Asian Journal of Research in Computer Science

12(4): 52-66, 2021; Article no.AJRCOS.77722
ISSN: 2581-8260

An Improved Rasa for Load Balancing in Cloud Computing

Oyekanmi Ezekiel Olufunminiyi [a*], Oladoja Ilobekemen Perpetual [b] and Omotehinwa Temidayo Oluwatosin [a]

[a] *Department of Mathematical Sciences, Achievers University, Owo, Nigeria.*
[b] *Department of Computer Science, Federal University of Technology, Akure, Nigeria.*

Authors' contributions

This work was carried out in collaboration among all authors. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/AJRCOS/2021/v12i430294
Editor(s):
(1) Dr. G. Sudheer, GVP College of Engineering for Women, India.
Reviewers:
(1) Norizan Mat Diah, Malaysia.
(2) Syed Shameem, Kl University, India.
Complete Peer review History, details of the editor(s), Reviewers and additional Reviewers are available here:
https://www.sdiarticle5.com/review-history/77722

Original Research Article

Received 06 October 2021
Accepted 13 December 2021
Published 14 December 2021

ABSTRACT

Cloud is specifically known to have difficulty in managing resource usage during task scheduling, this is an innate from distributed computing and virtualization. The common issue in cloud is load balancing management. This issue is more prominent in virtualization technology and it affects cloud providers in term of resource utilization and cost and to the users in term of Quality of Service (QoS). Efficient procedures are therefore necessary to achieve maximum resource utilization at a minimized cost. This study implemented a load balancing scheme called Improved Resource Aware Scheduling Algorithm (I-RASA) for resource provisioning to cloud users on a pay-as-you-go basis using CloudSim 3.0.3 package tool. I-RASA was compared with recent load balancing algorithms and the result shown in performance evaluation section of this paper is better than Max-min and RASA load balancing techniques. However, it sometimes outperforms or on equal balance with Improved Max-Min load balancing technique when using makespan, flow time, throughput, and resource utilization as the performance metrics.

Keywords: Improved RASA; load balancing; cloud computing; and resource utilization.

**Corresponding author: E-mail: ezekny@gmail.com, e.oyekanmi@achievers.edu.ng;*

1. INTRODUCTION

Cloud Computing consists of a network of infrastructure which allows enterprises to achieve more efficient use of their IT hardware, software, and other services investments. These infrastructures are available to users as a utility on an on-demand basis and are charged proportionally to the amount of resources consumed by the users. Cloud is a heterogeneous pool of resources, and to enable one to have access to these resources; there must be a Service Level Agreement (SLA) from the Cloud Provider [1]. Efficient resource utilization can be optimized by the Cloud Service Providers (CSP) via a better scheduling algorithm in cloud computing [2]. Better scheduling could be achieved by breaking down the physical barrier that is essential to isolated system and automate it as a single entity [3].

In Cloud Computing, effective task scheduling needs an infrastructure from the cloud provider. The scheduling of task, jobs are mapped on available resources and submitted to a cloud environment in such a way that the total response time and the makespan are minimized [4]. Makespan is the maximum time taken by any of the computing resources assigned a set of tasks to complete execution. Efficient resource utilization can only be guaranteed when the workload was efficiently shared among the resources [5]. As the cloud computing environment continues to enjoy massive migration by enterprises, it would be important to improve the distribution of tasks among resources. Cloud service providers are greatly concerned about addressing the challenge of ensuring that Virtual Machines (VMs) are not overloaded or underloaded. This optimization problem has been gaining attention in recent times.

This research therefore, focused on improving RASA algorithm to give a better performance on load-balancing in cloud computing, and compare its performance with other existing model for load balancing. The reason is towards improving the virtualization technology in cloud environment.

2. BACKGROUND OF THE STUDY

A lot of research had been done in this area, few studies with their contributions were discussed as follows:

Muthusamy and Chandran [6] proposed an artificial bee foraging optimization for load balancing in the cloud. The study adopted a preemptive task scheduling approach to minimize the response and execution time. This showed a significant improvement in QoS metric in comparison to the Honey Bee Based (HBB) load balancing.

In a bid to enhance machine performance for balanced sharing of load, maximize virtual machine throughput and optimize the waiting time of tasks, Jena, Das, and Kabat [7] hybridized Modified Particle Swarm Optimization (MPSO) and improved Q-learning. The experimental analysis results showed that the hybrid QMPSO outperformed MPSO and Q-learning.

In [8], an adaptive Starvation Threshold Load Balancing (STLB) algorithm was proposed for load balancing. The objectives of the study were to minimize response time and migration cost, and maximize server utilization rate. The key feature used in the algorithm was not invoked until at least one of the VM is close to starvation; this lowers the number of migration. The proposed algorithm was compared to the HBB load balancing algorithm in terms of makespan, average response time, average idle time, and number of tasks migrated, the STLB showed significant improvement.

In [9], a hybrid algorithm that integrated the Elephant Herding Optimization (EHO) into the Grey Wolf Optimizer (GWO) was proposed and its performance was compared to some existing load balancing algorithms such as Constraint Measure (CMBLB), Fractional Dragonfly, EHO, and GWO load balancing algorithms using makespan and minimum load as metrics. The makespan of EHGWO algorithm was better than EHO and GWO, and it has the lowest minimum load value compared with CMBLB and Fractional Dragonfly algorithms.

