Computing and Informatics, Vol. 38, 2019, 525–554, doi: 10.31577/cai_2019_3_525

INVESTIGATION OF CLOUD SCHEDULING ALGORITHMS FOR RESOURCE UTILIZATION USING CLOUDSIM

Altaf Hussain, Muhammad Aleem, Muhammad Azhar Iqbal
Muhammad Arshad Islam

Faculty of Computing
Capital University of Science and Technology
Islamabad, Pakistan
e-mail: aleem@cust.edu.pk

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

Keywords: Distributed computing, scheduling algorithm, high-performance computing, scheduling

Mathematics Subject Classification 2010: 68W15

1 INTRODUCTION

Cloud is a large pool of virtualized resources that are provisioned on-demand in a scalable manner. The efficient job scheduling mostly increases the user's satisfaction, improves the system utilization, reduces the job execution time, and minimizes the energy consumption. Scheduling heuristics are generally classified as static and dynamic. A static scheduling heuristic forms a complete job mapping plan before execution while a dynamic scheduling technique generally relies on the runtime parameters to schedule jobs in a best-effort with the use of resources in a more scalable manner as per user requirements. The static Cloud scheduling heuristics avoid the migration of Virtual Machines (VMs) to avoid communication overheads to reduce the execution time. In addition, most of the static techniques produce good turnaround time and irrefutable Quality of Service (QoS) because of the pre-ensured availability of the computing resources for the workload execution [1]. However, the static scheduling techniques may produce inefficient and lower resource utilization due to runtime changes in workload and computing environment [2].

In Cloud environment, scheduling is employed at two levels:

1. VM scheduling is concerned with the mapping of virtual machines to the physical hosts in a Cloud data-center, and
2. job scheduling is concerned with the assignment of jobs to the virtual machines.

Various metrics are harnessed to determine the performance of job scheduling algorithms. These performance metrics include makespan, throughput, resource utilization, response time, and energy consumption. A crucial aspect of scheduling is to map Cloud jobs in a load balanced manner to reduce the makespan of a job pool. Load balanced mapping refers to a distribution of jobs (among VMs) so that all the VMs accomplish the execution of assigned workload within the approximately same time duration. Importantly, a balanced load ensures improved resource utilization, higher throughput, and lower execution time for a job pool.

Several Cloud scheduling heuristics [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16] have been presented by the scientific community. However, the selection of an appropriate scheduler according to a given environment (jobs computing requirements and the available computing resources) to achieve desired scheduling objectives (such as reduced makespan, higher throughput, etc.) is still a difficult task. Since each heuristic contains different underlying assumptions; therefore, a precise comparison cannot be made. In this regard, we empirically scrutinize and experimentally compare ten state-of-the-art static scheduling heuristics (i.e., Opportunistic Load Balancing (OLB) [1, 10, 13], Proactive Simulation-Based Scheduling and Load Balancing (PSSLB) [17], Proactive Simulation-Based Scheduling and Enhanced Load Balancing (PSSELB) [17], Minimum Completion Time (MCT) [6, 18], Min-Min [7, 8, 9, 10, 11], Load Balance Improved Min-Min (LBIMM) [11, 17], Max-Min [5, 10, 13], Resource-Aware Scheduling Algorithm (RASA) [12, 19], Task-Aware Scheduling Algorithm (TASA) [16], and Sufferage [10, 14, 15]).

In an empirical analysis, Syed Hamid Hussain Madni et al. provides the investigation of First Come First Serve (FCFS), Minimum Execution Time (MET), MCT, Min-Min, Max-Min, and Sufferage scheduling algorithms [3]. Based on their analysis, Madni et al. concluded that the hybridization of these techniques may result in more improved results and overcome the limitations of each other to achieve the optimization of task scheduling in cloud computing [3]. Therefore, in this research work, some hybridized scheduling techniques (i.e. RASA [12, 19], TASA [16], LBIMM [11, 17], PSSLB [17], and PSSELB [17] algorithms) are also considered for performance investigation of resource utilization in cloud computing. Figure 1 shows ten scheduling algorithms; where the techniques on the tail of each arrow are the modified and hybridized techniques based on the scheduling techniques directed by the arrow symbols (i.e., the mechanism of RASA is based on Max-Min and Min-Min, the mechanism of TASA is based on Sufferage and Min-Min, LBIMM is the modified version of Min-Min, PSSLB is the modified version of Max-Min, and PSSELB is the modified version of PSSLB technique).

In this study, we consider the following assumptions for the empirical-based comparison of the employed scheduling heuristics. One such assumption is that a workload is referred to as a collection of independent and compute-intensive tasks (without inter-task data dependencies). The mapping of these tasks is performed statically to minimize the scheduling overhead and to evade job migrations [13]. This empirical study provides a detailed analysis of the scheduling heuristics and insights of the scheduling mechanisms where a higher throughput and reduced execution time is attained; however, a considerable load imbalance is observed in the experimentation. We argue that the existing state-of-the-art static scheduling heuristics should address the load-balancing issue to attain exquisite resource utilization in Cloud computing. A near-optimal resource utilization will produce higher throughput, reduced execution time (for the Cloud job pool), and energy efficient execution.

In summary, we present an analysis of the resource utilization of virtual resources in terms of workload distribution among all the VMs. The empirical investigation reveals that most of the scheduling heuristics are not efficient enough to exploit the full computing capabilities of virtual machines in Cloud infrastructure. This empirical study has highlighted various pressing research gaps that must be overcome

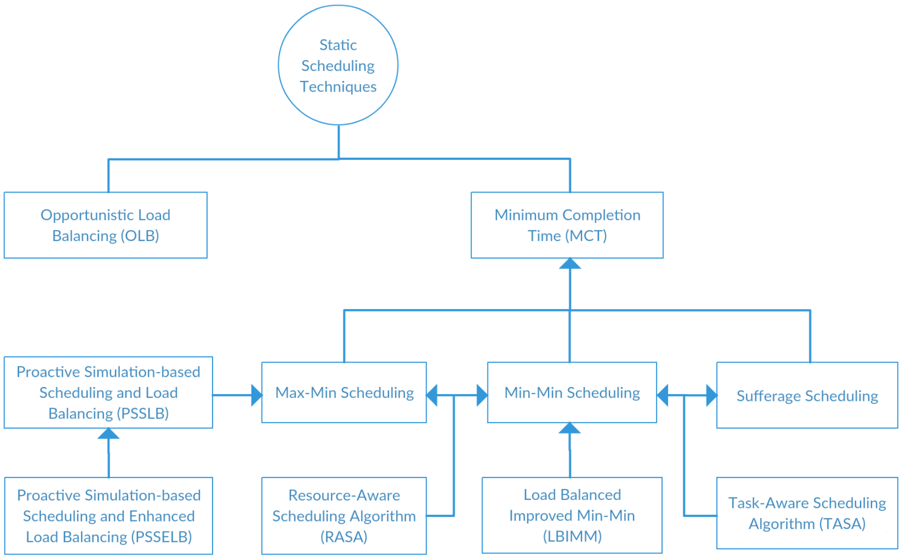

Static Scheduling Techniques
Opportunistic Load Balancing (OLB)
Minimum Completion Time (MCT)
Proactive Simulation-based Scheduling and Load Balancing (PSSLB)
Max-Min Scheduling
Min-Min Scheduling
Sufferage Scheduling
Proactive Simulation-based Scheduling and Enhanced Load Balancing (PSSELB)
Resource-Aware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

Figure 1. Hybridization of Cloud scheduling algorithms

to improve the scheduling performance and to reduce cost at Cloud service provider level (discussed in Section 5). Major contributions of this work include:

1. a critical analysis and synthesis of the existing state-of-the-art static Cloud scheduling algorithms to identify the pros and cons of each algorithm;

2. in-depth performance analysis (in terms of turnaround time, resource utilization, and throughput) and empirical assessment of the existing static scheduling algorithms using two synthetic datasets and one benchmark dataset for Cloud and distributed computing [37];

3. identification of the potential research directions that could assist the scientific community to cope with the challenges pertaining to the static scheduling heuristics for Cloud computing.

The rest of the paper is organized as follows. Section 2 discusses the working semantics of scheduling heuristics. Section 3 illustrates the experimental setup and workload compositions. Section 4 examines the experimental results. The resource utilization and load imbalance in workload distribution caused by scheduling heuristics are presented in Section 5 and the potential research directions are identified. Section 6 concludes this research work.

2 CLOUD SCHEDULING HEURISTICS

In this study, we synthesize and evaluate ten prominent static Cloud scheduling heuristics which are: MCT [3, 6], OLB [1, 10, 13], Min-Min [7, 8, 9, 10, 11], Max-Min [5, 10, 13], Sufferage [10, 14, 15], RASA [12, 20], PSSLB [17], PSSELB [17], LBIMM [11, 17], and TASA [16]. The working of these heuristics is delineated below.

2.1 OLB Algorithm [1, 4, 10, 13]

This scheduling technique assigns each job, in an arbitrary order, to the next available machine regardless of considering the job's execution time on that particular machine. OBL is a simple scheduling scheme having low scheduling overhead and complexity. A major scheduling objective of the OBL scheme is to make all the Cloud machines as busy as possible [13]. However, OBL scheduling heuristic mostly results in poor makespan because it is not resource-aware.

2.2 MCT Algorithm [3, 8, 13]

MCT technique assigns a candidate job to a machine that consumes minimum time for the job [13]. The MCT heuristic examines the current load of machines to find a suitable target machine for job assignment [8]. At each scheduling step, MCT heuristic has to scan all the available machines to find the most appropriate computing resource (i.e., machine producing the minimum completion time for a job). The expensive search mechanism employed by the MCT (at each scheduling step) causes a significant scheduling overhead.

