cloudlets Using Random Dataset

RanDS, Max-Min scheduling algorithm outperforms all other schemes on the basis of ARUR. On RanDS the RALBA has attained only 36% resource utilization for 500 cloudlets and for 1000 cloudlets its utilization becomes poorer (i.e., 16%) as given in Figure 5.21. The experimental results show that RASA is second-best algorithm on the basis of ARUR using RanDS. The MCT scheme attains the highest ARUR (13.5% resource utilization), as compared to Min-Min, OLB and FCFS for RanDS. Furthermore, SJF and RS scheduling schemes have poor resource utilization among all scheduling schemes.

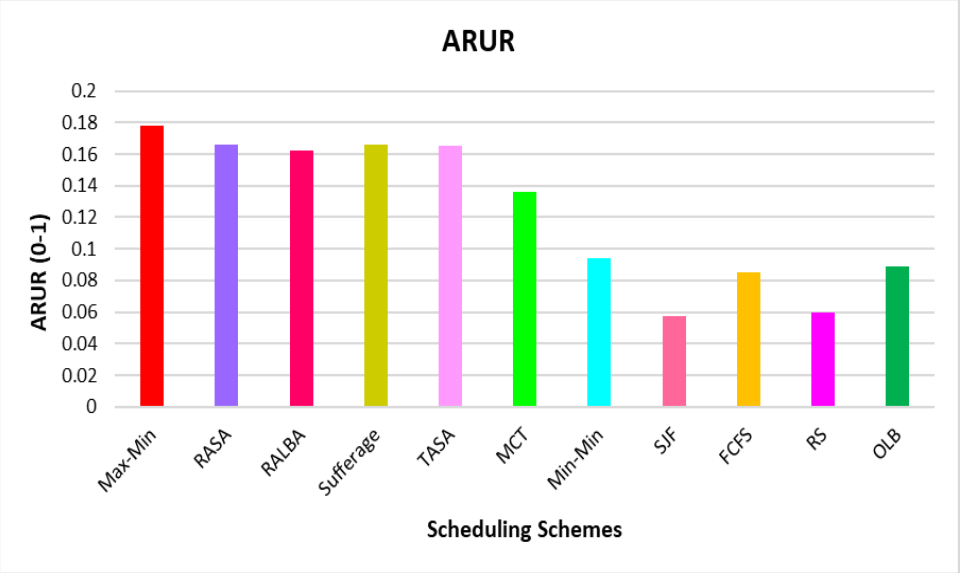

FIGURE 5.21: ARUR for 1000 Cloudlets Using RanDS

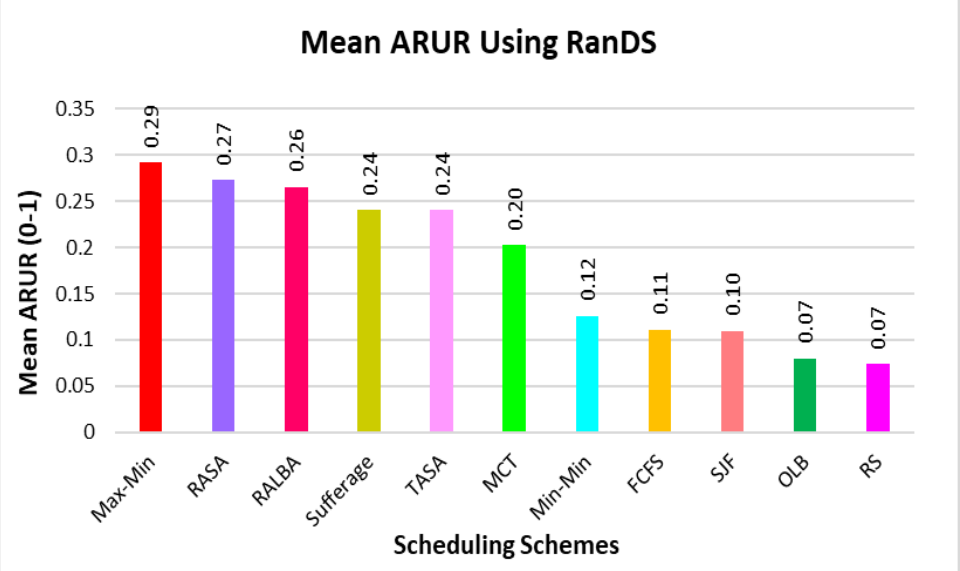

FIGURE 5.22: Mean ARUR Using RanDS

Figure 5.22 show the mean ARUR based experimental results for the execution of RanDS of all scheduling schemes. Using RanDS, Max-Min has outperformed all other scheduling schemes on the basis of mean ARUR while OLB and RS have given poor results on mean ARUR.

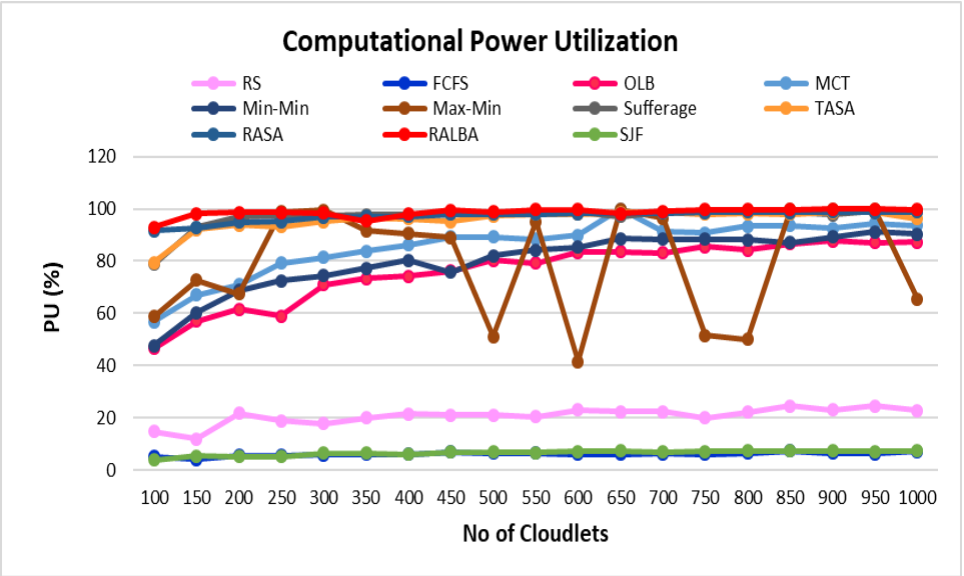

FIGURE 5.23: Computational Power Utilization Using GoCJ

5.7 P_U-Based Results

In this section, we compare all scheduling schemes on the basis of computational power utilization (P_U) of each VM by using own prepared formula as discussed in Chapter 1 in equation **??**. The algorithm which consumes more computational power is considered best. Figure 5.23 presents computational power utilization-based results of all scheduling schemes for GoCJ benchmark workload. The x-axis presents number of cloudlets and y-axis shows the computational power utilization in percentage. For more clarity in the simulation results, the average computational power utilization is calculated using Equation 5.4 and results are given in Figure 5.24.

$$Average\_P_U = \sum_{i=1}^{N} \frac{P_{Ui}}{N} \tag{5.4}$$

Figure 5.24, presents the average computational power utilization-based results for the execution of GoCJ workload. RALBA has attained 98.6% computational power utilization and outperforms rest of the other algorithms. After RALBA, the second-best algorithm is RASA, which is resource aware scheduling algorithm and it attained 97% utilization, as shown in Figure 5.24. OLB algorithm has 76% computational power utilization when we have different number of cloudlets using