Quadri and Ravi [10] developed a new algorithm to reduce the overall time taken to complete a set of assigned tasks and ensure a balanced distribution of tasks to computing resources. The two objectives (minimize makespan and maximize utility) were achieved by applying the Weighted Sum Method (WSM), which was one of the multi-criterion decision-making methods. The percentage deviation (Coefficient of Variation) of the resources with maximum and minimum execution time of all tasks from the mean were measured. The result showed that the proposed algorithm has lower makespan than scheduling algorithms such as NHTBS, Opportunistic Load Balancing (OLB), MET and RASA. In terms of utilization only max-min scheduling technique could equal its performance.

According to [11], many algorithms have been used in the past; amongst them are Max-min, Min-min, and RASA, which are very popular during scheduling of tasks on resources. The study focused on max-min algorithm, based on completion time of processed tasks. This algorithm was improved upon using expected execution time [11]. The results showed that Improved Max–min (I-MM) performed better by 2% in completion time than RASA.

Our proposed improved RASA algorithm gives a better performance on load-balancing in cloud computing and especially in virtualization environment.

2.1 Overview of Past load-balancing Techniques in a Scheduling Environment

Different techniques have been used to improve the performance and resource usage on task scheduling, quality of service, load balancing, and resource utilization. In [12], it was suggested that load balancing in cloud avoids overloading/under-loading of virtual machines, which itself is an obstacle in cloud computing, thereby making it a requisite for researchers to develop a suitable load balancer for parallel and distributed cloud environments.

Load balancing is a method that enables frameworks on resources by giving a Maximum throughput with the least response time [7]. In cloud computing, the burden (overloading/ underloading) could occur in the resources used in the datacentres. The altering of this burden was a way to divide the action between all servers (in the case of more than one datacentre) or resources (say, virtual machines), so that job could be sent and response got quickly while the stack was on modification.

Various load balancing algorithms that give better throughput with quick response time in cloud condition do exist [13], but, each of them has favourable circumstance [14, 15, 16]. These include:

2.1.1 Static Algorithm (SA)

Static Algorithms, as shown in Fig. 1, are used where the load was of low variations. This algorithm needs a prior knowledge of server resources for the processors to perform better and this was determined at the beginning of the implementation [17]. The major limitation of Static Load Balancing Algorithm is that the load balancing tasks only work after being created.

2.1.2 Dynamic Algorithm (DA)

Dynamic Algorithm searches the lightest server resource, and gives it a preference for load balancing [17]. The current state of the machine was used to control the load. This was explained in Fig 2.

2.2.3 Round Robin Algorithm (RRA)

This algorithm assigns tasks to server resources. Its mode of operation uses the FCFS algorithm when the quantum time is high firstly, but once it reduces, tasks were assigned on a random basis in round- robin [18]. The round-robin method circularly assigned tasks without defining any priority. All the processes have different loading times. Some resources might be heavily loaded, while others remain under-utilized. The algorithm was shown in Fig 3.

```
Start
create different classes of job
assign next job to the belong class
Sort the class tasks based on weight of the execution time
for all classes
        for all task in selected class
                Assign task with min-weight to the next resource for processing
                 remove the task from the selected class list
                next task
        end
         next class
end
End
```

Fig. 1. Load balancing static algorithm

```
Start
create different classes of job , C
assign next job to the belong class
Sort the class tasks based on weight of the execution time
For all Resource, R
    for (i = 0; i<|C|; i++) // |C| is the total number of classes
        Assign task with min-weight in $C_i$ on R
        remove the task from the selected class list
        Next class
    end
    Next Resource
end
End
```

Fig. 2. Load balancing dynamic algorithm

```
Start
create different classes of job , C
assign next job to the belong class
Sort the class tasks based on weight of the execution time
For all Resource, R
    for (i = 0; i<|C|; i++) // |C| is the total number of classes
        if(task with min-weight in $C_i$ <quantum_time)
            Assign the task in $C_i$ on R
            remove the task from the selected class list
        else
            Process the task for based on quantum_time
            Update the task burst_time
            Relocate the task to the end of the class list
        Next class
    end
    Next Resource
end
End
```

Fig. 3. Round-robin algorithm

```
Start
create different classes of job
assign next job to the belong class
Sort the class tasks based on weight of the execution time

$N = \sum_{i=1}^{|C|} |T_{C_i}|$
Counter=0;
Do
    i = Random (1, |C|)
    for all tasks in selected class $C_i$
        Assign task with min-weight to the next resource for processing
        remove the task from the selected class list
        Counter++;
        next task
    end
While (Counter < N)
End
```

Fig. 4. Opportunistic load balancing algorithm

2.2.4 Opportunistic Load Balancing (OLB) algorithm

This algorithm keeps each server resource busy, as shown in Fig 4, without considering machines' current workload. Irrespective of the current workload on each of the resources, OLB distributes all the unfinished tasks to them randomly [19].

2.2.5 Minimum to Minimum (Min-Min) algorithm

The concept of Min-min algorithms in Load Balancing is to assign tasks with minimum completion time first for execution on resource with minimum execution time [20]. This procedure continues until all were mapped. This algorithm, as shown in Fig 5, seems to be the fastest in a situation where many smaller tasks are more than larger ones.

2.2.6 Maximum to Minimum (Max-Min) Algorithm

Maximum-Min Load Balancing Algorithm is similar to Min-min load balancing algorithm, still the difference is that the task with maximum completion time was selected after searching and assigned to the machine (resource) with minimum execution time [21]. The execution time of all tasks were updated, and the assigned task was removed from the list. Fig 6 shows the detailed code.