2.3 Min-Min Algorithm [11, 13, 21, 22]

Min-Min scheduling first determines the minimum completion time of all the unallocated jobs and proceeds with the assignment of a job having overall minimum completion time on a certain machine. Both Min-Min and MCT scheduling heuristics rely on the completion time of a job on a certain machine [6]. MCT considers the current job only for scheduling decision (at a certain scheduling step), whereas the Min-Min considers the minimum completion time for all the unallocated jobs (in each scheduling decision). Min-Min scheduling heuristic favors (i.e., schedules first) the small-sized jobs while penalizing (causing delayed execution) for larger jobs [8, 9, 10, 11, 12]. Therefore, Min-Min mostly overloads the faster machines with larger number of small-sized jobs, while the slower machines are assigned fewer but larger jobs. Thus, the larger jobs mapped on slower resources often cause a higher makespan for the execution of the job-pool [23].

2.4 Max-Min Algorithm [3, 21, 24]

Max-Min scheduling first computes the expected minimum completion time for all the jobs and then the job requiring a maximum completion time is assigned to the concerned machine. The scheduling process is repeated until all the un-allocated Cloud jobs are scheduled. To avoid longer response time (for the larger jobs), Max-Min scheduling heuristic selects larger jobs to be executed early [6, 14]. Max-Min heuristic mostly performs better in the scenario when there is a large number of small-sized jobs with a few larger size jobs [11, 14, 15]. The inherent mechanism of both Max-Min and Min-Min heuristics adversely affects the resource utilization in Cloud (as evident in our experimental results presented in Section 4).

2.5 Sufferage Algorithm [11, 21, 22]

Sufferage scheduling calculates the sufferage value for each job. To calculate the sufferage value (i.e., a penalty in terms of longer execution time), the minimum completion time and the second best minimum completion time producing VMs are determined for each job (in each scheduling iteration). Afterward, the job experiencing the highest sufferage value is assigned to the machine (producing minimum completion time for that job). Sufferage heuristic produces good results often with reduced makespan; however, this scheduling mechanism causes higher scheduling overhead (due to the calculation of sufferage value for each job in each scheduling iteration) as compared to OLB, MCT, Max-Min, and Min-Min [18, 21, 22].

2.6 RASA Algorithm [12, 19]

RASA technique contemplates both Min-Min and Max-Min heuristics in the alternate scheduling decisions until all the jobs are scheduled. RASA exploits the merits of both Min-Min and Max-Min to evade corresponding limitations of these two scheduling algorithms in certain cases (as discussed above). Mostly, RASA results in a lower makespan when it considers smaller and larger jobs in alternate scheduling steps [12]. However, RASA penalizes smaller size jobs (causing delayed execution) when the number of larger jobs is higher in the workload [24].

2.7 TASA Algorithm [16, 30]

TASA favors the smaller jobs in the first scheduling step (based on Min-Min heuristic) and finds an appropriate machine for the job (using the Sufferage heuristic) in the second scheduling step [16]. In most of the cases, TASA produces better makespan as compared to other scheduling heuristics such as Min-Min, Max-Min, and OLB [16].

2.8 LBIMM Algorithm [11, 17]

LBIMM [11, 17] assigns jobs to machines using the Min-Min scheduling technique in the first phase. In the subsequent phase, LBIMM finds the smallest job on the most loaded machines and determines its completion time on the other machines. After that, the minimum completion time of that job is compared with makespan. If this time value is less than the makespan, the job is assigned to a new machine and the ready time of both machines are modified. This procedure is repeated for the next smallest job on the most loaded machine, too. The process is repeated until there is no other machine that can produce the minimum completion time for the smallest job on the heavily loaded machine than makespan on another machine. This technique shares a load of heavy machines with the idle or lighter machines. LBIMM produces better makespan and load balancing than Min-Min heuristic.

2.9 PSSLB Algorithm [17]

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

2.10 PSSELB Algorithm [17]

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

Assuming N number of Cloud jobs to be scheduled on M machines, Table 1 presents the summary of strengths, weaknesses, and time-complexity of the eight scheduling heuristics, and the employed simulation tool (by the authors of the mentioned research work).

Heuristics	**Strengths**	**Weaknesses**	**Complexity**	**Tools Used**
OLB [5, 10, 13]	Low complexity, Minimal overhead, keeps machines busy [5, 10]	Load-imbalance, and No-fairness in scheduling [4]	$O(M)$	tGSF Simulator [18], CloudSim Simulator [4].
MCT [3, 8, 13]	Improved makespan than OLB [9], Machine-aware scheduling [24]	Load-imbalance [5, 15], Overloads faster VMs	$O(M \cdot N)$	CloudSim [4], NS Simulator [7]
MinMin [3, 24, 26]	Favors smaller jobs [12], Reduced makespan for smaller jobs [1, 8]	Overloads faster VMs with smaller Jobs [12], Penalizes larger jobs.	$O(M \cdot N^2)$	C-Language [24], Matlab [15], Java-based Simulation
LBIMM [11, 17, 27]	Improved makespan than Min-Min [27], improved resource utilization than Min-Min [11]	A few smaller jobs are penalized while rescheduled [11]	$O(M \cdot N^2)$	Matlab [11]
MaxMin [3, 21, 24]	Favors larger jobs [1, 28], Reduced makespan for larger jobs [22].	Penalizes smaller jobs [16], Load-imbalance for job pool with more larger jobs [12].	$O(M \cdot N^2)$	Matlab and Java-based Simulation [8]
RASA [12, 19, 20]	Fair treatment of larger and smaller jobs [14].	Penalizes smaller jobs in dataset with more larger jobs [23]	$O(M \cdot N^2)$	GridSim [11].
Sufferage [3, 9, 14]	Improved makespan than MCT, Min-Min, and Max-Min [21], Job allocation to appropriate VM [22].	High scheduling overhead due to Sufferage value calculation [10].	$O(M \cdot N^2)$	C++ and Java implementations [14], Matlab
TASA [16, 30]	Improved makespan than Max-Min, Min-Min and RASA [16], Favors smaller jobs.	Load balancing is not considered [16].	$O(M \cdot N^2)$	CloudSim [16].
PSSLB [17]	Reduces completion and response time for larger jobs [17]	Penalizes smaller jobs [17]	$O(M \cdot N^2)$	CloudSim [17].
PSSELB [17]	Improved makespan than PSSLB [17]	Results in load imbalance compared to PSSLB [17]	$O(N^2/2)$	CloudSim [17].

Table 1. Summary of scheduling algorithms in related work

3 EXPERIMENTAL SETUP

Evaluation of scheduling and resource allocation policies on real Cloud (with a varying load and system size) is a challenging problem. The use of real testbeds restricts the experiments to the scale of the test environment. Cloud computing model is based on a pay-per-use model, thus repeatable experiments on real Cloud may incur a high monetary cost. Therefore, an ideal alternative to evaluate resource management related Cloud policies is to use a simulation environment that enables Cloud developers to conduct experiments by employing the desired and varying configurations related to computing infrastructure and dataset (i.e., Cloud jobs). In this work, we use a renowned Cloud simulator called CloudSim [31] (version 3.0.2). A user job is represented as cloudlet in CloudSim and the job's size (computational requirement) is measured in terms of Million Instructions (MI). We perform the simulation-based experiments on a machine equipped with Intel Core i3-4030U Quad-core processor (having 1.9 GHz clock speed) and 4 GB of main memory. Liu and Cho [32] characterize the computing machines and workloads on a Google cluster and found that 93 % of the machines are fairly homogeneous on Google cluster with approximately 6 % of the machines with a greater computing capability [32]. Using the characteristics of the real computing machines (found in Liu and Cho's study [32]) we build an experimental setup for empirical evaluation. Table 2 illustrates the configuration details of the employed simulation environment. Figure 2 presents the overall statistics of the employed VMs with computing powers in terms of Million Instructions Per Seconds (MIPS).

Parameters	**Details**
Power of Cloud Host Machines	4 Dual core (4 000 MIPS), 26 Quad core (4 000 MIPS)
Total Host Machines	30 Host Machines
Total VMs	50 Virtual Machines
Total Cloudlets	100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, 700, 750, 800, 850, 900, 950, 1 000

Table 2. Configuration of simulation environment

3.1 Workload Generation

In this research work, one GoCJ benchmark dataset [37] and two synthetic datasets are used for the performance assessment of scheduling algorithms in simulation-based experimentation. The detail of these datasets are described as follows.

3.1.1 GoCJ Dataset [37]

The data confidentiality and other such policies maintained by the Cloud service providers [33] hinder to acquire real Cloud workload for the empirical investigations. The contemporary state-of-the-art has been scrutinized to explore a real workload

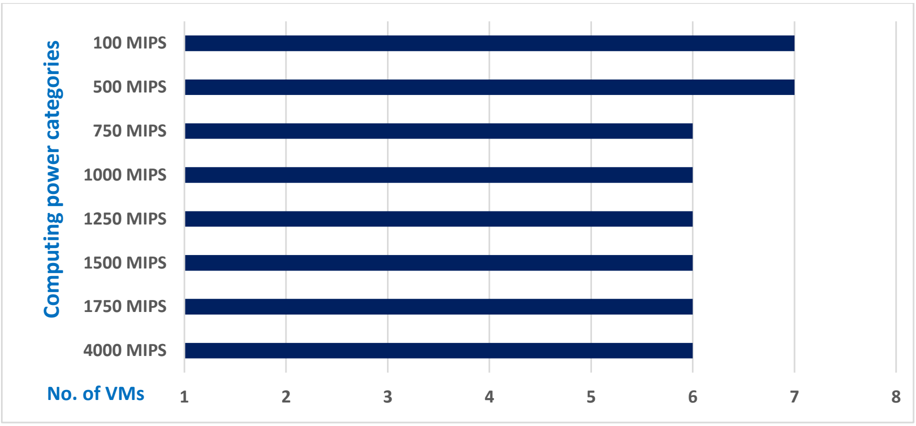


Figure 2. VMs in the cloud datacenter

behavior in Google cluster traces [32, 33, 34, 35, 36] and MapReduce logs from the M45 supercomputing cluster [36].