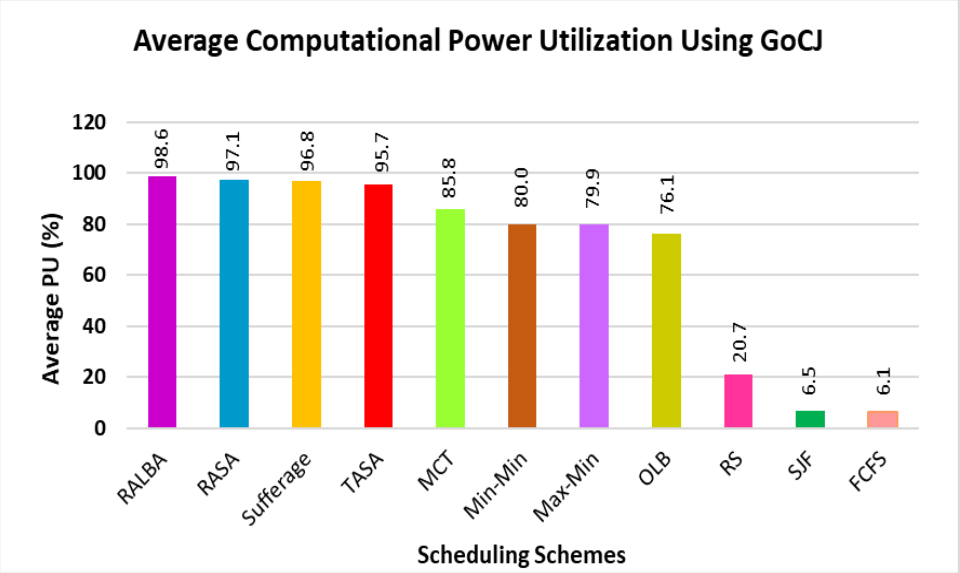

FIGURE 5.24: Average Computational Power Utilization Using GoCJ

GoCJ (as given in Table 1). However, RS has 20.78% computational power utilization because it assigns tasks to VMs randomly without considering the current load of VMs. On the other hand, SJF and FCFS have almost similar resource utilization (i.e., 6.5% and 6.1%) which is very low as compared to other scheduling schemes.

Figure 5.25 and 5.26 present the computational power-utilization results of 11 scheduling schemes using RanDS for 500 and 1000 cloudlets. As shown in Figure 5.25 and 5.26 for RanDS, Max-Min scheduling algorithm outperforms all other schemes on the basis of resource utilization. The reason is that Max-Min is likely to do better than other scheduling schemes in cases where there are many shorter tasks than longer tasks. However, on RanDS, the RALBA has attained only 45.6% resource utilization for 500 cloudlets and for 1000 cloudlets its utilization becomes poorer (i.e., 27.3%) as given in Figure 5.26. The experimental results show that both RASA and RALBA attain equal resource utilization using RanDS. The MCT scheme attains highest resource utilization (i.e., 37.9%), as compared to Min-Min, OLB, SJF and FCFS for RanDS.

Figure 5.27 shows the average computational power utilization-based results for the execution of RanDS of all scheduling schemes. Using RanDS; Max-Min, RASA,

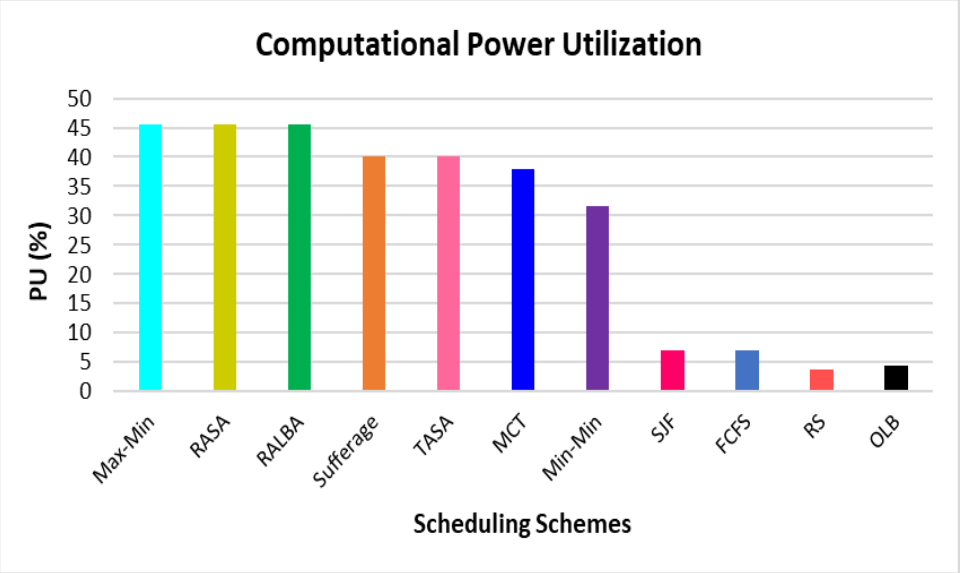

FIGURE 5.25: Computational Power Utilization for 500 Cloudlets Using RanDS

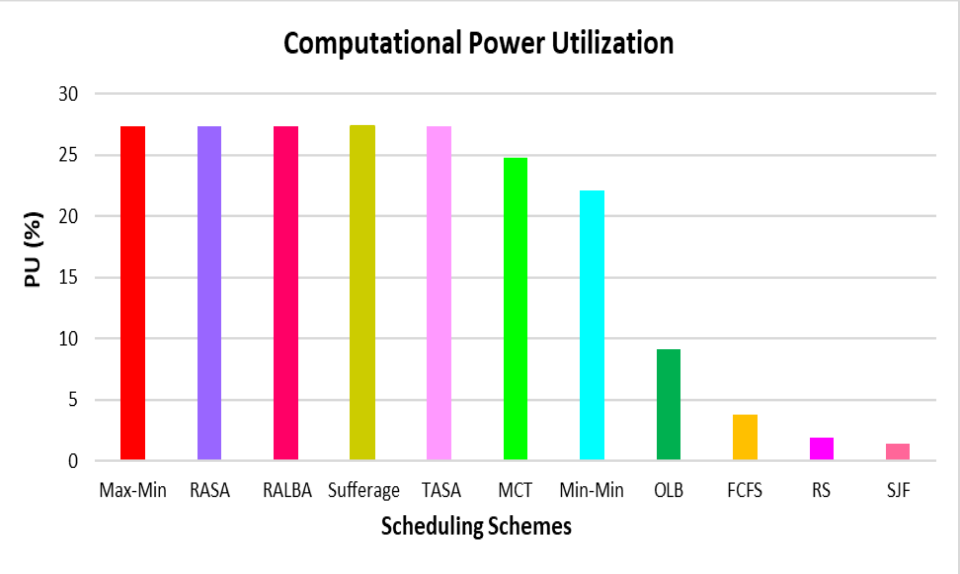

FIGURE 5.26: Computational Power Utilization for 1000 Cloudlets Using RanDS

and RALBA has attained same resource utilization on average while RS and SJF performed worst on the basis of resource utilization.

5.8 Analysis of Best Performing Scheme

This experiment is conducted to analyze the impact of above-discussed outperforming scheme on resource utilization. As RALBA outperforms other scheduling

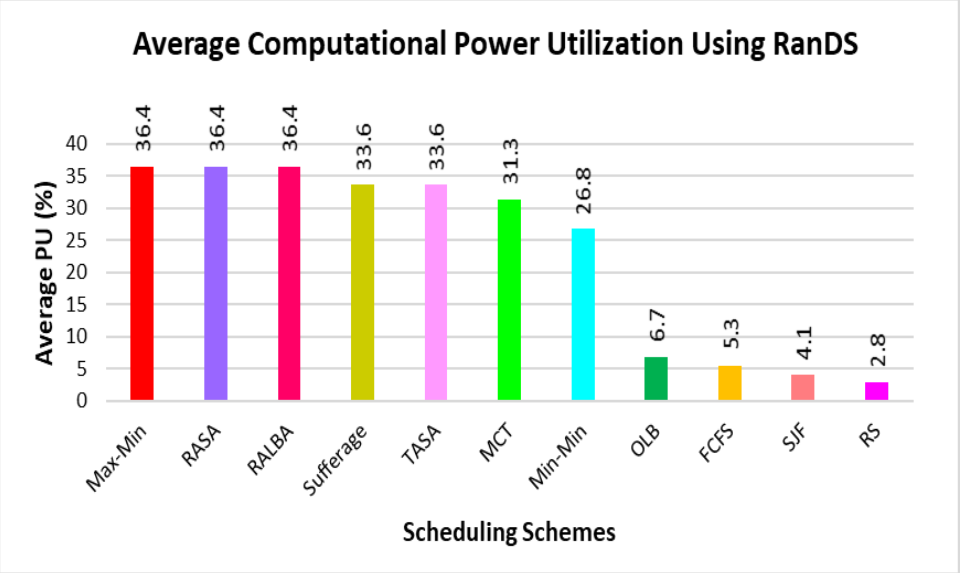

FIGURE 5.27: Average Computational Power Utilization Using RanDS

schemes in previously discussed results, so this experiment is conducted to analyze the impact of resource utilization of RALBA by changing the computational power of VMs. This experiment is comprised of two scenarios. In these two scenarios we have increased the non-uniformity of VMs.