2.2.7 Resource Aware Scheduling Algorithm (RASA)

This is a combination of both Max-min and Min-min algorithms. In RASA, the appraisal of the completion time for each task on available resources was calculated, after which the Max-min and Min-min algorithms were applied alternatively, as shown in Fig 7, thereby making use of the advantage of both algorithms and avoiding their drawbacks [4]. RASA executes small tasks to avoid delays in large ones. It also supports simultaneous executions of large and small tasks.

2.2.8 Improved Max-Min

The basic operations of improved Max-min (I-Max-Min) was to assign task with maximum execution time to a resource that has minimum completion time as shown in Fig 8. The original Max-min assigned task with maximum completion time to resources with minimum execution time [22].

```
Start
Create different classes of job, C
Assign next job to the belong class
Sort the class tasks based on weight of the execution time
//R is the resources
for (i = 0; i<|C|; i++) // |C| is the total number of classes
   Assign task with min-weight in $C_i$ on $min\ (R_{1_{executiontime}}, \dots, R_{|R|_{executiontime}})$ resource
   Remove the task from the selected class list
   Next class
end
End
```

Fig. 5. Opportunistic load balancing algorithm

```
Start
Create different classes of job, C
Assign next job to the belong class
Sort the class tasks based on Max-weight of the execution time
//R is the resources
for (i = 0; i<|C|; i++) // |C| is the total number of classes
   Assign task with Max-weight in $C_i$ on $min\ (R_{1_{executiontime}}, \dots, R_{|R|_{executiontime}})$ resource
   Remove the task from the selected class list
   Next class
end
End
```

Fig. 6. Opportunistic load balancing algorithm

```
Start
Create different classes of job, C
Assign next job to the belong class
Sort the class tasks based on Max-weight, Min-Weight of the execution time interchangeably
//R is the resources
for (i = 0; i<|C|; i++) // |C| is the total number of classes
        Assign next task in $C_i$ on $min\ (R_{1_{completiontime}}, \dots, R_{|R|_{completiontime}})$ resource
        Remove the task from the selected class list
        Next class
end
End
```

Fig. 7. Resource aware scheduling algorithm

```
Start
Create different classes of job, C
Assign next job to the belong class
Sort the class tasks based on Max-weight of the execution time
//R is the resources
for (i = 0; i<|C|; i++) // |C| is the total number of classes
        Assign next task in $C_i$ on $min\ (R_{1_{completiontime}}, \dots, R_{|R|_{completiontime}})$ resource
        Remove the task from the selected class list
        Next class
end
End
```

Fig. 8. Improved max-min algorithm

3. META-TASK

The concept of Meta-task is simply to transform complex processes going through multiple resources into simple lists of tasks sorted out in batches for higher throughput during scheduling computation. This concept was implemented in Linux operating system by way of using Complete Fair Queuing (CFQ) scheduler, also known as modified anticipatory scheduler. CFQ are currently in use because aside that tasks were put in batches, it also overcomes deceptive idleness [23]. Report also shows that Apache web server achieved up to 71% more throughput from using this modified anticipatory scheduler [24].

Scheduling algorithms are with many policies but could be subdivided into immediate and batch scheduling, preemptive and non-preemptive scheduling, static and dynamic scheduling, and so on [25, 26,27]. In Immediate mode, tasks were scheduled using First Come First Serve (FCFS) in the computing environment, while in the batch mode, tasks were grouped into a batch; which means that, a set of meta-tasks would be assigned at a mapped out time depending on the scheduler's algorithm [26]. It is the algorithm that determines how the load balancing of the resource usage for this task. In a nutshell, meta-task is a way of assigning a mapped tasks to different entities in cloud for resource provisioning.

4. RESOURCE PROVISIONING

The task of mapping resources to different entities in cloud on-demand that is pay-as-you-go basis is known as resource provisioning. Resources were allocated in cloud so that the processing elements (resources) are not overloaded and that none is undergoing wastage either. Resources mapping in cloud entities were done in two levels:

4.1 Host

Host, in cloud computing, can contain more instances of VM. The VM were mapped to a single host subject for availability and capabilities. The Host then assigns processing cores to VM based on the provisioning policy, and this defines the basis of allocating processing cores to VM. The allocation policy ensures that the critical characteristics of the Host and VM do not mismatch.

4.2 Cloudlet or Task Mapping onto VM

Cloudlets were executed on VM, and each requires a certain amount of processing power for their completion. VM provides this processing power to the task(s) mapped on it. These tasks were mapped on VM based on their configuration and availability.

4.3 Task Scheduling Policy

Tasks were scheduled after the resources had been allocated to the cloud entities. These activities allow multiprogramming capabilities in a cloud environment and were enabled in two modes: Space shared and Time shared policies. In Space Shared policy at VM level, one task can be scheduled to a virtual machine at a time and when it was completed, another task is scheduled to the virtual machine. This policy behaves same as the First Come First Serve (FCFS) scheduling algorithm [28]. The algorithm of space shared policy is as follows:

Step 1: Tasks are arranged in a queue.

Step 2: First task is scheduled on the given virtual machine.

Step 3: When first task is completed it assigns the next task from the queue.

Step 4: If queue is empty it checks for new tasks.

Step 5: Then repeat Step 1.

Step 6: End.

Same algorithm is applicable for both Host level and VM level scheduling.