Liu and Cho studied a large-scale Google cluster usage traces of 29 days to examine the machine properties, workload behavior, and resource utilization [32]. The analysis of the Google cluster traces affirms that the majority of jobs execute for fairly a short duration (i.e., less than 15 minutes), while the low number of jobs execute over 300 minutes [32]. Further, the study [32] establishes the fact that approximately two thirds of the jobs in the Google cluster traces execute for less than five minutes and approximately 20 % of the jobs execute for less than one minute. The median length of a job in the Google cluster traces is approximately 3 minutes. Another similar study of Google cluster is presented by Chen et al. [35] and Reiss et al. [34].

In addition, Kavulya et al. [36] have scrutinized MapReduce logs of the M45 supercomputing cluster (logs of 10 months released by Yahoo). The study of MapReduce logs affirms that 95 % of the jobs complete the execution within 20 minutes and approximately 4 % of the jobs exceed execution up to 30 minutes [36]. Literature review [32, 33, 34, 35, 36] reveals that most of the Cloud jobs are of a short size and execute for less than 5 minutes.

Based on the analysis, we have generated a benchmark workload entitled Google Cloud Jobs (GoCJ) [37]. Considering the computing power of VMs in a Cloud datacenter (see Figure 1), the cloudlet completion time follows a long-tailed distribution (with 90 % of cloudlets in GoCJ workload completing their execution within 1.6 minutes). The longest executing cloudlet observed in the GoCJ workload lasts up to 15 minutes (6 % cloudlets execute for less than 5 minutes and 4 % execute for 15 minutes). The average size of a job in GoCJ workload is 5 minutes. Figure 3 presents the ratios and sizes of cloudlets distribution in GoCJ workload in terms of percentage and MIs, respectively.

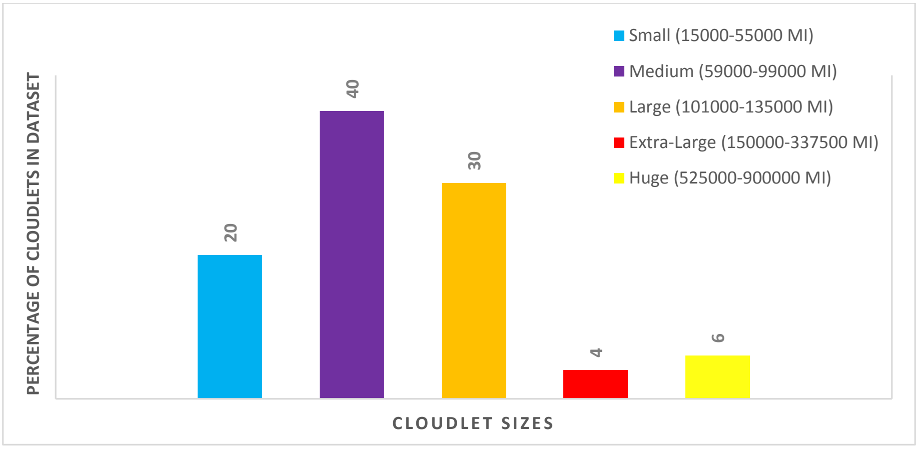


Figure 3. Composition of benchmark GoCJ dataset [37]

3.1.2 Synthetic Datasets

In addition to GoCJ dataset, we study the literature [28, 38, 39] to generate two synthetic datasets containing fix-sized jobs. Mehdi et al. [28] conducted the experimentation of a genetic scheduler with heterogeneous VMs (1.0 GHz to 4.0 GHz speed) to execute up to 100 cloudlets. In another work, Mehdi et al. [38] have examined the Cloud scheduling using 100 VMs (computing power of 1.0 GHz, 2.0 GHz, 2.5 GHz, and 3.0 GHz) to execute up to 500 cloudlets. Behzad et al. [39] presented a comparative analysis of different scheduling algorithms by using 7 000 and 15 000 jobs with a varying number of CPUs (i.e., 4 to 64 processors).

The synthetic-I workload is created with five fixed-size cloudlets (see Figure 4). The majority of cloudlets (i.e., 75 % of the cloudlets in synthetic-I workload) complete execution within 2 seconds and a small tail of the cloudlet distribution executes up to 45 seconds (i.e., 15 % of the cloudlets run for 15 seconds and 5 % of the cloudlets run for 45 seconds). The fixed sizes of tiny, small, medium, large and extra-large cloudlets in synthetic-I dataset are 200, 1 000, 5 000, 15 000, and 45 000 MIs, respectively.

The synthetic-II workload is generated using a random number generation mechanism by employing five cloudlet-size ranges (see Figure 4). The majority of cloudlets (i.e., 85 % in synthetic-II workload) are of a short size and a small tail of the cloudlet distribution completes execution within 45 seconds (i.e., 10 % of the cloudlets run for 10 seconds and 5 % of the cloudlets run for 45 seconds). The cloudlet-size ranges of tiny, small, medium, large, and extra-large cloudlets are 1–200, 800–1 200, 1 800–2 500, 7 000–10 000, and 30 000–45 000 MIs, respectively.

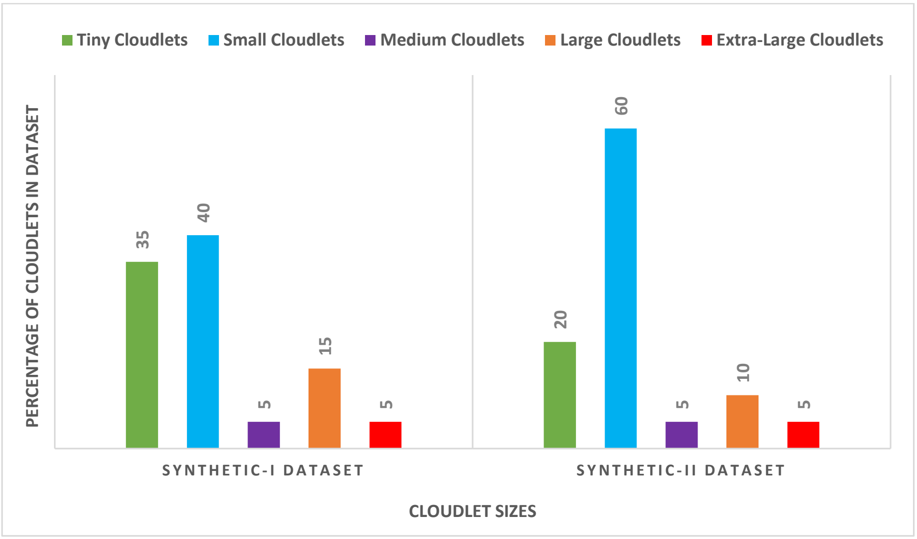


Figure 4. Composition of synthetic workloads

4 EXPERIMENTAL RESULTS

Three performance metrics are measured and presented for experimental evaluation, i.e., makespan, *Average Resource Utilization Ratio* (ARUR), and throughput.

4.1 Makespan-Based Results

We use term makespan to represent the completion of all the cloudlets execution in a workload. The smaller value of makespan represents a better execution performance. The makespan is mathematically expressed as follows:

$$Makespan = \max_{\forall j=1,2,3,\dots,m} (VM\_CT_j) \tag{1}$$

where m represents the total number of VMs (which is 50 in our experiments) and VM_CT_j is the completion time of VM_j by executing its assigned cloudlets. VM_CT_j is computed as:

$$VM\_CT_j = \sum_{i=1}^{n_j} \frac{Cloudlet_i.MI}{VM_j.MIPS} \tag{2}$$

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 represents the total number of cloudlets assigned to VM_j.

Figure 5 shows the makespan results of 10 scheduling algorithms for Synthetic-I, Synthetic-II, and GoCJ benchmark workloads. For more clarity, the average makespan (i.e., separately for synthetic-I, synthetic-II, and GoCJ workloads) of all the experiments using different number of cloudlets is calculated as follows:

$$Avg\_Makespan = \frac{\sum_{i=1}^{NE} Makespan_i}{NE} \tag{3}$$

where NE represents the number of experiments performed for each scheduling algorithm (i.e., using specified workload) and $Makespan_i$ represents the makespan of i^{th} experiment. Each experiment is repeated using a varying number of cloudlets (i.e., cloudlets 100–1 000, as presented in Table 2). The average makespan results for all scheduling algorithms are presented in Figure 6. The LBIMM, TASA, and Sufferage techniques produce the shortest makespan for Synthetic-I, Synthetic-II, and GoCJ workload, respectively. However, there is a minor difference with respect to makespan of LBIMM, TASA and sufferage algorithms for the three workloads. On the other hand, OLB achieves the largest makespan for Synthetic-I, Synthetic-II, and GoCJ benchmark workloads.