5.8.1 Scenario A

In this scenario , number of VMs is 30, out of which 9 VMs are considered slowest machines with computational power (100 MIPS). Other 21 VMs have different computational power. In both scenarios, three files are selected from GoCJ dataset having numbers of cloudlets: 100, 500, and, 1000, (each having different MIs). The statistics of VMs for scenario A are presented in Figure 5.2.

5.8.2 Scenario A: Resource Utilization of RALBA for 100 Cloudlets

In Table 5.2, MIPS and computational power utilization (P_U) of each VM for RALBA are listed when we have 100 cloudlets. The P_U of RALBA on each VM increases and decreases based on computational power of VMs, as shown in Figure

TABLE 5.2: P_U of RALBA for 100 cloudlets in Scenario A

VMID	**MIPS**	P_U (%)	**VMID**	**MIPS**	P_U (%)
1	100	0%	16	1000	96.37%
2	100	0%	17	1250	86.79%
3	100	59.55%	18	1500	99.98%
4	500	81.42%	19	1750	85.27%
5	750	83.44%	20	4000	95.04%
6	1000	84.89%	21	100	86.62%
7	1250	95.64%	22	100	0%
8	1500	98.82%	23	100	86.62%
9	1750	84.76%	24	500	84.02%
10	4000	87.81%	25	750	82.87%
11	100	32.48%	26	1000	83.16%
12	100	86.62%	27	1250	86.79%
13	100	0%	28	1500	86.64%
14	500	83.16%	29	1750	96.40%
15	750	82.87%	30	4000	88.11%
Average P_U					**73.54%**

5.28. We have 9 VMs whose computational power is 100 MIPS and these VMs are slowest. Out of these 9 VMS, RALBA has scheduled tasks on 5 VMs and 4 VMs remains idle in this scenario. RALBA heuristic produces a VM-level load-imbalance by overloading the faster VMs and under-utilizing the slower VMs (i.e., the VM1, VM2, VM13, and VM22 remains idle) as shown in Figure 5.28. RALBA has attained only 73.5% resource utilization in this scenario, while when the number of slow VMs were 3 as discussed in above section, then the resource utilization of RALBA was 92.9% for 100 number of cloudlets. We observed that, RALBA gives better resource utilization when we have relatively fast VMs, as shown in Figure 5.28. Similarly, the makespan for 100 cloudlets in above result was almost 386 seconds but now the makespan increased (471.76 seconds) and throughput also decreased due to low resource utilization. Based on these results, we conclude that when we have a greater number of slower VMs and relatively a lesser number of cloudlets, then the performance of RALBA is significantly changed.

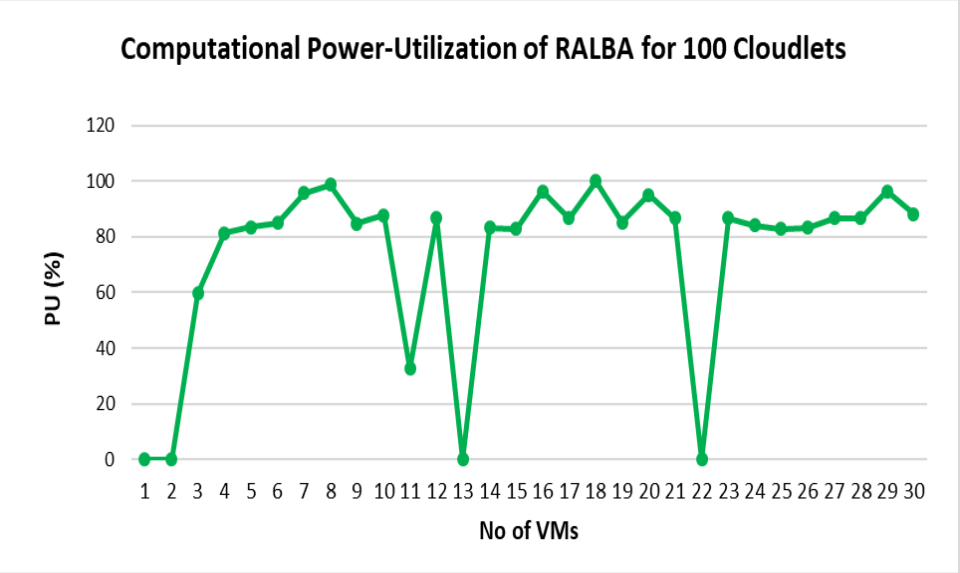

FIGURE 5.28: P_U of RALBA for 100 Cloudlets in Scenario A

5.8.3 Scenario A: Resource Utilization of RALBA for 500 Cloudlets

The computational power utilization of RALBA for 500 cloudlets at each VM is shown in Figure 5.29 and Table 5.3. In this experiment, RALBA has almost similar resource utilization for 500 cloudlets as discussed in section 4.2.3. But makespan of RALBA for 500 cloudlets increases and throughput decreases because we have a greater number of slow VMs as compared to above experimental setup. In this experiment, RALBA has achieved above 95% resource utilization for all VMs as given in Table 5.3

5.8.4 Scenario A: Resource Utilization of RALBA for 1000 Cloudlets

The computational power utilization of RALBA for 1000 cloudlets at each VM is shown in Figure 5.30 and Table 5.4. In this experiment, RALBA attained 98% resource utilization for 1000 cloudlets and it is almost similar to the previous value of resource utilization as discussed in section 4.2.3. But makespan of RALBA for

TABLE 5.3: P_U of RALBA for 500 cloudlets in Scenario A

VMID	MIPS	P_U (%)	VMID	MIPS	P_U (%)
1	100	97.83%	16	1000	99.80%
2	100	97.83%	17	1250	99.59%
3	100	97.83%	18	1500	99.99%
4	500	98.94%	19	1750	99.60%
5	750	99.18%	20	4000	99.84%
6	1000	99.51%	21	100	96.81%
7	1250	98.92%	22	100	97.83%
8	1500	99.85%	23	100	96.81%
9	1750	98.93%	24	500	99.35%
10	4000	99.46%	25	750	99.04%
11	100	97.83%	26	1000	99.40%
12	100	96.81%	27	1250	98.72%
13	100	97.83%	28	1500	99.37%
14	500	99.35%	29	1750	99.28%
15	750	99.62%	30	4000	99.38%
Average P_U					**98.82%**

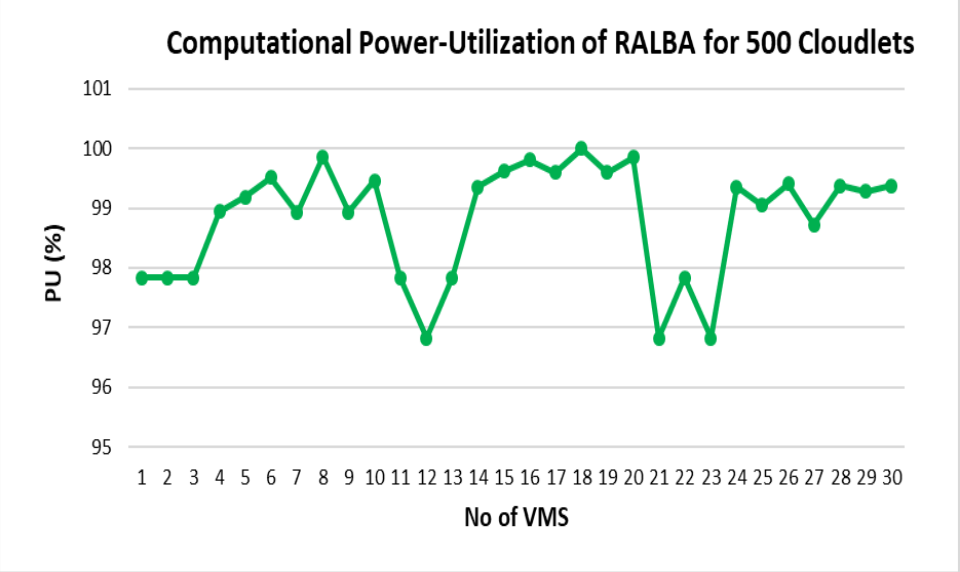

FIGURE 5.29: P_U of RALBA for 500 Cloudlets in Scenario A

1000 cloudlets increase and throughput decrease as given in Table 4.8 because we have a greater number of slow VMs as compared to above setup.