In Time Shared policy at Host level, virtual machines are scheduled on the CPU cores simultaneously amongst the VM, while at VM level, the scheduling policy schedules all the tasks on the VM at the same time, and this is done among all tasks. This algorithm is the same like the Round Robin (RR) scheduling algorithm [29]. The algorithm of time shared policy can be represented as follows:

Step 1: All the tasks are arranged in a queue.

Step 2: Then schedule the tasks simultaneously on the virtual machine.
Step 3: When queue is empty it checks for new tasks.

Step 4: If new task arrives it schedules similarly as in Step 2.
Step 5: End.

The algorithm can be applied for both Host level and VM level of scheduling.

This study implements resource mapping at both host and VM levels via load balancing using CloudSim 3.0.3 package. Load balancing in cloud provides an efficient solution to various issues applicable to cloud computing set-ups and usage. However, this does not necessarily result in shorting makespan [11]. Hence, we proposed a new load balancer called I-RASA.

5. PROPOSED METHOD FOR LOAD BALANCING

In this section, an improved Resource Aware Scheduling Algorithm (I-RASA) as a load balancer for both small and large distributed system has been developed. This method as shown in Fig 9, calculates the expected completion time of the submitted tasks on each resource. The max-min algorithm is applied on the tasks length for the resources used first while, the min-min algorithm is then applied for same length of resources used. The max-min algorithm is recalled again on the remaining tasks with the overall minimum expected execution time assigned to the resource that had the minimum overall completion time. After scheduling, the task is removed from meta-tasks and all calculated times are updated and the processing is repeated until all submitted tasks are executed. This method helps in minimizing the total makespan which is the total complete time in both small and large distributed system. The proposed method is compared with the last two discussed algorithms in section II of this paper.

The flowchart processing was shown in Fig. 10. A procedure is created which performs the work of sorting based on execution time of each cloudlets (tasks) and completion time of each virtual machine installed on the host. Once the procedure (function/module) is called, the highlighted steps were executed and a test is done to know if more task remains. If Yes, the procedure recalled. In other words, the procedural flowchart symbol used here is a recursive one which keeps recalling itself as long as the meta-task is not empty.

5.1 Simulation configuration and tools

The implementation was done using CloudSim 3.0.3 as a framework in the scalable simulator environment. The simulation was performed on core i3 processor with 6GB Ram and 500GB disk space, performed on Window 7 Ultimate Edition operating system. The experiment was written in java language on Jcreator IDE. The study considered two Datacenters, VM, host and cloudlet components from CloudSim for execution analysis on I-RASA, RASA and I-Max-Min algorithms. The simulation process comprises of different requests by the user to be processed. The total number of cloudlets or request is 50 and it ranges from 300 MB to 23000 MB. The range value was adopted from [30]. Table 1 shows the configuration of the datacenters which consist of four (4) virtual machines with the same configurations.

```
Start
Create different classes of Metatask, C
Assign next Metatask to its class //This is done as request (Metatask) comes in

//After all the Metatasks has been assigned to each class then
for (i = 0; i<|C|; i++) // |C| is the total number of classes
    First arrange Metatasks with Max-weight of execution time based on number of resources
    Arrange the next Metatasks with Min-weight of execution time still based on number of resources
    Arrange the remaining Metatasks based on Max-weight of execution time
   Next class
end
//R is the resources
for (i = 0; i<|C|; i++) // |C| is the total number of classes
      Assign next task in $C_i$ on $min\ (R_{1_{completiontime}}, \ldots, R_{|R|_{completiontime}})$ resource
      Remove the task from the selected class list
     Next class
end
End
```

Fig. 9. Pseudo-code for I-RASA

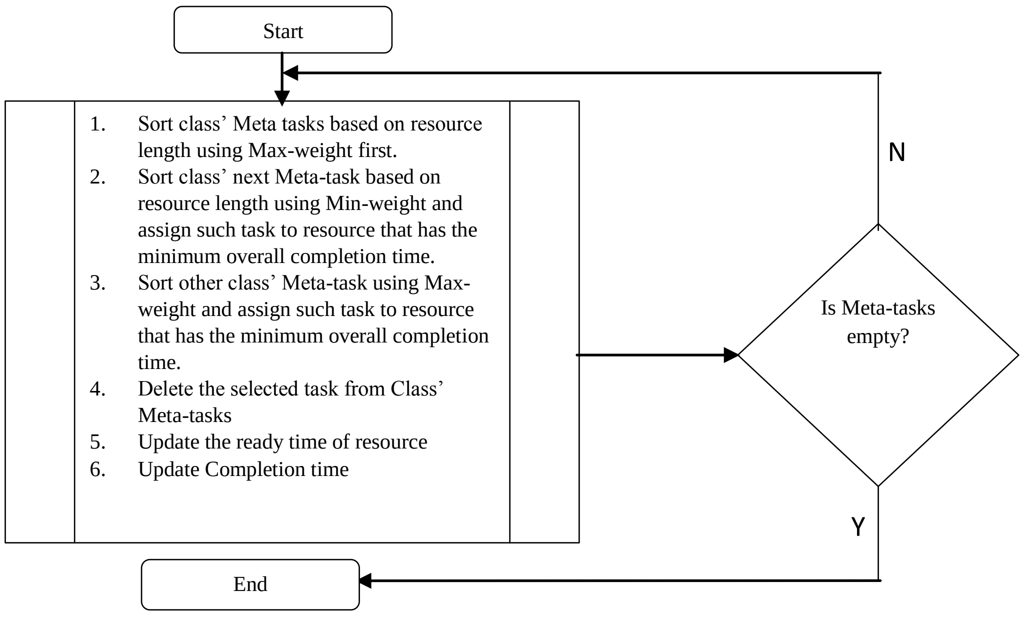
Start
1. Sort class' Meta tasks based on resource length using Max-weight first.
2. Sort class' next Meta-task based on resource length using Min-weight and assign such task to resource that has the minimum overall completion time.
3. Sort other class' Meta-task using Max-weight and assign such task to resource that has the minimum overall completion time.
4. Delete the selected task from Class' Meta-tasks
5. Update the ready time of resource
6. Update Completion time
Is Meta-tasks empty?
N
Y
End