4.2 Throughput-Based Results

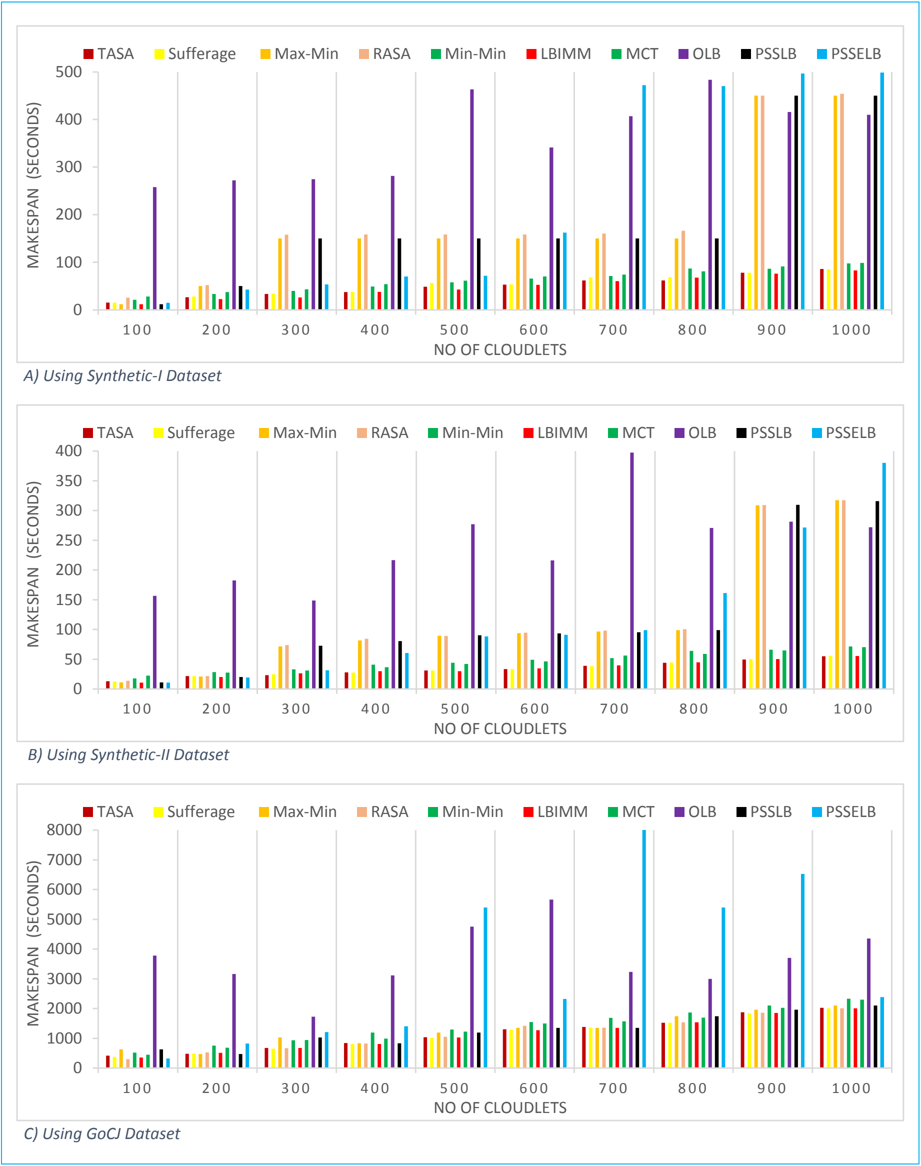
Throughput is the number of jobs executed during the span of per unit time. In our experiments, the throughput is referred as the number of cloudlets executed per second. Throughput can be calculated as follows:

$$Throughput = \frac{n}{Makespan} \tag{4}$$

where n is the number of employed cloudlets. A scheduling technique producing higher throughput value is assumed a better performing algorithm. For more clarity in results, the average throughput for synthetic-I, synthetic-II, and GoCJ workloads is calculted as:

$$Avg\_Throughput = \frac{\sum_{i=1}^{NE} Throughput_i}{NE}. \tag{5}$$

Figure 7 presents the throughput results for execution of Synthetic-I, Synthetic-II, and GoCJ benchmark workloads. While, for more clarity in the simulation results, Figure 8 represents the average throughput for all scheduling algorithms using the given workloads. Likewise average makespan results, the LBIMM, TASA, and Sufferage algorithms achieve the highest throughput for Synthetic-I, Synthetic-II, and GoCJ benchmark workloads, respectively. Similarly, OLB technique achieves the least throughput using given three workloads.



A) Using Synthetic-I Dataset

B) Using Synthetic-II Dataset

C) Using GoCJ Dataset

Figure 5. Makespan results

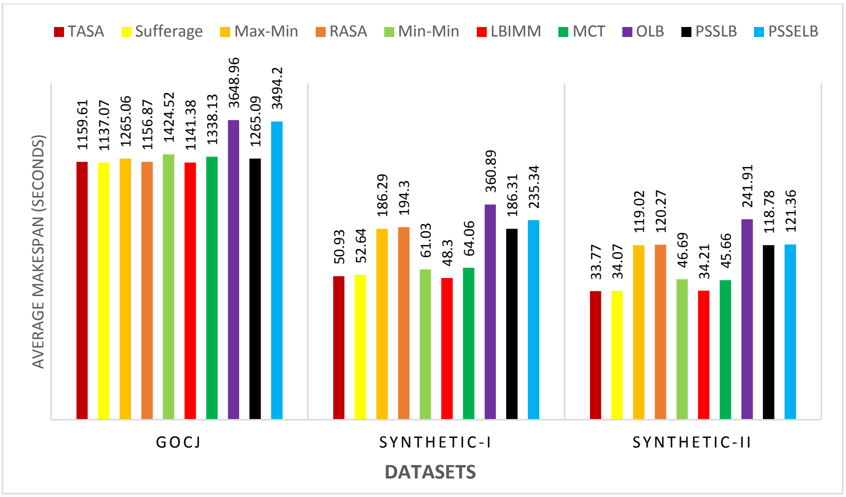


Figure 6. Average makespan results

4.3 ARUR-Based Results

ARUR shows the average resource utilization ratio for a compute Cloud. ARUR is the ratio of average makespan to the makespan of the Cloud system and is calculated as follows [10].

$$ARUR = \frac{\frac{\sum_{j=1}^{m} VM\_CT_j}{m}}{Makespan}. \tag{6}$$

ARUR value remains between 0 and 1, where value close to 1 shows exceptional resource utilization (i.e., nearest to 100 % resource utilization). Figure 9 shows the ARUR-based experimental results of ten scheduling algorithms for Synthetic-I, Synthetic-II, and GoCJ benchmark workloads. Mean ARUR value for each heuristic is reported based on the following equation:

$$Mean\_ARUR = \frac{\sum_{i=1}^{NE} ARUR_i}{NE}. \tag{7}$$

Figure 10 presents the Mean ARUR results for the execution of Synthetic-I, Synthetic-II, and GoCJ benchmark workloads. The LBIMM technique attains the highest ARUR (76.5 % resource utilization), as compared to other scheduling algorithms for Synthetic-I workload. However, in case of Synthetic-II and GoCJ benchmark workloads, Sufferage algorithm produces the highest resource utilization (75.7 % and 86.3 % resource utilization, respectively), as compared to other scheduling techniques. The OLB scheduling produces the least resource utilization among all scheduling techniques using given three workloads (i.e., 18.8 %, 20.7 %,

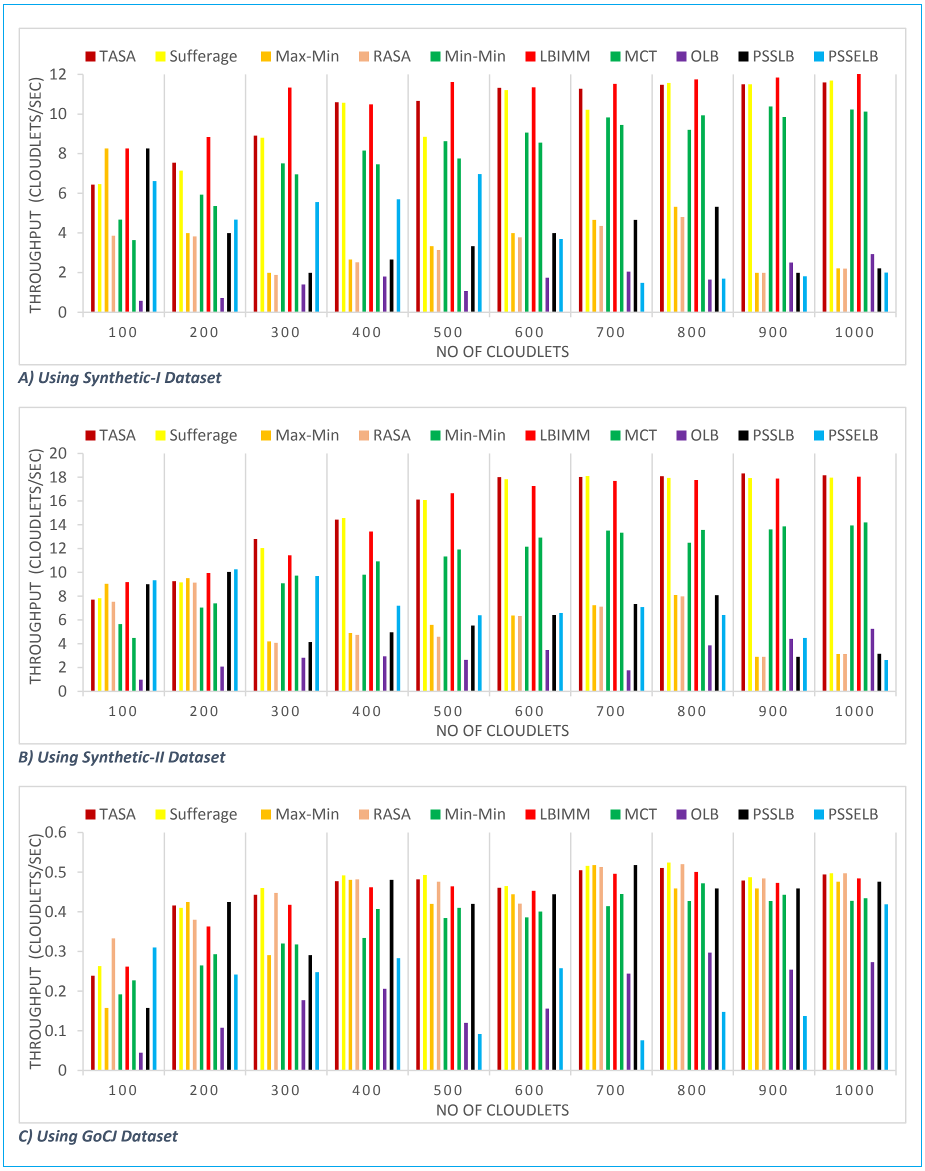


Figure 7. Throughput results

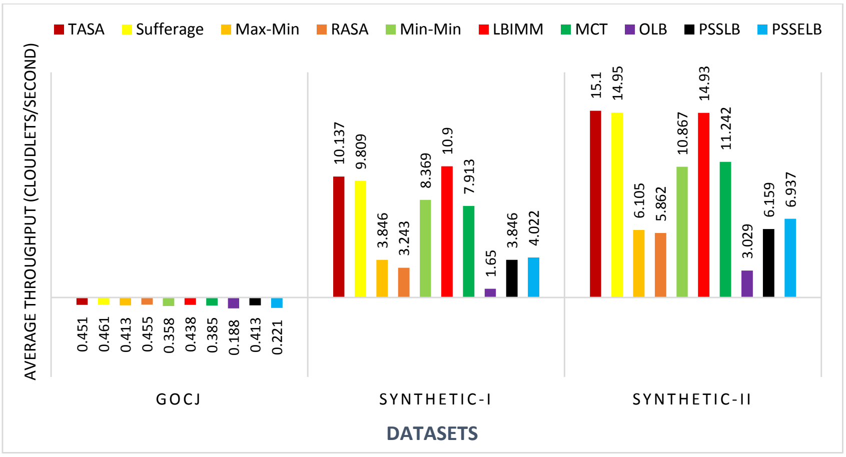


Figure 8. Average throughput results

and 41.2 % resource utilization for Synthetic-I, Sythetic-II, and GoCJ workloads, respectively).