TABLE 5.4: P_U of RALBA for 1000 cloudlets in Scenario A

VMID	MIPS	P_U (%)	VMID	MIPS	P_U (%)
1	100	98.11%	16	1000	99.94%
2	100	97.61%	17	1250	98.43%
3	100	97.10%	18	1500	99.99%
4	500	97.96%	19	1750	98.74%
5	750	98.67%	20	4000	98.65%
6	1000	98.40%	21	100	97.10%
7	1250	98.63%	22	100	97.61%
8	1500	98.57%	23	100	97.10%
9	1750	98.67%	24	500	97.96%
10	4000	98.73%	25	750	98.60%
11	100	97.60%	26	1000	98.67%
12	100	97.10%	27	1250	98.44%
13	100	98.11%	28	1500	98.51%
14	500	97.96%	29	1750	98.48%
15	750	98.60%	30	4000	98.60%
Average P_U					**98.82%**

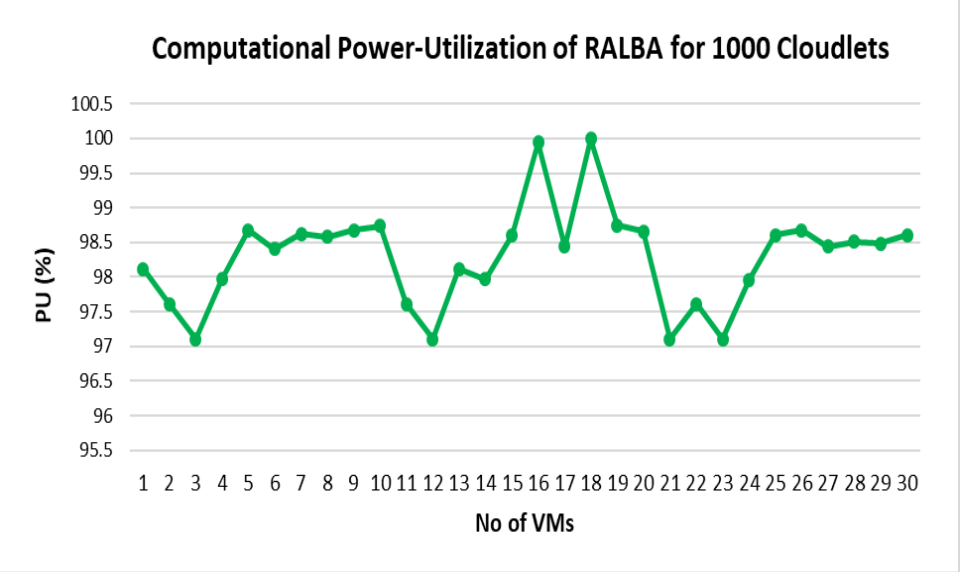

FIGURE 5.30: P_U of RALBA for 1000 Cloudlets in Scenario A

5.8.5 Scenario B

In this scenario, the computational power for slowest VMs is changed from 100 MIPS to 500 MIPS. The statistics of VMs for simulating Scenario B are presented in Figure 5.3.

TABLE 5.5: P_U of RALBA for 100 cloudlets in Scenario B

VMID	MIPS	P_U (%)	VMID	MIPS	P_U (%)
1	500	84.18%	16	750	81.94%
2	750	79.69%	17	1000	86.71%
3	1000	76.61%	18	1250	85.75%
4	1250	82.63%	19	1500	84.37%
5	1500	84.09%	20	1750	84.21%
6	1750	84.21%	21	4000	95.66%
7	4000	86.98%	22	500	83.06%
8	500	83.96%	23	750	83.06%
9	750	82.87%	24	1000	81.10%
10	1000	79.41%	25	1250	83.06%
11	1250	86.20%	26	1500	83.24%
12	1500	83.65%	27	1750	84.21%
13	1750	80.41%	28	4000	87.09%
14	4000	99.97%	29	500	84.18%
15	500	81.94%	30	750	86.80%
Average P_U					**84.09%**

5.8.6 Scenario B: Resource Utilization of RALBA for 100 Cloudlets

The computational power utilization of RALBA for 100 cloudlets at each VM is shown in Figure 5.31 and Table 5.5. It can be observed from Figure 5.31, by changing the computational power of VMs, utilization of RALBA also affects. When we have 9 VMs whose computational power was 100 MIPS in above scenario, then the average resource utilization of RALBA was 73.5%, but now RALBA has 84% average resource utilization by changing the computational power of VMs from 100 to 500 MIPS. However, RALBA gives better resource utilization when we have relatively fast VMs, as shown in Figure 5.31. Similarly, the makespan and throughput for 100 cloudlets in this scenario is relatively better than the results of scenario A, as given in Table 5.5.