Fig. 10. Flowchart for I-RASA

Table 1. VM Configuration

Parameter	Value
Size (MB)	10000
Ram (MB)	512
Processing Speed (MIPS)	1000
Bandwidth (MBBS)	1000
VM Name	“Xen”
PesNumber	1

6. PERFORMANCE METRIC USED FOR THE EXPERIMENT

The results of the performance of the algorithms I-RASA, I-Max-Min and RASA were evaluated using the following metrics tested in CloudSim toolkit.

6.1 Makespan

Makespan is the finishing time of the last task and in scheduling of task, one of the optimization criterions is minimization of makespan as most of the users desire fastest execution of their application. Equation 1 shows its mathematical representation.

$$makespan = max_{i \in tasks}\{F_i\} \qquad (1)$$

where F_i denotes the finishing time of the last task.

6.2 Economic Cost

It indicates the total amount the user needs to pay to service provider for resource utilization. The mathematical representation is shown in equation 2.

$$Economic\ Cost = \sum_{i \in resources}\{C_i * T_i\} \qquad (2)$$

where C_i denotes the cost of resources i per unit time and Ti denotes the time for which resource i utilized. From the CloudSim 3.0.3 version a pre-defined cost value for resources used is as follows:

the cost of using processing in VM resource is 3.0

the cost of using memory in VM resource is 0.05

the cost of using storage in VM resource is 0.001

The cost in total is 3.051

6.3 Flow Time

This indicates the total sum of finishing times of all the tasks. And to minimize this, tasks should be executed in ascending order of their processing time.

$$Flowtime = \sum_{i \in tasks} F_i$$

where F_i denotes the finishing time of task i.

6.4 Resource Utilization

This helps in understanding how busy the resources are. This is very important as service providers would like to earn maximum profit by renting limited number of resources. Equation 3 shows how it can be implemented mathematically.

$$Average\ resource\ utilization = \sum_{i=1}^{n} Time\ taken\ by\ resource\ i\ to\ finish\ all\ jobs \Big/ Makespan\ *\ n \qquad (3)$$

where n is the number of resources.

6.5 Throughput

This can be defined, as shown in equation 4, as the total number of jobs completing execution per unit time.

$$Average\ throughput = Number\ of\ Tasks/makespan \qquad (4)$$

7. PERFORMANCE EVALUATION

Table 2 shows the output of 20 processed cloudlets using I-Max-Min algorithm. The Gantt chart for the flow of process is represented in Fig. 10. From Fig. 10, the makespan of the last finished task is 59.

Table 3 shows the output of 20 processed cloudlets using Resource Aware Scheduling Algorithm. The Gantt chart for the flow of process in this algorithm is shown in Fig. 11. The makespan from Fig. 11 is 65. This is higher compare with the makespan in improved max-min.

Table 4 shows the output of 20 processed cloudlets using our proposed algorithm called Improved Resource Aware Scheduling Algorithm. The Gantt chart for the flow of process in this algorithm is shown in Fig. 12. The makespan is 53. This is low when compared with the makespan in the two previous algorithms.

Table 2. Simulation result of four virtual machines using improved max-min

Cloudlet ID	VM ID	Time	Cloudlet Length	Start Time	Finish Time
3	3	19	19259	0	19
2	2	20	20332	0	20
1	1	21	20581	0	21
0	0	23	22726	0	23
6	1	16	16377	21	37
4	3	19	19132	19	38
7	0	16	16010	23	39
5	2	19	18811	20	39
11	2	10	9895	39	49
10	0	11	10502	39	50
9	3	12	11666	38	50
8	1	14	13765	37	51
15	1	4	3960	51	55
13	0	7	7060	50	57
14	3	6	6455	50	56
12	2	8	7725	49	57
16	1	4	3843	55	59
19	2	2	2103	57	59
17	0	4	3535	57	61
18	3	3	3421	56	59

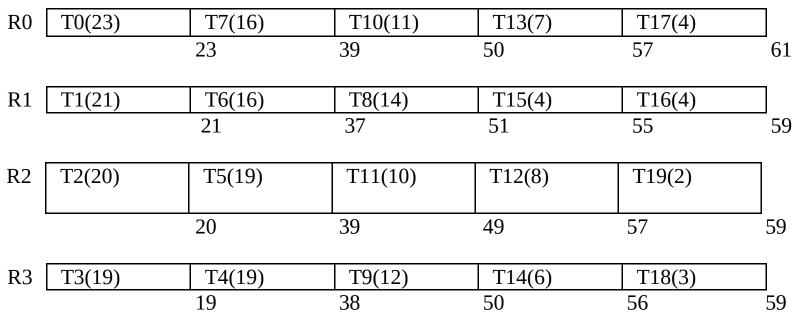

Fig. 11. Gant-chart of I-max-min

Table 3. Simulation result of four virtual machines using RASA

Cloudlet ID	VM ID	Time	Cloudlet Length	Start Time	Finish Time
1	1	2	2103	0	2
3	3	3	3421	0	3
5	3	4	3535	3	7
2	2	21	20581	0	21
4	1	20	20332	2	23
0	0	23	22726	0	23
7	2	4	3843	21	25
6	3	19	19259	7	26
9	0	4	3960	23	27
11	3	6	6455	26	32
13	3	7	7060	32	39
8	1	19	19132	23	42
12	0	16	16377	27	43
10	2	19	18811	25	44
15	1	8	7725	42	49
17	2	10	9895	44	54
14	3	16	16010	39	55
16	0	14	13765	43	57
18	1	12	11666	49	61
19	2	11	10502	54	65