4.4 Comparative Discussion

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

A workload is referred to as positively skewed if it contains a large number of shorter size jobs with a few very long jobs [25]. On the other hand, if the workload comprises a large number of longer jobs with a few shorter jobs then the workload is referred to as negatively skewed [25]. The skewness in synthetic and GoCJ benchmark workloads used in this study is shown in Figures 3 and 4.

Min-Min and Max-Min do not produce good results (in terms of execution time) for a skewed workload [25]. Min-Min scheduling attains improved execution time when the workload has shorter size jobs or cloudlets. On the other hand, Min-Min produces longer makespan for a positively skewed workload (due to the inherent penalty for larger jobs). In case of workload containing most of the shorter jobs with few longer jobs, Max-Min achieves improved makespan by executing longer jobs on faster machines; and concurrently executing the shorter jobs on comparatively slower machines. However, Max-Min and Min-Min perform worst for a skewed workload and provide improved results for a non-skewed workload [25].

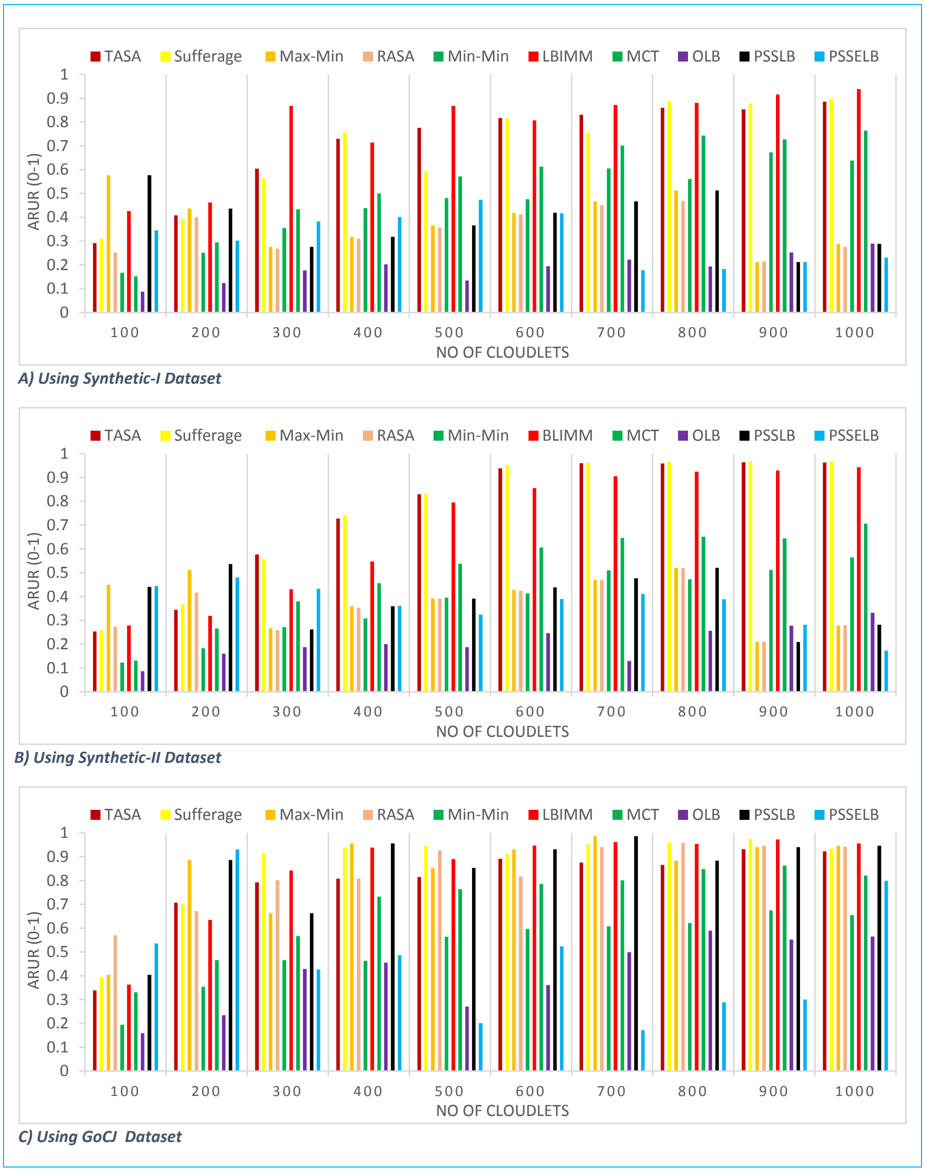


Figure 9. ARUR results

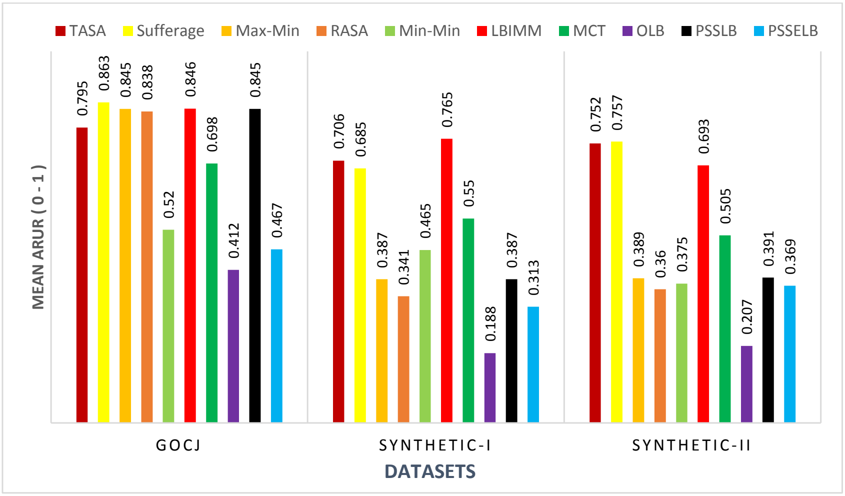


Figure 10. Mean ARUR results

RASA scheduling mechanism benefits from the merits of Min-Min and Max-Min producing a lower response time for both smaller and larger size jobs [12]. TASA, Sufferage, and LBIMM are modified versions of Min-Min [12, 16]. Therefore, these scheduling techniques provide better results (for a non-skewed workload) as compared to Min-Min. Additionally, TASA produces higher resource utilization (as evident by the results shown in Section 4.3). Similarly, PSSLB and PSSELB algorithms are the modified versions of Max-Min techniques (as shown in Figure 1). PSSELB provides better resource utilization as compared to Max-Min, while it introduces a slight degradation in makespan and throughput of PSSELB, as compared to Max-Min. On the other hand, PSSLB shows resemblance in results (i.e., makespan, resource utilization and throughput results), as compared to Max-Min technique.

5 RESOURCE UTILIZATION AND LOAD-IMBALANCE

We scrutinize the literature [1, 3, 4, 11, 12, 13, 16, 17] to examine the scheduling aspects related to load balancing and resource utilization for Cloud computing platform. The detailed analysis of the literature revealed that most of the existing scheduling algorithms [1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 21, 22, 23, 24, 25, 26, 28, 29] are more inclined towards the decrease in turnaround and response time of a Cloud workload. For example, MCT, Min-Min, Max-Min, RASA, TASA, and Sufferage algorithms are designed and proposed to minimize makespan of the Cloud workload. The OLB, LBIMM, PSSLB, and PSSELB techniques consider additional consideration of load balancing, too. However, most of these scheduling algorithms are still unable to fully utilize the computing resources and result in an imbalanced work distribution among the virtual machines.

Our empirical analysis reveals that LBIMM, TASA, and Sufferage techniques produce comparatively better utilization of computing resources (i.e., higher resource utilization), as compared to other scheduling algorithms presented in this work. In Table 3, the results show that the LBIMM heuristic attains higher resource utilization (i.e., 8.36 % and 11.52 % higher, respectively), as compared to the TASA and Sufferage for the execution of Synthetic-I workload. The Sufferage heuristic attained higher resource utilization (i.e., 0.66 % and 9.24 % higher, respectively), as compared to the TASA and LBIMM for the execution of Synthetic-II workload. Similarly, the Sufferage scheduling achieves higher resource utilization (i.e., 2.01 % and 8.55 % higher resource utilization, respectively), as compared to the LBIMM and TASA for the GoCJ workload. Moreover, the Sufferage, Max-Min, and PSSLB heuristics attain higher resource utilization too for the execution of a GoCJ workload. The Max-Min and RASA have achieved higher resource utilization of 5.40 % and 6.29 %, respectively, as compared to TASA for GoCJ workload. This minor improvement in resource utilization by RASA and Max-Min techniques over TASA is due to the lower resource utilization incurred by the Min-Min heuristic (as compared to Max-Min and RASA for the execution of GoCJ workload). Further, it is observed that the LBIMM, TASA and Sufferage schedulers achieve the minimal completion time and better resource utilization as compared to Max-Min, RASA, Min-Min, MCT, OLB, PSSLB, and PSSELB techniques. The experimental results reveal that the LBIMM, TASA and Sufferage mechanisms attain on average lower makespan compared to the other state-of-the-art. However, the most of these mechanisms still lack a higher resource utilization (see Table 3) that could be improved to further lessen the makespan for the execution of a Cloud workload.