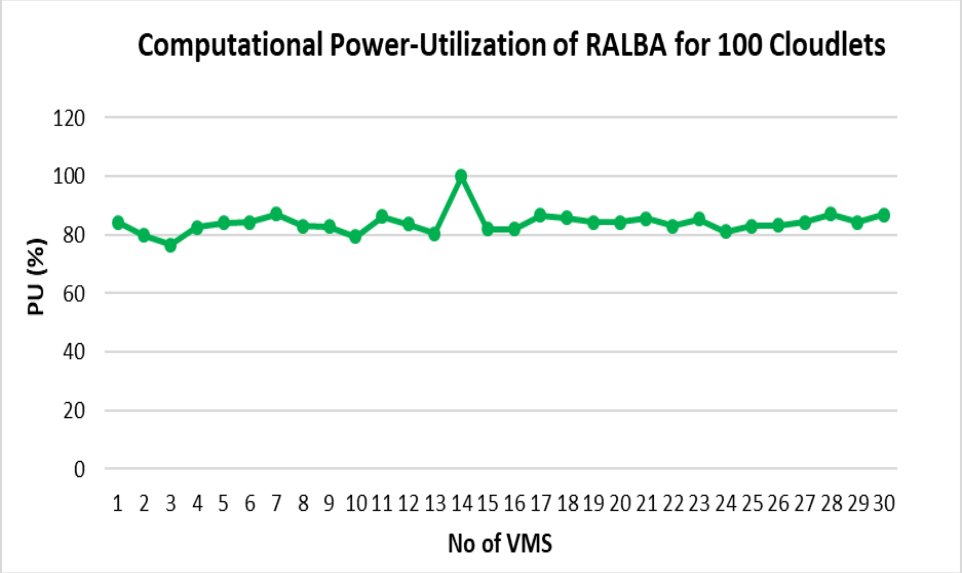

FIGURE 5.31: P_U of RALBA for 100 Cloudlets in Scenario B

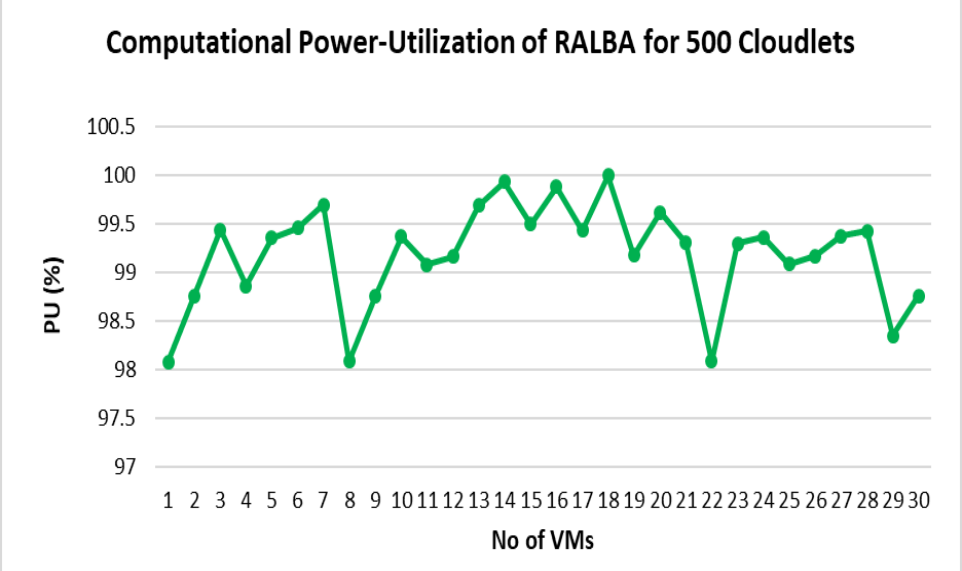

FIGURE 5.32: P_U of RALBA for 500 Cloudlets in Scenario B

5.8.7 Scenario B: Resource Utilization of RALBA for 500 Cloudlets

The resource utilization of RALBA for 500 cloudlets at each VM is shown in Figure 5.32 and Table 5.6. In this experiment, RALBA attained 99% average resource utilization for 500 cloudlets. As shown in Figure 5.32, RALBA consumes 98% computational power of each VM. The makespan and throughput also improves in this scenario as compared to scenario A.

TABLE 5.6: P_U of RALBA for 500 cloudlets in Scenario B

VMID	MIPS	P_U (%)	VMID	MIPS	P_U (%)
1	500	98.07%	16	750	99.87%
2	750	98.74%	17	1000	99.42%
3	1000	99.43%	18	1250	99.99%
4	1250	98.85%	19	1500	99.17%
5	1500	99.34%	20	1750	99.61%
6	1750	99.45%	21	4000	99.29%
7	4000	99.69%	22	500	98.07%
8	500	98.07%	23	750	99.28%
9	750	98.75%	24	1000	99.35%
10	1000	99.36%	25	1250	99.08%
11	1250	99.07%	26	1500	99.16%
12	1500	99.15%	27	1750	99.37%
13	1750	99.68%	28	4000	99.41%
14	4000	99.92%	29	500	98.34%
15	500	99.42%	30	750	98.75%
Average P_U					**99.17%**

5.8.8 Scenario B: Resource Utilization of RALBA for 1000 Cloudlets

The resource utilization of RALBA for 1000 cloudlets at each VM is shown in Figure 5.33 and Table 5.7. In this experiment, RALBA attained 99.6% resource utilization for 1000 cloudlets and it is relatively better than the results of scenario A. Based on these results we conclude that with the increasing number of cloudlets and greater number of fast VMs, resource utilization of RALBA also increases. But when we have relatively small number of cloudlets and greater number of slower VMs then the resource utilization of RALBA is biased.

As per our results, RALBA has the lowest makespan, highest throughput, and resource utilization while RS showed the highest makespan, lowest throughput, and resource utilization by using the GoCJ dataset. With RanDS dataset, RALBA, RASA, and Max-Min showed the same amount of lowest makespan, highest throughput, and resource utilization. One significant result observed that the maximum resource utilization was 98% on the GoCJ dataset and 36.4% by using RanDS for the best performing scheme.

TABLE 5.7: P_U of RALBA for 1000 cloudlets in Scenario B

VMID	MIPS	P_U (%)	VMID	MIPS	P_U (%)
1	500	98.90%	16	750	99.49%
2	750	99.88%	17	1000	99.36%
3	1000	99.43%	18	1250	99.65%
4	1250	99.55%	19	1500	99.77%
5	1500	99.63%	20	1750	99.95%
6	1750	99.95%	21	4000	99.68%
7	4000	99.99%	22	500	99.62%
8	500	98.91%	23	750	99.47%
9	750	99.43%	24	1000	99.57%
10	1000	99.88%	25	1250	99.69%
11	1250	99.75%	26	1500	99.61%
12	1500	99.73%	27	1750	99.79%
13	1750	99.95%	28	4000	99.77%
14	4000	99.99%	29	500	99.04%
15	500	99.49%	30	750	99.79%
Average P_U					**99.62%**

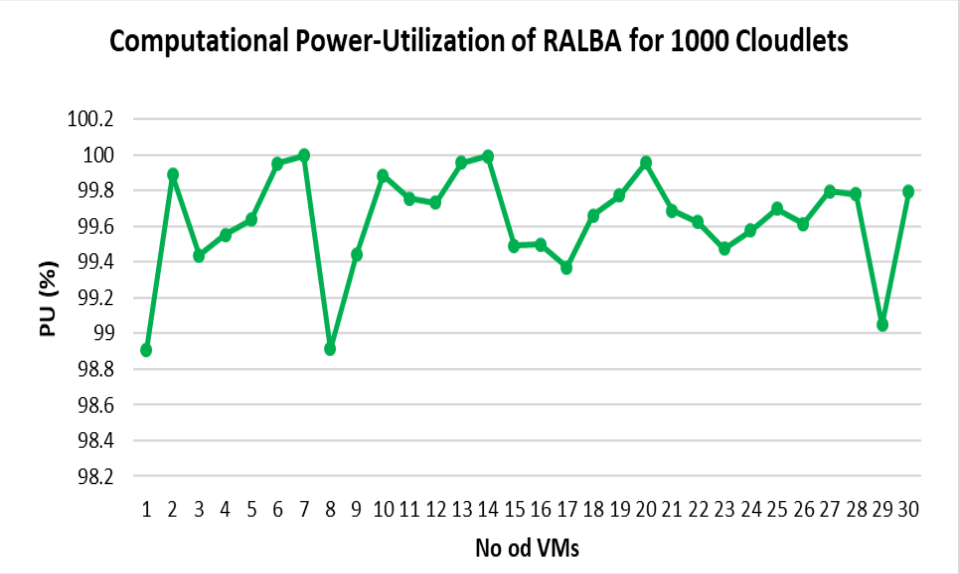

FIGURE 5.33: P_U of RALBA for 1000 Cloudlets in Scenario B

TABLE 5.8: Comparative Table of Scenario A and B for RALBA

For 100 Cloudlets	**Scenario A Results**	**Scenario B Results**
Makespan (sec)	461.7	356.3
Throughput (jobs/sec)	0.2100	0.2694
Throughput (MI/sec)	29256.25	37910.78
ARUR (0-1)	0.73	0.84
P_U (%)	73.5	84.1
For 500 Cloudlets	**Scenario A Results**	**Scenario B Results**
Makespan (sec)	1972.8	1478.5
Throughput (jobs/sec)	0.2483	0.3327
Throughput (MI/sec)	32949.55	43964.71
ARUR (0-1)	0.98	0.99
P_U (%)	98.8	99.1
For 1000 Cloudlets	**Scenario A Results**	**Scenario B Results**
Makespan (sec)	3964.7	2938.0
Throughput (jobs/sec)	0.2476	0.3355
Throughput (MI/sec)	32703.70	44131.82
ARUR (0-1)	0.98	0.99
P_U (%)	98.2	99.6

Chapter 6

Analysis and Discussion

In this chapter, comparitive analysis and discussion of simulation results about above-mentioned scheduling schemes is provided.