Table 4. Simulation result of four virtual machines using I-RASA

Cloudlet ID	VM ID	Time	Cloudlet Length	Start Time	Finish Time
6	3	19	19259	0	19
4	2	20	20332	0	20
2	1	21	20581	0	21
1	3	2	2103	19	21
0	0	23	22726	0	23
3	2	3	3421	20	23
5	1	4	3535	21	25
7	3	4	3843	21	25
12	1	16	16377	25	41
14	3	16	16010	25	41
8	0	19	19132	23	42
10	2	19	18811	23	42
19	0	11	10502	42	53
17	2	10	9895	42	52
18	3	12	11666	41	53
16	1	14	13765	41	55
9	1	4	3960	55	59
11	3	6	6455	53	59
13	2	7	7060	52	59
15	0	8	7725	53	61

Table 5. Comparison table of the algorithms

Metrics/Algorithms	RASA	Improved Max-Min	Proposed (I-RASA)
Makespan	65	59	53
Economic Cost	726.14	726.14	726.14
Flowtime	696	899	794
Throughput (%)	31	34	38
Resource Utilization (%)	92	100	100

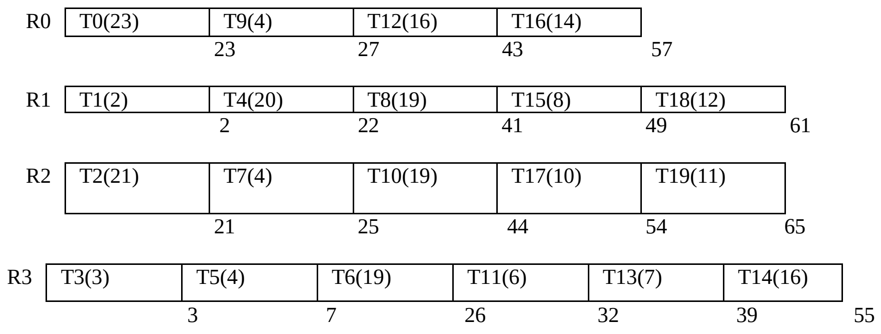

Fig. 12. Gant-Chart of RASA

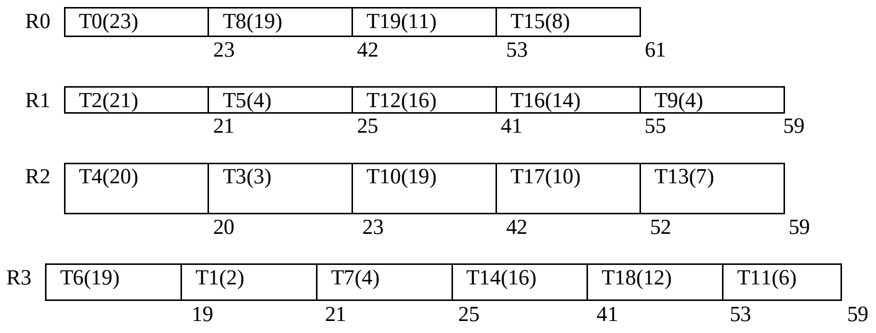

Fig. 13. Gant-Chart of I-RASA

Following the report from Table 5 and the graph shown in Fig. 14, our proposed algorithm, I-RASA out-perform improved max-min and resource aware scheduling algorithms in terms of low makespan, high throughput and on the same merge with the improved max-min in the resource utilization. Although the flow time of RASA is lower compared with our proposed algorithm and that of improved max-min, however, the flow time of I-RASA still outperform improved max-min algorithm. Fig. 15 shows the load balancing on the four (4) virtual machines used with respect to the completion time of each.