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

Table 3. Percentage of resource utilization of scheduling algorithms

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

5.1 Discussion on VM Category and VM-wise Load Imbalance

For the empirical investigation, we employ the simulation environment based on 50 VMs (8 different sizes, as shown in Figure 2). For a balanced workload distribution, the cloudlets must be submitted to a compute Cloud for execution by considering the computing capabilities of the employed VMs. Moreover, it is critical to consider the current load of a VM, too. The computing load or share of each VM_j is presented as $Share_j$ and can be calculated as:

$$Share_j = \sum_{i=1}^{n} Cloudlet_i.MI \times \frac{VM_j.MIPS}{\sum_{k=1}^{m} VM_k.MIPS} \tag{8}$$

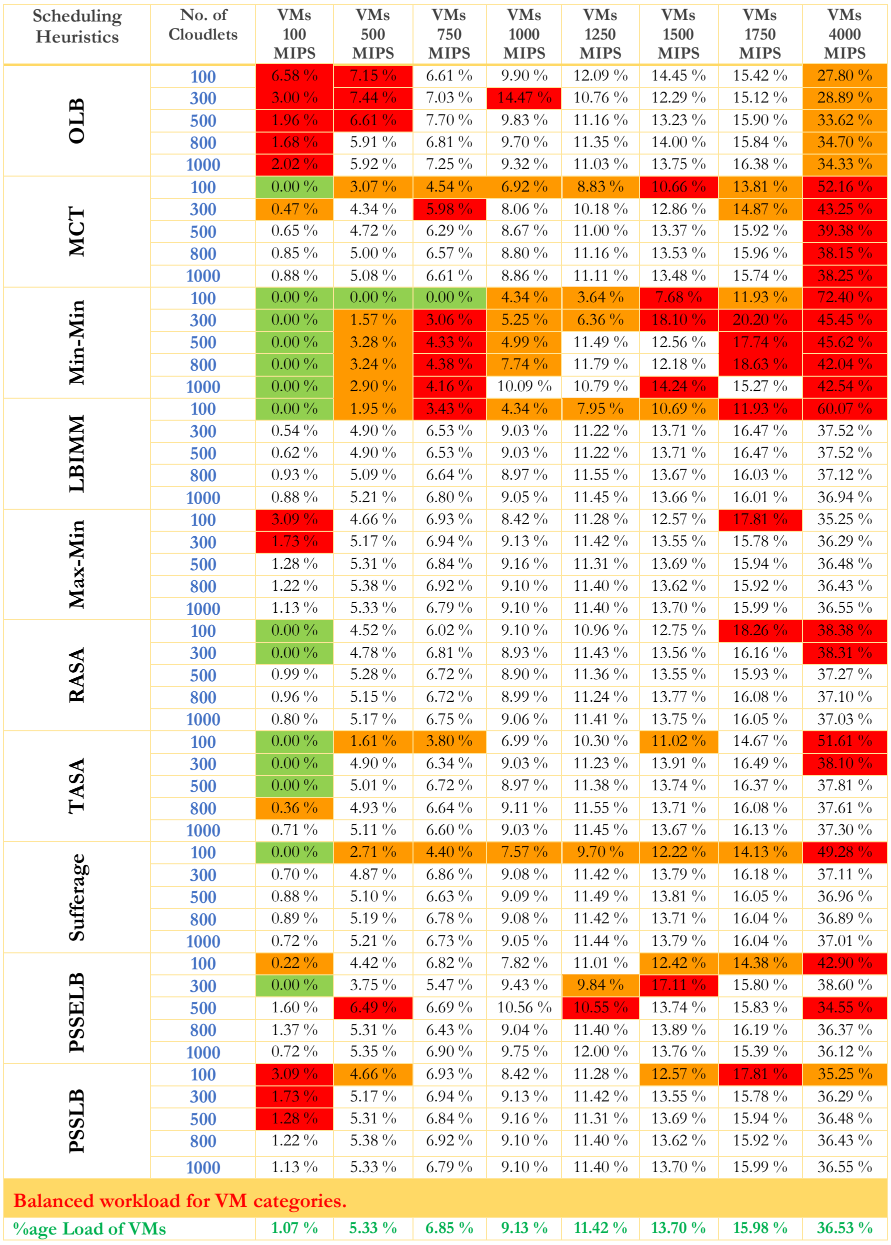
where $Share_j$ is the amount of workload in terms of MI that needs to be allocated to VM_j to attain a load-balanced scheduling. The balance share for each VM category in terms of percentage workload is represented as $VMCat\_Share_c$ and is calculated as follows:

$$VMCat\_Share_c = \frac{\sum_{a=1}^{cm} Share_a}{\sum_{i=1}^{n} Cloudlet_i.MI} \times 100 \tag{9}$$

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

Percentage workload distribution by the 10 scheduling algorithms is presented in a tabular form to highlight the load imbalance (see Figure 11). $VMCat\_Share_c$ of VM categories (see Figure 2) in a simulation environment is presented as a reference for load distribution attained by the employed scheduling algorithms (see Figure 11, presented in the last row). The imbalanced workload allocations are depicted such as the underutilized resources are filled with orange-color background, heavily loaded resources with red-background, and idle resources with a green background.

All VMs based on 100 MIPS (14 % of computing nodes in the experimental setup) and a few of VMs with 500 and 750 MIPS remain idle when the GoCJ workload is scheduled using Min-Min algorithm. Additionally, the Min-Min overloads the fastest VMs (based on 4 000 MIPS). On the other hand, the Max-Min scheduling produces better workload distribution as compared to the Min-Min; however, only a few VMs (both the slower and faster) are overloaded. This load mapping scenario is mainly contributed by the composition of GoCJ workload; where a small portion of large-sized cloudlets is present along with a majority of small-sized cloudlets. The Max-Min algorithm overcomes the imbalance produced by the Min-Min due to the presence of a few large-sized cloudlets which suits the Max-Min scheduling.



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

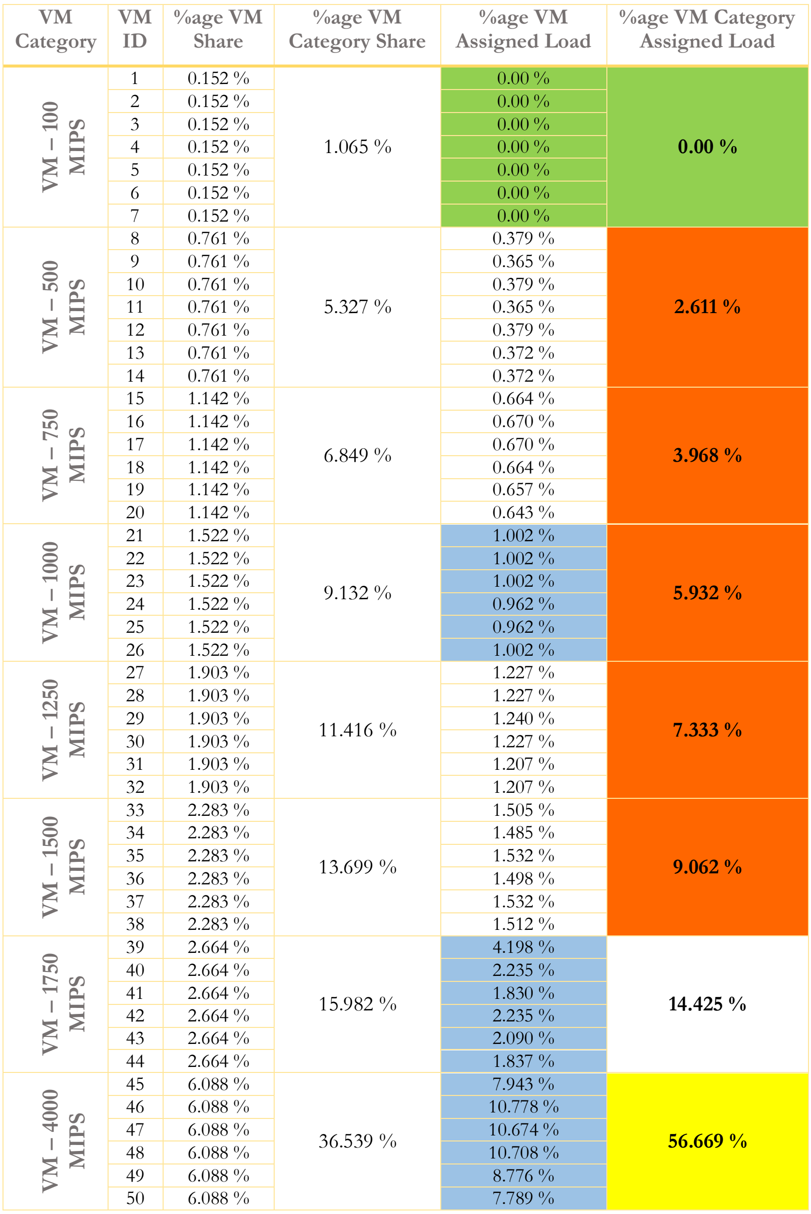
Figure 11. VM Category-wise percentage workload distribution for GoCJ workload