6.1 Makespan-Based Analysis

In Figure 5.5, scheduling schemes are compared on the basis of makespan for GoCJ benchmark workload. RALBA has completed the execution of all the tasks earlier than the other schemes, as shown in Figure 5.5. It is observed that by changing the dataset and computational power of resources it affects the performance of different schemes in terms of makespan, throughput, and resource utilization. Figure 5.8 shows the average makespan result of eleven scheduling schemes by using RanDS. RALBA, RASA, and Max-Min produce the same makespan on RanDS while on GoCJ dataset, RALBA outperforms the other schemes as shown in Figure 5.5. The makespan of RALBA is almost equivalent to RASA and Max-Min on average as shown in Figure 5.8. On the other hand, the makespan of Max-Min on GoCJ dataset for 500 and 1000 cloudlets is much larger than RALBA and RASA. The reason is that, Max-Min mostly performs better in the scenario when there is a large number of small-sized jobs with a fewer larger jobs. Max-Min algorithm assigns task on resources where larger tasks have more priority than smaller

tasks. Our experimental results show that after RALBA and RASA, the Sufferage scheduling scheme has lower makespan, higher throughput and better resource utilization as compared to the TASA, Min-Min, Max-Min, MCT, OLB, FCFS, SJF and RS schemes by using GoCJ dataset. Reduced makespan and higher resource utilization is achieved due to the selection of a suitable VM for cloudlet scheduling. The suitable VM means which will suffer most in terms of makespan. By assigning suitable machines to the tasks that have the highest sufferage values among all contending tasks, the Sufferage algorithm reduces the overall completion time.

The scheduling mechanism of TASA is based on Sufferage and Min-Min schemes. Therefore, TASA provides reduced makespan as compared to Min-Min, Max-Min, MCT, OLB, RS, SJF and FCFS. On the other hand, RASA uses the Min-Min strategy to execute small tasks before the large ones and applies the Max-Min strategy to avoid delays in the execution of large tasks and to support concurrency in the execution of large and small tasks, therefore it gives mostly good results. Sufferage and RASA have performed almost identical to RALBA due to resource-aware scheduling mechanism. However, RALBA also considers load-balance factor for scheduling; therefore, it has produced higher resource utilization as compared to RASA and Sufferage on GoCJ dataset. But it is observed that the RALBA scheduling mechanism produces a schedule that results in slower machines being idle (for the small-sized job pool), however, a gradual improvement in resource utilization was observed for the large size job pool and faster VMs (as discussed above in scenario 1 and 2).

The experimental results have revealed that on GoCJ dataset the Min-Min algorithm has lower makespan than the Max-Min due to a larger number of shorter size jobs in the dataset. Alternatively, Max-Min has achieved lower makespan on RanDS and has significantly improved resource utilization on both GoCJ and RanDS as compared to the Min–Min which overloads faster VMs with small-sized tasks. The makespan given by the OLB is larger than the makespan obtained by MCT, Min-Min, and Max-Min on both datasets because OLB assigns a task to the machine that becomes ready next, without considering the execution time of

the task onto that machine. If multiple machines become ready at the same time, then one machine is arbitrarily chosen by OLB.

As shown in Figure 5.5, RS has performed worse than all other scheduling schemes for GoCJ dataset because, RS arbitrarily assigns a job to a randomly selected VM. However, the unfair scheduling method used by the RS leads to the poor makespan.On the other hand, OLB has achieved largest makespan on RanDS the reason is that, OLB keeps all machines as busy as possible regardless of considering the job's execution time on that particular machine. As a result, OLB scheduling scheme mostly results in poor makespan.

6.2 Throughput-Based Analysis

Figure 5.11 shows that, for GoCJ dataset the RALBA has achieved 1.2, 1.6, 2.7, 15.6, 23.0, 22.7, and 225.3% higher throughput on average as compared to RASA, Sufferage, TASA, MCT, Min-Min, Max-Min, and OLB schemes, respectively. As shown in Figures 5.17 and 5.14 Max-Min, RALBA and RASA have same throughput, similarly TASA and Sufferage attained same throughput for 500 cloudlets using RanDS, while on GoCJ dataset RALBA outperforms these scheduling schemes on the basis of throughput. RALBA attained (37205.58 MI/second) throughput for 500 cloudlets using GoCJ dataset, while using RanDS RALBA has attained low throughput i.e., 17186.88 MI/second. Similarly, the performance of other scheduling schemes also significantly changed by changing the dataset.

It is observed that the RALBA, RASA, and Max-Min produce the same throughput using RanDS but the performance of these scheduling schemes becomes poor by increasing the number of cloudlets (i.e., from 500 to 1000). On GoCJ dataset, RALBA outperforms the other schemes as shown in Figure 5.11. Using RanDS, Max-Min has same amount of throughput as RALBA and RASA while on GoCJ dataset the throughput of RALBA and RASA was better than Max-Min. It is observed that the Sufferage and TASA schemes have almost same throughput on

both GoCJ and RanDS. However, SJF and FCFS have achieved least throughput while RS performed worse than all other scheduling schemes on average using RanDS as given in Figure 5.17.

The Experiments have revealed that MCT scheduling scheme gives the good results as compared to the Min-Min, OLB, RS, SJF, and, FCFS. This is because the MCT assigns the job to the most appropriate VM that is able to accomplish the job within the given constraints. Furthermore, it can be noted that RS, SJF, and FCFS performs worst based on throughput in all experiments.

6.3 Resource Utilization-Based Analysis

In Table 6.1, the results showed that the RALBA heuristic attained higher resource utilization (i.e., 1.54%, 3.03% and 1.85%) higher, as compared to the RASA, TASA and Sufferage for the execution of GoCJ workload. However, RASA and Max-Min has attained same resource utilization as RALBA for RanDS. The Max-Min algorithm attained 8.33% higher resource utilization as compared to Sufferage and TASA for RanDS while on GoCJ dataset Sufferage attained 21.15% higher resource utilization than Max-Min and 1.14%, higher than TASA. Moreover, Max-Min achieved 4.99% higher resource utilization as compare to OLB for GoCJ dataset, while on RanDS, Max-Min has achieved 35.82% and 439.2% higher resource utilization as compare to Min-Min and OLB.

Furthermore, it is observed that MCT has achieved (7.38% and 7.25%) higher resource utilization, as compared to Max-Min and Min-Min for GoCJ dataset, while on RanDS, Max-Min has better resource utilization (i.e., 16.29%, 35.82%) than MCT and Min-Min. The Sufferage and RASA have achieved higher resource utilization of 1.14% and 1.46%, respectively, as compared to TASA for GoCJ workload. This minor improvement in resource utilization by RASA over TASA is due to the lower resource utilization produced by the Min-Min scheduling scheme.

TABLE 6.1: Percentage of Average PU of Scheduling Schemes

Algorithms	GoCJ Dataset	RanDS
MCT	85.8 %	31.3 %
Min-Min	80 %	26.8 %
Max-Min	79.9 %	36.4 %
Sufferage	96.8 %	33.6 %
RALBA	98.6 %	36.4 %
RASA	97.1 %	36.4 %
TASA	95.7 %	33.6 %
OLB	76.1 %	6.75 %
RS	20.7 %	2.88 %
SJF	6.5 %	4.16 %
FCFS	6.1 %	5.38 %

Figure 5.22 shows the mean ARUR based experimental results for the execution of RanDS of all scheduling schemes. Using RanDS, Max-Min has attained 6.9, 10, 21.3, 21.4, 43.9, and 131.5% higher resource utilization as compared to the RASA, RALBA, Sufferage, TASA, MCT, and Min-Min schemes, respectively. On the other hand, on GoCJ dataset, Figure 5.19 shows that the RALBA has attained 7.2, 4.9, 9.2, 18.3, 50.7, 21.5, and 28.3% higher resource utilization than RASA, Sufferage, TASA, Max-Min, Min-Min, MCT, and OLB schemes, respectively. Similarly, using RanDS, RS and OLB performed worst while on GoCJ dataset, RS and SJF has lowest resource utilization.

During the implementation of Min-Min algorithm, it is noticed that mostly all VMs based on 100 MIPS and a few of VMS with 500 MIPS remain idle when the Min-Min algorithm is used to schedule the GoCJ workload. Additionally, the Min-Min overloads the fastest VMs with shorter tasks (based on 4000 MIPS). The Max-Min scheduling, on the other hand, produces better distribution of workload compared to the Min-Min; however, only a few VMs (both slower and faster) are overloaded. The Max-Min algorithm overcomes the Min-Min imbalance due to the presence of a few large cloudlets that suits the Max-Min scheduling.