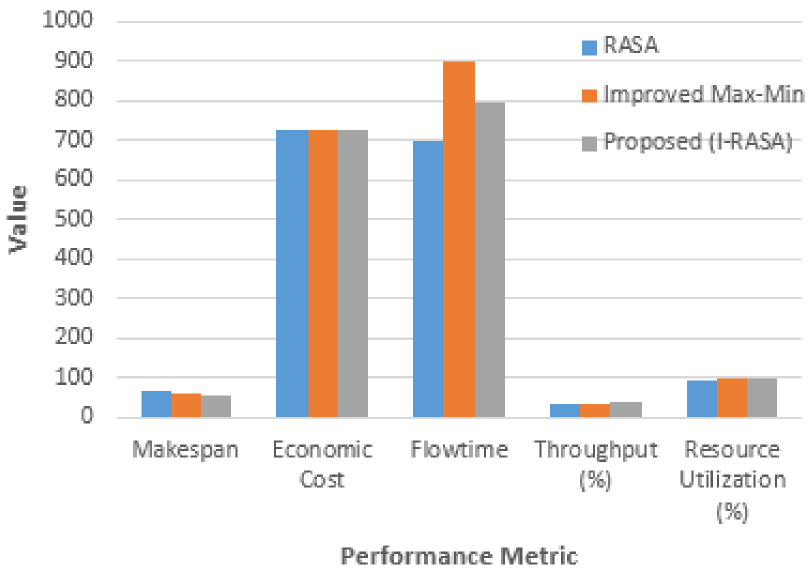

Fig. 14. Performance metric chart for the simulation

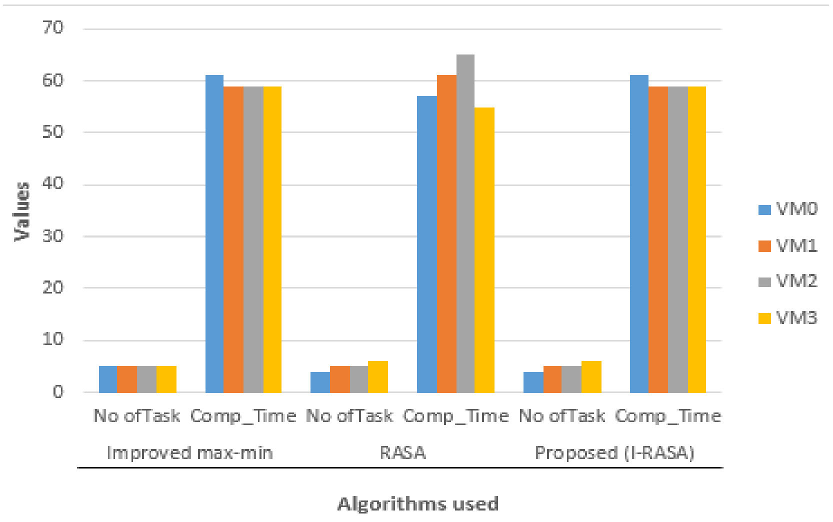

Fig. 15. Load-balancing Chart for the simulation

Table 6. Makespan of 20 processed cloudlets on 10 different iterations

	Makespan		
Iteration	**Improved Max-Min**	**RASA**	**Proposed(I-RASA)**
1	57	63	51
2	53	59	48
3	50	54	44
4	45	50	41
5	55	59	47
6	58	65	52
7	72	78	61
8	55	62	50
9	59	65	53
10	61	66	54

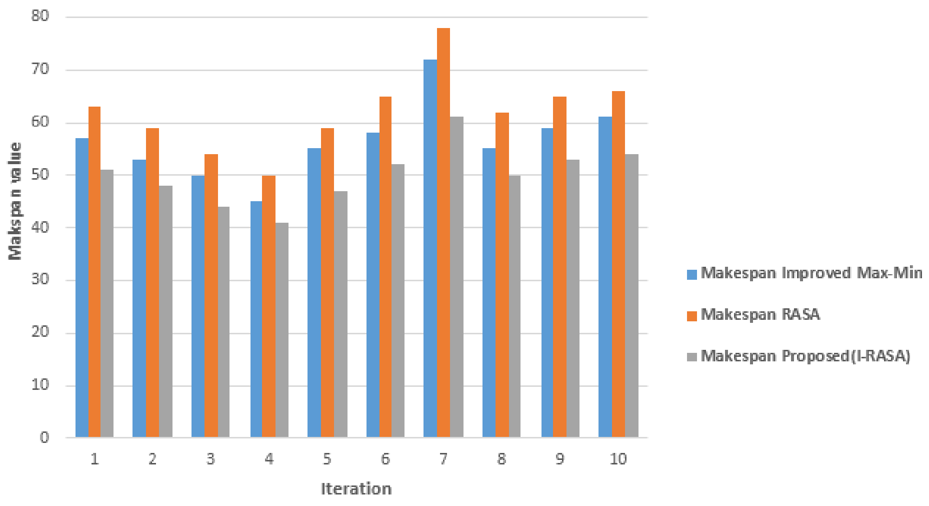

Fig. 16. Makespan chart of 20 processed cloudlets on 10 different iterations

The performance evaluation of the proposed I-RASA was shown in Table 6 where the makespan of 20 processed cloudlets were taken alongside with two other algorithms for comparism. It was shown that from ten (10) different iterations, the makespan of the proposed algorithm is still the lowest, followed by improved max-min and lastly RASA. Fig 16 shows the bar-chart of the makespan difference for the ten iterations.

8. CONCLUSION

Cloud controls the lifting of computing-intensive jobs in cloud computing thereby placing enormous amounts of data on the platform especially in mobile cloud computing. Both the data processing and data storage are done in the cloud external of the mobile devices. As mobile applications increase, the leverage on the computing power of the cloud tends to increase as well, it therefore becomes imperative to efficiently manage computing resources for these applications for improving the performance. One of the criterions to improve the performance is by achieving a minimized makespan during load balancing. The study proposed a new load balancing technique, which is an improvement over RASA. The makespan is reduced compared with RASA and Improved Max-Min. The proposed algorithm is however, having flow time more than the RASA, further research can be done on this in the future. The performance analysis from the presented results proved that our proposed approach is efficient in optimizing task scheduling and load balancing.