The PSSLB algorithm shows almost the same behavior for the workload distribution like Max-Min because both of these algorithms favor larger jobs. The LBIMM, Sufferage, RASA, Max-Min, and TASA produce comparatively a load-balanced schedule. Among these algorithms, Sufferage produces the highest resource utilization because of the resource-aware mechanism. However, an interesting observation is that the VMs based on 100 MIPS remain idle. Moreover, the recourse-aware mechanism employed by the Sufferage also produces a notable load imbalance, when the scheduling is performed using lesser number of cloudlets (i.e., 100 cloudlets), as shown in Figure 11. Similarly, LBIMM algorithm shows an improved load balancing in workload distribution. Also, RASA scheduling produces better load balancing due to the inherent usage of Min-Min and Max-Min (in alternate scheduling steps). However, the slowest machines remain idle (due to the inherent use of Min-Min algorithm) and the fastest VMs remain overloaded when a small number of cloudlets are scheduled by the RASA (see Figure 11). The employed alternate Min-Min and Max-Min mechanisms (by the RASA) produce a fair scheduling for both the large and small size cloudlets. Similarly, TASA technique produces the minimal makespan among the ten employed scheduling heuristics. However, most of the slower VMs (i.e., 100 and 500 MIPS based) become idle due to the use of Min-Min in alternate scheduling steps. On the other hand, TASA overcomes the load imbalance (caused by the Min-Min algorithm) to some extent with the help of inherent Sufferage based mechanism (in alternate scheduling steps). The Sufferage scheduling heuristic produces a better load-balanced schedule; however, very few slow VMs (with 100 MIPS) remain idle.

The results reveal that there is sufficient possibility of imbalance workload distribution (among VMs) even a scheduling technique attains an improved ARUR value; (as presented in Section 4.3). It is empirically evident that most of the existing scheduling mechanisms produce a reduced makespan with a higher throughput. However, often these algorithms result in a load imbalanced scheduling.

For example, Sufferage produces a higher ARUR value 0.863 (i.e., 86.3 % resource utilization) using the GoCJ workload (see Figure 22). However, the scheduling by Sufferage in this scenario does not utilize the VMs with computer power of 100 MIPS (i.e., those VMs remained idle). In addition, VMs with the computing capability of 500 and 750 MIPS are underutilized too and the VMs with 4 000 MIPS are heavily loaded (for the schedule of 100 cloudlets-based job pool (see Figure 11)). On the other hand, the VMs with the computing power of 100 MIPS were being utilized by the Sufferage scheduling when the number of cloudlets in the job pool increased.

Similarly, LBIMM algorithm attains 0.846 ARUR (i.e., 84.6 % resource utilization); however, VMs with 100 MIPS remain idle. Moreover, VMs with 4 000 MIPS are observed heavily overloaded and all the other VMs (in the employed experimental setup) are observed as underutilized (for 100 cloudlets-based scheduling). TASA technique produces 0.795 ARUR (i.e., 79.5 % resource utilization) for the GoCJ workload (see Figure 10); however, VMs with 100 MIPS remain idle (see Figure 11). TASA utilizes the VMs with 100 MIPS; however, most of these VMs (100 MIPS based) remained underutilized when the number of cloudlets to be scheduled are



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

Figure 12. VM-wise percentage workload distribution by Min-Min using 250 cloudlets of GoCJ workload

increased in the experiments. Min-Min scheduling technique produces 0.52 ARUR (i.e., 52 % resource utilization) for the GoCJ workload. Figure 11 depicts a load imbalance profile of the workload distribution by the Min-Min scheduling algorithm. The VMs with 100 MIPS remain idle due to the imbalanced scheduling by Min-Min algorithm. The VMs with 500 and 750 MIPS are assigned with cloudlets; however, these VMs are significantly underutilized (as shown in Figure 11), when the number of cloudlets increases. Similarly, VMs with 1 000, 1 250, 1 500, and 1 750 MIPS also remain underutilized for the Min-Min based scheduling. Contrarily, VMs with 4 000 MIPS are heavily overloaded by Min-Min technique (see Figure 11).

This empirical investigation reveals that the scheduling algorithms producing better ARUR value still result in a machine level load-imbalance for workload distribution. The imbalanced distribution of workload among the VMs within the same VM category is also observed. Figure 12 presents the workload allocation among all VMs by the Min-Min scheduling algorithm for the GoCJ workload (using 250 cloudlets). Figure 12 highlights the imbalanced distribution of workload. The underutilized VM categories are highlighted in orange color. The heavily overloaded or idle VM categories are highlighted with the yellow and green background, respectively. Similarly, the load imbalance of VMs within a specific VM category is shown with a light-blue color. Despite balancing the workload assigned to VM category with 1 750 MIPS, it can be seen that the VM with ID 39 is heavily overloaded (i.e., 4.198 % workload is assigned) and VMs with ID 41 and ID 45 are underutilized with only 1.830 and 1.837 % workload assignment, respectively (see Figure 12).

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

- a balanced distribution of workload among computing resources to be accomplished to achieve improved resource utilization with reduced makespan, and increased throughput in Cloud computing;
- designing and implementing a resource-aware holistic scheduling that not only considers application's computing requirements, but also contemplates virtual machine level attributes to provide a higher ARUR and near-optimal load balancing for the Cloud workload execution.

6 CONCLUSIONS

The inefficient utilization of resources by investigating the static heuristics for workload execution is empirically analyzed in this study. For this purpose, ten renowned Cloud scheduling heuristics are scrutinized and a comprehensive empirical study is conducted using the CloudSim simulation tool. The experiments are conducted using three workloads: two synthetics (i.e., Synthetic-I and Synthetic-II) and one benchmark GoCJ workloads. All the three workloads are based on static, non-pre-emptive, and compute-intensive Cloud jobs. Sufferage, LBIMM, Max-Min, and RASA produced the higher ARUR (i.e., 86.3 %, 84.6 %, 84.5 %, and 83.8 % resource utilization, respectively) using GoCJ workload. For LBIMM, Sufferage, and RASA based scheduling, most of the VMs remain idle or underutilized and the faster VMs (4 000 MIPS) are overloaded with the imbalanced workload. TASA scheduling mechanism has achieved a resource utilization of up to 79.5 % for the GoCJ workload while the majority of machines (based on 100 to 500 MIPS) mostly remained idle or underutilized and the faster machines (i.e., 4 000 MIPS) were overloaded. Similarly, the RASA scheduling mechanism produces a schedule that results in slower machines being idle (for the small-sized job pool); however, a gradual improvement in load-balanced was observed for the large size job pool. These results reveal that the out-performing heuristics are also unable to utilize the full computing capacity of Cloud resources. This empirical study carves out that the improper resource utilization and load imbalance is a crucial research issue that needs to be addressed comprehensively. This study identifies that the workload should be mapped in a balanced manner among the virtual machines considering both the computing capabilities of the Cloud resources and the applications computing requirements. In a consequent to this work, the authors are designing and implementing a resource-aware scheduler that considers the machines computing capabilities, the applications' computing requirements, and a balanced workload distribution constraint.

REFERENCES

[1] Aditya, A.—Chatterjee, U.—Gupta, S.: A Comparative Study of Different Static and Dynamic Load Balancing Algorithm in Cloud Computing with Special Emphasis on Time Factor. International Journal of Current Engineering and Technology, Vol. 5, 2015, No. 3, pp. 1898–1907.

[2] Jennings, B.—Stadler, R.: Resource Management in Clouds: Survey and Research Challenges. Journal of Network and Systems Management, Vol. 23, 2015, No. 3, pp. 567–619, doi: 10.1007/s10922-014-9307-7.

[3] Madni, S. H. H.—Abd Latiff, M. S.—Abdullahi, M.—Abdulhamid, S. M.—Usman, M. J.: Performance Comparison of Heuristic Algorithms for Task Scheduling in IaaS Cloud Computing Environment. PLoS One, Vol. 12, 2017, No. 5, pp. 1–26, doi: 10.1371/journal.pone.0176321.

[4] Mohialdeen, I. A.: Comparative Study of Scheduling Algorithms in Cloud Computing Environment. Journal of Computer Science, Vol. 9, 2013, No. 2, pp. 252–263, doi: 10.3844/jcssp.2013.252.263.

[5] Elzeki, O. M.—Rashad, M. Z.—Elsoud, M. A.: Overview of Scheduling Tasks in Distributed Computing Systems. International Journal of Soft Computing and Engineering, Vol. 2, 2012, No. 3, pp. 470–475.

[6] Maheswaran, M.—Ali, S.—Siegel, H. J.—Hensgen, D.—Freund, R. F.: Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing, Vol. 59, 1999, No. 2, pp. 107–131, doi: 10.1006/jpdc.1999.1581.

[7] Lenzini, L.—Mingozzi, E.—Stea, G.: Tradeoffs Between Low Complexity, Low Latency, and Fairness with Deficit Round-Robin Schedulers. IEEE/ACM Transactions on Networking, Vol. 12, 2004, No. 4, pp. 681–693, doi: 10.1109/tnet.2004.833131.

[8] Maipan-uku, J. Y.—Muhammed, A.—Abdullah, A.—Hussin, M.: Max-Average: An Extended Max-Min Scheduling Algorithm for Grid Computing Environment. Journal of Telecommunication, Electronic and Computer Engineering, Vol. 8, 2016, No. 6, pp. 43–47.

[9] Biradar, S.—Pawar, D.: A Review Paper of Improving Task Division Assignment Using Heuristics. International Journal of Science and Research, Vol. 4, 2015, No. 1, pp. 609–613.

[10] Mathew, T.—Sekaran, K. C.—Jose, J.: Study and Analysis of Various Task Scheduling Algorithms in the Cloud Computing Environment. 2014 IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2014, pp. 658–664, doi: 10.1109/icacci.2014.6968517.

[11] Chen, H.—Wang, F.—Helian, N.—Akanmu, G.: User-Priority Guided Min-Min Scheduling Algorithm for Load Balancing in Cloud Computing. 2013 National Conference on Parallel Computing Technologies (PARCOMPTECH 2013), 2013, pp. 1–8, doi: 10.1109/parcomptech.2013.6621389.