The RALBA, RASA, Sufferage, TASA and Max-Min produce comparatively a load-balanced schedule. Among these algorithms, RALBA and RASA produces

the highest resource utilization because of the resource-aware mechanism. However, an interesting observation is that the few VMs based on 100 MIPS remain idle in case of scenario 1 for RALBA. In addition, the recourse-aware method used by the RALBA often creates a large load imbalance when scheduling is done by using slow VMs and a smaller number of cloudlets (i.e. 100 cloudlets), as shown in Figure 5.28.

Furthermore, due to the inherent usage of Min-Min and Max-Min alternatively, in scheduling process, RASA produces better results. However, the slowest VMs still idle due to the inherent use of Min-Min algorithm and the fastest VMs remain overloaded when a small number of cloudlets are scheduled by the RASA. The alternative Min-Min and Max-Min methods used by the RASA ensures a fair scheduling for both large and small-sized cloudlets. Likewise, in the eleven scheduling schemes implemented, the TASA strategy mostly provides the smallest makespan. Moreover, due to the use of Min-Min in alternative scheduling steps, most of the slower VMs (i.e. 100 MIPS) are idle for the small-sized job pool. On the other hand, TASA overcomes the load imbalance (caused by the Min-Min algorithm) to some extent by using the inherent method of Sufferage in alternative scheduling steps. A better resource utilization is provided by the Sufferage scheduling scheme; however, very few slow VMs (with 100 MIPS) remain idle.

The results show that there is sufficient possibility of unbalanced workload distribution among VMs, even a scheduling scheme achieves an improved value of resource utilization. It is empirically evident that most of the existing scheduling schemes produce a reduced makespan with a higher throughput. However, often these algorithms result in a load imbalanced scheduling. We suggest that the scheduling schemes should address the load-balancing issue to attain resource utilization for cloud computing resources.

Chapter 7

Conclusion and Future Direction

7.1 Conclusion

An empirical analysis is carried out by studying eleven prominent static scheduling schemes against makespan, throughput and resource utilization. The experiments are conducted on CloudSim simulation tool by using two datasets, a third-party GoCJ benchmark dataset and other is our own prepared dataset named as RanDS. RanDS is used to evaluate the worst case scenario. Both datasets are based on static, non-preemptive and independent tasks. It was observed that the partial definitions of throughput and resource utilization are used in the literature which we have studied. As, throughput is referred as the number of tasks completed per unit time. This definition of throughput is for the tasks of nearly equal size. However, if the task size differs significantly, then this definition of throughput does not truly reflect. After examining the definition we have formulated a complete and thorough definition of throughput which incorporates the size of the tasks as follows:
Throughput is referred as the total size of all cloudlets (MIs) completed per unit time, and is expressed in Equation 3.4.

Similarly, ARUR is used in previous studies to compute the resource utilization. ARUR is useful when we have the same computational power of VMs but when the

computational power of VMs differ then ARUR does not truly reflect that which VM is actually used. We have formulated a new definition to compute resource utilization which covers all aspects including non-even computational power of VMs as follows:
P_U is a performance metric which shows the overall utilization of the system, and is expressed in Equation 3.6

It was observed that the RALBA was at the top with lowest makespan, highest throughput, and resource utilization by using both GoCJ and RanDS dataset. However, the resource utilization of RALBA was observed as 36.4% by using RanDS, which proves that still there is a room to devise a scheme with improved resource utilization.

7.2 Future Direction

The scheme which is best reported in the literature has achieved 36.4% resource utilization as per our results by using RanDS. In a consequent to this work, there is a room to devise a resource-aware scheduling scheme with improved resource utilization.

Bibliography

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

[2] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "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.

[3] 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.

[4] S. Taherian Dehkordi and V. Khatibi Bardsiri, "Tasa: A new task scheduling algorithm in cloud computing," *Journal of Advances in Computer Engineering and Technology*, vol. 1, no. 4, pp. 25–32, 2015.

[5] 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.

[6] 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.

[7] 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.

[8] N. Chowdhury, K. A. Uddin, S. Afrin, A. Adhikary, and F. Rabbi, "Performance evaluation of various scheduling algorithm based on cloud computing system," *Asian Journal of Research in Computer Science*, pp. 1–6, 2018.

[9] R. Aluri, S. Mehra, A. Sawant, P. Agrawal, and M. Sohani, "Priority based non-preemptive shortest job first resource allocation technique in cloud computing," *International Journal of Computer Engineering and Technology (IJCET)*, vol. 9, no. 2, pp. 132–139, 2018.

[10] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," *National Institute of Standards and Technology: Gaithersburg,*, 2011.

[11] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, 2012.

[12] A. Zahariev, "Google app engine," *Helsinki University of Technology*, pp. 1–5, 2009.

[13] D. Chappell *et al.*, "Introducing the azure services platform," *White paper, Oct*, vol. 1364, no. 11, 2009.

[14] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of network and computer applications*, vol. 34, no. 1, pp. 1–11, 2011.

[15] N. Almezeini and A. Hafez, "Review on scheduling in cloud computing," *International Journal of Computer Science and Network Security*, vol. 18, no. 2, pp. 108–111, 2018.

[16] P. Suri and S. Rani, "Design of task scheduling model for cloud applications in multi cloud environment," in *International Conference on Information, Communication and Computing Technology*. Springer, 2017, pp. 11–24.

[17] G. U. Srikanth, V. U. Maheswari, A. Shanthi, and A. Siromoney, "Task scheduling model," *Indian Journal of Science and Technology*, vol. 8, p. 33, 2015.

[18] D. A. Alboaneen, H. Tianfield, and Y. Zhang, "Glowworm swarm optimisation based task scheduling for cloud computing," in *Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing*, 2017, pp. 1–7.

[19] A. Thomas, G. Krishnalal, and V. J. Raj, "Credit based scheduling algorithm in cloud computing environment," *Procedia Computer Science*, vol. 46, pp. 913–920, 2015.

[20] D. Agarwal, S. Jain *et al.*, "Efficient optimal algorithm of task scheduling in cloud computing environment," *International Journal of Computer Trends and Technology (IJCTT)*, vol. 9, no. 7, pp. 344–349, 2014.

[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] S. Biradar and D. Pawar, "A review paper of improving task division assignment using heuristics," *International Journal of Science and Research (IJSR)*, vol. 4, no. 1, pp. 609–613, 2015.

[23] S. S. Brar and S. Rao, "Optimizing workflow scheduling using max-min algorithm in cloud environment," *International Journal of Computer Applications*, vol. 124, no. 4, 2015.

[24] P. Banga and S. Rana, "Heuristic based independent task scheduling techniques in cloud computing: a review," *International Journal of Computer Applications*, vol. 166, no. 1, pp. 0975–8887, 2017.

[25] I. De Falco, U. Scafuri, and E. Tarantino, "Two new fast heuristics for mapping parallel applications on cloud computing," *Future Generation Computer Systems*, vol. 37, pp. 1–13, 2014.

[26] S. Atiewi, S. Yussof, and M. Ezanee, "A comparative analysis of task scheduling algorithms of virtual machines in cloud environment," *Journal of Computer Science*, vol. 11, no. 6, p. 804, 2015.

[27] S. A. Ali and M. Alam, "A relative study of task scheduling algorithms in cloud computing environment," in *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE, 2016, pp. 105–111.

[28] 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.

[29] P. G. Gopinath and S. K. Vasudevan, "An in-depth analysis and study of load balancing techniques in the cloud computing environment," *Procedia Computer Science*, vol. 50, pp. 427–432, 2015.

[30] G. Patel, R. Mehta, and U. Bhoi, "Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing," *Procedia Computer Science*, vol. 57, pp. 545–553, 2015.

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

[32] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen *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.

[33] O. Elzeki, M. Reshad, and M. Elsoud, "Improved max-min algorithm in cloud computing," *International Journal of Computer Applications*, vol. 50, no. 12, 2012.

[34] U. Bhoi, P. N. Ramanuj *et al.*, "Enhanced max-min task scheduling algorithm in cloud computing," *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, vol. 2, no. 4, pp. 259–264, 2013.