DISCLAIMER

The products used for this research are commonly and predominantly use products in our area of research and country. There is absolutely no conflict of interest between the authors and producers of the products because we do not intend to use these products as an avenue for any litigation but for the advancement of knowledge. Also, the research was not funded by the producing company rather it was funded by personal efforts of the authors.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Proshikshya M, Prasant K, Tanmaya S, Amlan D. Task scheduling algorithm based on multi criteria decision making method for cloud computing environment: TSABMCDMCCE, Open Comput. Sci.; 2019,9:279–291.
2. Nayak S, Parida S, Tripathy C. Modeling of task scheduling algorithm using petri-net in cloud computing, progress in advanced computing and intelligent engineering. Advances in Intelligent Systems and Computing, Springer, Singapore. 2018;563: 633–643.
3. Sumanpreet K, Navtej S. Review on dynamic resource allocation based on lease types in cloud environment, International Journal of Computers & Technology. 2017;16:7581-7585.
4. Saeed P, Reza E. RASA: A new grid task scheduling algorithm, International Journal of Digital Content Technology and its applications. 2009;3:91-99.
5. Neelima P, Reddy ARM. An efficient load balancing system using adaptive dragonfly algorithm in cloud computing. Cluster Computing. 2020;23(1):2891-2899.
6. Muthusamy G, Chandran SR. Task scheduling using artificial bee foraging optimization for load balancing in cloud data centers. Comput Appl Eng Educ. 2020;28:769– 778.
7. Jena UK, Das PK, Kabat MR. Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. Journal of King Saud University – Computer and Information Sciences; 2020. Available:https://doi.org/10.1016/j.jksuci2020.01.012
8. Semmoud A, Hakem M, Benmammar B, and Charr J-C. Load balancing in cloud computing environments based on adaptive starvation threshold. Concurrency and Computation: Practice and Experience. 2020;32(11):259-277.
9. Arora P, Dixit A. An elephant herd grey wolf optimization (EHGWO) algorithm for load balancing in cloud. International Journal of Pervasive Computing and Communications. 2020;16(3):259-277.
10. Abdulquadri OS, Ravi G. Dual objective task scheduling algorithm in cloud environment. International Journal in Advanced Trends in Computer Science and Engineering. 2020;9(3):2527-2534.
11. Elzeki O, Reshad M, Elsoud M. Improved max-min algorithm in cloud computing. International Journal of Computer Applications. 2012;50(12):22-27.
12. Pawan K, Rakesh R. Issues and challenges of load balancing techniques in cloud computing: A survey. ACM Computing Surveys (CSUR) Volume 2019;51(6).

13. Udayraj P, Hemant G. Review of load balancing technique in cloud computing, IJRAR- International Journal of Research and Analytical Reviews, 2019;6(2):826-833.
14. Wang S, Yan K, Chen C. A three-phases scheduling in a hierarchical cloud computing network, in: Communications and Mobile Computing (CMC), 2011 Third International Conference on IEEE. 2011; 114–117.
15. Neetesh K, Deo P. A green SLA constrained scheduling algorithm for parallel/scientific applications in hetero-geneous cluster systems. ELSEVIER, Sustainable Computing: Informatics and Systems. 2019;22:107-119.
16. Bhoi U, Ramanuj P. Enhanced max-min task scheduling algorithm in cloud computing. International Journal of Application or Innovation in Engineering and Management (IJAIEM). 2013;2319—4847.
17. Venubabu K. Dynamic load balancing for the cloud. International Journal of Computer Science and Electrical Engineering; 2012.
18. Danuta S, Ignacio C, Deepak M, Barry O. On energy- and cooling-aware data centre workload management. IEEE. 2015;1111-1114.
19. Che-Lun H, Hsiao-hsi W, Yu-Chen H. Efficient load balancing algorithm for cloud computing network. IEEE. 2012;9: 70-78.
20. Zhi Z, Fangming L, Ruolan Z, Jiangchuan L, Hong X, Hai J. Carbon-aware online control of geo-distributed cloud services. IEEE. 2015;1-14.
21. Mao Y, Chen X, Li X. Max-min task scheduling algorithm for load balance in cloud computing. Proceedings of International Conference on Computer Science and Information Technology; Springer; 2014.
22. Li X, Mao Y, Xiao X, Zhuang Y. An improved max-min task-scheduling algorithm for elastic cloud. Computer, Consumer and Control (IS3C), 2014 International Symposium on; 2014: IEEE.
23. Morton A. IO scheduler benchmarking. linux-kernel (Mailing list). Archived from the original on 2 June 2007; 2003. Retrieved 23rd May 2007.
24. Iyer S, Druschel P. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. 18th ACM Symposium on Operating Systems Principles; 2001.
Retrieved 20th April, 2010.
25. George D. Amalarethinam, Muthulakshmi P. An overview of the scheduling policies and algorithms in grid computing. Inter-national Journal of Research and Reviews in Computer Science. 2011;2(2):280-294.
26. Fatos Xhafa, Ajith A. Computational models and heuristics methods for grid scheduling problems. Future Generation Computer systems. 2010;26:608-621.
27. Casavant T, Kuhl J. A taxonomy of scheduling in general purpose distributed computing systems. IEEE Trans on Software Engineering. 1988;14(2):141-154.
28. Buyya R, Ranjan R, Calheiros R. Modeling and simulation of scalable cloud computing environments and the CloudSim Toolkit: Challenges and opportunities. In Inter-national Conference on High Performance Computing and Simulation (HPCS); 2009.
29. Singh A, Goyal P, Batra S. Optimized round robin scheduling algorithm for CPU scheduling. International Journal on Computer Science and Engineering. 2010; 02(07):2383-2385.
30. Neha G, Parminder S. Load balancing using genetic algorithm in mobile cloud computing. International Journal of Innovations in Engineering and Technology (IJIET). 2014;1(4).