[12] Parsa, S.—Entezari-Maleki, R.: RASA: A New Grid Task Scheduling Algorithm. International Journal of Digital Content Technology and Its Applications, Vol. 3, 2009, No. 4, pp. 152–160, doi: 10.4156/jdcta.vol3.issue4.10.

[13] Braun, T. D.—Siegel, H. J.—Beck, N.—Bölöni, L. L.—Maheswaran, M.—Reuther, A. I.—Robertson, J. P.—Theys, M. D.—Yao, B.—Hensgen, D.—Freund, R. F.: 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, 2001, No. 6, pp. 810–837, doi: 10.1006/jpdc.2000.1714.

[14] Tabak, E. K.—Cambazoglu, B. B.—Aykanat, C.: Improving the Performance of Independent Task Assignment Heuristics MinMin, MaxMin and Sufferage. IEEE Transactions on Parallel and Distributed Systems, Vol. 25, 2014, No. 5, pp. 1244–1256, doi: 10.1109/tpds.2013.107.

[15] Li, B.—Pei, Y.—Wu, H.—Shen, B.: Heuristics to Allocate High-Performance Cloudlets for Computation Offloading in Mobile Ad Hoc Clouds. Journal of Supercomputing, Vol. 71, 2015, No. 8, pp. 3009–3036, doi: 10.1007/s11227-015-1425-9.

[16] Dehkordi, S. T.—Bardsiri, V. K.: TASA: A New Task Scheduling Algorithm in Cloud Computing. Journal of Advances in Computer Engineering and Technology, Vol. 1, 2015, No. 4, pp. 25–32.

[17] Alaei, N.—Safi-Esfahani, F.: RePro-Active: A Reactive – Proactive Scheduling Method Based on Simulation in Cloud Computing. Journal of Supercomputing, Vol. 74, 2018, No. 2, pp. 801–829, doi: 10.1007/s11227-017-2161-0.

[18] Tchernykh, A.—Lozano, L.—Schwiegelshohn, U.—Bouvry, P.—Pecero, J. E.—Nesmachnow, S.—Drozdov, A. Y.: Online Bi-Objective Scheduling for IaaS Clouds Ensuring Quality of Service. Journal of Grid Computing, Vol. 14, 2016, No. 1, pp. 5–22, doi: 10.1007/s10723-015-9340-0.

[19] Elzeki, O. M.—Reshad, M. Z.—Elsoud, M. A.: Improved Max-Min Algorithm in Cloud Computing. International Journal of Computer Applications, Vol. 50, 2012, No. 12, pp. 22–27, doi: 10.5120/7823-1009.

[20] Soltani, N.—Neysiani, B. S.—Barekatain, B.: Heuristic Algorithms for Task Scheduling in Cloud Computing: A Survey. International Journal of Computer Network and Information Security, Vol. 9, 2017, No. 8, pp. 16–22, doi: 10.5815/ijcnis.2017.08.03.

[21] Mao, Y.—Chen, X.—Li, X.: Max-Min Task Scheduling Algorithm for Load Balance in Cloud Computing. In: Patnaik, S., Li, X. (Eds.): Proceedings of International Conference on Computer Science and Information Technology. Springer, New Delhi, Advances in Intelligent Systems and Computing, Vol. 255, 2014, pp. 457–465, doi: 10.1007/978-81-322-1759-6_53.

[22] Sharma, G.—Banga, P.: Task Aware Switcher Scheduling for Batch Mode Mapping in Computational Grid Environment. International Journal of Advanced Research in Computer Science and Software Engineering, Vol. 3, 2013, No. 6, pp. 1292–1299.

[23] Li, H.—Wang, J.—Peng, J.—Wang, J.—Liu, T.: Energy-Aware Scheduling Scheme Using Workload-Aware Consolidation Technique in Cloud Data Centres. China Communications, Vol. 10, 2013, No. 12, pp. 114–124, doi: 10.1109/cc.2013.6723884.

[24] De Falco, I.—Scafuri, U.—Tarantino, E.: Two New Fast Heuristics for Mapping Parallel Applications on Cloud Computing. Future Generation Computer Systems, Vol. 37, 2014, pp. 1–13, doi: 10.1016/j.future.2014.02.019.

[25] Panda, S. K.—Agrawal, P.—Khilar, P. M.—Mohapatra, D. P.: Skewness-Based Min-Min Max-Min Heuristic for Grid Task Scheduling. Proceedings of the 2014 Fourth International Conference on Advanced Computing and Communication Technologies (ACCT '14), 2014, pp. 282–289.

[26] Yu, X.—Yu, X.: A New Grid Computation-Based Min-Min Algorithm. 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2009), 2009, pp. 43–45, doi: 10.1109/fskd.2009.81.

[27] Thaman, J.—Singh, M.: Green Cloud Environment by Using Robust Planning Algorithm. Egyptian Informatics Journal, Vol. 18, 2017, No. 3, pp. 205–214, doi: 10.1016/j.eij.2017.02.001.

[28] Mehdi, N. A.—Mamat, A.—Ibrahim, H.—Subramaniam, S. K.: Impatient Task Mapping in Elastic Cloud Using Genetic Algorithm. Journal of Computer Science, Vol. 7, 2011, No. 6, pp. 877–883, doi: 10.3844/jcssp.2011.877.883.

[29] Patel, R.—Chandel, M.: Analysis of Various Task Scheduling Algorithms in Cloud Computing. International Research Journal of Engineering and Technology, Vol. 3, 2016, No. 3, pp. 493–496.

[30] Gupta, K.—Katiyar, V.: Survey of Resource Provisioning Heuristics in Cloud and Their Parameters. International Journal of Computational Intelligence Research, Vol. 13, 2017, No. 5, pp. 1283–1300.

[31] Calheiros, R. N.—Ranjan, R.—Beloglazov, A.—De Rose, C. A. F.—Buyya, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software: Practice and Experience, Vol. 41, 2011, No. 1, pp. 23–50, doi: 10.1002/spe.995.

[32] Liu, Z.—Cho, S.: Characterizing Machines and Workloads on a Google Cluster. 2012 41st International Conference on Parallel Processing Workshops, 2012, pp. 397–403, doi: 10.1109/icppw.2012.57.

[33] Moreno, I. S.—Garraghan, P.—Townend, P.—Xu, J.: An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models. Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, 2013, pp. 49–60, doi: 10.1109/sose.2013.24.

[34] Reiss, C.—Tumanov, A.—Ganger, G. R.—Katz, R. H.—Kozuch, M. A.: Towards Understanding Heterogeneous Clouds at Scale: Google Trace Analysis. Intel Science and Technology Center for Cloud Computing, Technical Report ISTC–CC–TR–12–101, 2012.

[35] Chen, Y.—Ganapathi, A. S.—Griffith, R.—Katz, R. H.: Analysis and Lessons from a Publicly Available Google Cluster Trace. EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS–2010–95, 2010.

[36] Kavulya, S.—Tan, J.—Gandhi, R.—Narasimhan, P.: An Analysis of Traces from a Production MapReduce Cluster. 2010 11th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010, pp. 94–103, doi: 10.1109/ccgrid.2010.112.

[37] Hussain, A.—Aleem, M.: GoCJ: Google Cloud Jobs Dataset for Distributed and Cloud Computing Infrastructures. MDPI Data, Vol. 3, 2018, No. 4, pp. 1–12, doi: 10.3390/data3040038.

[38] Mehdi, N. A.—Mamat, A.—Ibrahim, H.—Syrmabn, S. K.: Virtual Machines Cooperation for Impatient Jobs under Cloud Paradigm. International Journal of Computer and Information Engineering, Vol. 5, 2011, No. 3, pp. 300–306.

[39] Behzad, S.—Fotohi, R.—Effatparvar, M.: Queue Based Job Scheduling Algorithm for Cloud Computing. International Research Journal of Applied and Basic Sciences, Vol. 4, 2013, No. 11, pp. 3785–3790.

[40] Liu, L.—Mei, H.—Xie, B.: Towards a Multi-QoS Human-Centric Cloud Computing Load Balance Resource Allocation Method. The Journal of Supercomputing, Vol. 72, 2016, No. 7, pp. 2488–2501, doi: 10.1007/s11227-015-1472-2.

Altaf Hussain received his M.S. degree in computer software engineering from National University of Science and Technology (NUST), Islamabad, Pakistan in 2010. He received his B.S. in computer science with distinction from NWFP AUP, Pakistan. His research interests include software testing, data mining, sentiment analysis and distributing computing comprising scheduling, performance analysis and Cloud computing. He is currently a Ph.D. scholar and the member of Parallel Computing Network (PCN) Research Group at Capital University of Science and Technology, Islamabad, Pakistan.

Muhammad Aleem received his Ph.D. degree in computer science from the Leopold-Franzens Universität Innsbruck, Austria in 2012. His research interests include parallel and distributed computing comprising programming environments, multi-/many-core computing, performance analysis, cloud computing, and big-data processing. He is currently working as Assistant Professor at Capital University of Science and Technology, Islamabad, Pakistan.

Muhammad Azhar Iqbal is Assistant Professor at the Capital University of Science and Technology, Islamabad, Pakistan. He received his Ph.D. degree in communication and information systems from the Huazhong University of Science and Technology, Wuhan, P.R. China in 2012. His research interests include coding-aware routing in vehicular ad hoc networks, energy-efficient MAC for wireless body area networks, largescale simulation modeling and analysis of computer networks in Cloud.

Muhammad Arshad Islam completed his doctorate from University of Konstanz, Germany in 2011. His dissertation is related to routing issues in opportunistic network. His current research interests are related to MANETs, DTNs, social-aware routing and graph algorithms. He is currently working as Assistant Professor at Capital University of Science and Technology, Islamabad, Pakistan.