[35] S. A. Hamad and F. A. Omara, "Genetic-based task scheduling algorithm in cloud computing environment," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 4, pp. 550–556, 2016.

[36] J. Maipan-uku, A. Mishra, A. Abdulganiyu, and A. Abdulkadir, "An extended min-min scheduling algorithm in cloud computing," *i-manager's Journal on Cloud Computing*, vol. 5, no. 2, p. 20, 2018.

[37] M.-L. Chiang, H.-C. Hsieh, W.-C. Tsai, and M.-C. Ke, "An improved task scheduling and load balancing algorithm under the heterogeneous cloud computing network," in *2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST)*. IEEE, 2017, pp. 290–295.

[38] T. Kokilavani, D. G. Amalarethinam *et al.*, "Load balanced min-min algorithm for static meta-task scheduling in grid computing," *International Journal of Computer Applications*, vol. 20, no. 2, pp. 43–49, 2011.

[39] J. Maipan-uku, I. Rabiu, and A. Mishra, "Immediate/batch mode scheduling algorithms for grid computing: A review," *International Journal of Research-GRANTHAALAYAH*, 2017.

[40] 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.

[41] 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.

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

[43] 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.

[44] Z. Liu and S. Cho, "Characterizing machines and workloads on a google cluster," in *2012 41st International Conference on Parallel Processing Workshops*. IEEE, 2012, pp. 397–403.

Appendix A

Pseudo-Code of MCT Scheduling Scheme

```
1: for all tasks T_i in meta − task M_v do
2:    for all machines m_j do
3:       c_ij=e_ij + r_j
4:    end for
5: end for
6: do until all tasks in M_v are mapped
7:    for each task T_k in M_v find the minimum completion time
8:    find the task T_k with the earliest completion time
9:    assign task T_k to the resource R_j that gives the earliest completion time
10:   delete task T_k from M_v
11:   update r_j
12:   update c_ij for all i
13: end do
```

FIGURE A1: Pseudo-Code of MCT Scheduling Scheme

Appendix B

Pseudo-Code of Min-Min Scheduling Scheme

```
1: for all tasks T_i in meta − task M_v do
2:    for all machines m_j do
3:       c_ij=e_ij + r_j
4:    end for
5: end for
6: do until all tasks in M_v are mapped
7:    for each task in M_v find the earliest completion time obtained by machine
8:    find the task T_k with the minimum earliest completion time
9:    assign task T_k to the machinem_l that gives the earliest completion time
10:   delete task T_k from M_v
11:   update r_l
12:   update c_ij for all i
13: end do
```

FIGURE B1: Pseudo-Code of Min-Min Scheduling Scheme

Appendix C

Pseudo-Code of Sufferage Scheduling Scheme

```
1: for all tasks T_i in meta − task M_v do
2:    for all machines m_j do
3:       c_ij=e_ij + r_j
4:    end for
5: end for
6: do until all tasks in M_v are mapped
7:    mark all machines as unassigned
8:    for each task T_k in M_v
9:       find machine m_j that gives the earliest completion time
10:      sufferage value = second earliest completion time −
         earliest completion time
11:      if machine m_j is unassigned
12:         assign task T_k to the machine m_j, delete T_k from M_v,
            mark m_j assigned
13:      else
14:         if sufferage value of task T_i already assigned to m_j is less than
               the sufferage value of task T_k
15:            unassign T_i, add T_i back to M_v, assign T_k to the machine m_j,
               delete T_k from M_v
16:   end for
17:   update r_j
18:   update c_ij for all i
19: end do
```

FIGURE C1: Pseudo-Code of Sufferage Scheduling Scheme

Appendix D

Pseudo-Code of RASA Scheduling Scheme

```
1: for all tasks T_i in meta − task M_v do
2:    for all machines m_j do
3:       c_ij=e_ij + r_j
4:    end for
5: end for
6: do until all tasks in M_v are mapped
7:    if the number of resources is even then
8:       for each task T_k in M_v
9:          find the earliest completion time obtained by machine
10:         find the task T_k with the maximum earliest completion time
11:         assign task T_k to machine m_l that gives the earliest completion time
12:         delete task T_k from M_v
13:         update r_j
14:         update c_ij for all i
15:      end for
16:   else
17:      for each task T_k in M_v
18:         find the earliest completion time obtained by machine
19:         find the task T_k with the minimum earliest completion time
20:         assign task T_k to machine m_j that gives the earliest completion time
21:         delete task T_k from M_v
22:         update r_j
23:         update c_ij for all i
24:      end for
25:   end if
26: end do
```

FIGURE D1: Pseudo-Code of RASA Scheduling Scheme

Appendix E

Pseudo-Code of TASA Scheduling Scheme

```
1: for all tasks T_i in meta − task M_v do
2:    for all machines m_j do
3:       c_ij=e_ij + r_j
4:    end for
5: end for
6: do until all tasks in M_v are mapped
7:    if the number of resources is even then
8:       for each task T_k in M_v
9:          find the earliest completion time obtained by machine
10:         sufferage value = second earliest completion time −
            earliest completion time
11:         find the task T_k with the maximum sufferage value
12:         assign task T_k to machine m_l that gives the earliest completion time
13:         delete task T_k from M_v
14:         update r_j
15:         update c_ij for all i
16:      end for
17:   else
18:      for each task T_k in M_v
19:         find the earliest completion time obtained by machine
20:         find the task T_k with the minimum earliest completion time
21:         assign task T_k to machine m_l that gives the earliest completion time
22:         delete task T_k from M_v
23:         update r_j
24:         update c_ij for all i
25:      end for
26:   end if
27: end do
```

FIGURE E1: Pseudo-Code of TASA Scheduling Scheme

Appendix F

Pseudo-Code of RALBA Scheduling Scheme

Input: $vmCrMap - set\ of\ VMs\ with\ computation\ ratio$,
$cloudletList - list\ of\ cloudlets\ c_1, c_2, \ldots\ldots\ldots, c_n$
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: **end if**
7: *return* $cloudletVmMap$

SPLL SCHEDULER
Input: $cloudletList - list\ of\ cloudletsc_1, c_2, \ldots\ldots\ldots, c_n$,
$vmList - list\ of\ VMs\ v_1, v_2, \ldots\ldots\ldots, v_m$,
$cloudletVmMap$–*set of cloudlets to VMs mapping by Fill Scheduler*
Output: $cloudletVmMap$ –*set of cloudlets to VMs mapping*
1: **while** $cloudletList.size() \geq 1$
2: $cloudlet = getMaxCloudletVm(cloudletList)$
3: $v = getVmWithEFT(cloudlet, vmList)$
4: $cloudletVmMap.add(cloudlet, v)$
5: $cloudletList.remove(cloudlet)$
6: **end while**
7: *return* $cloudletVmMap$

FILL SCHEDULER

Input: $vmCrMap - set\ of\ VMs\ with\ computation\ ratio$,
$cloudletList - list\ of\ cloudlets\ c_1, c_2, \ldots\ldots\ldots, c_n$
Output: $cloudletVmMap$ –$set\ of\ cloudlets\ to\ VMs\ mapping$

1: $totalLength = 0$
2: $newVShare = 0$
3: $cloudletVmMap = Null$
4: $vmList = getVmList(vmCrMap)$
5: $vShareMap < v, share >= Null$
6: **for all** *cloudlet* in *cloudletList* **do**
7: $totalLength = totalLength + cloudlet.getCloudletLength()$
8: **end for**
9: **for all** v in $vmList$ **do**
10: $vShareMap_v = totalLength * vmCrMap_v$
11: **end for**
12: **while** $getMinCloudlet(cloudletList) \geq getLargeShare(vShareMap)$
13: $v = getLargeShareVm(vShareMap)$
14: $cloudlet = getMaxPCloudletVm(v.cloudletList)$
15: $cloudletVmMap.add(cloudlet, v)$
16: $newVShare = vShareMap.get(v) - cloudlet.getCloudletLength()$
17: $vShareMap.modify(v.newVShare)$
18: $cloudletList.remove(cloudlet)$
19: **end while**
20: $return\ cloudletVmMap$

FIGURE F1: Pseudo-Code of RALBA Scheduling Scheme