Load Balancing Approach to Enhance the Performance in Cloud Computing

Iehab AL Rassan[†] and Noof Alarifi [2],

Department of Computer Science, College of Computer and Information Sciences
King Saud University, Riyadh, Saudi Arabia

Abstract:
Virtualization technologies are being adopted and broadly utilized in many fields and at different levels. In cloud computing, achieving load balancing across large distributed virtual machines is considered a complex optimization problem with an essential importance in cloud computing systems and data centers as the overloading or underloading of tasks on VMs may cause multiple issues in the cloud system like longer execution time, machine failure, high power consumption, etc. Therefore, load balancing mechanism is an important aspect in cloud computing that assist in overcoming different performance issues. In this research, we propose a new approach that combines the advantages of different task allocation algorithms like Round robin algorithm, and Random allocation with different threshold techniques like the VM utilization and the number of allocation counts using least connection mechanism. We performed extensive simulations and experiments that augment different scheduling policies to overcome the resource utilization problem without compromising other performance measures like makespan and execution time of the tasks. The proposed system provided better results compared to the original round robin as it takes into consideration the dynamic state of the system.

Key words:
Cloud Computing, Load Balancing, Round Robin, Virtual Machine

1. INTRODUCTION

Within the recent decade, major innovations in the field of network technology have emerged, that potentially add more convenience to daily life practices not only on an enterprise level but on an individual level as well. Cloud computing technology has witnessed significant advances in its implementation and become widely adopted by both private and public sectors since it provides different resources over the Internet. Cloud computing allows sharing hardware and software resources like networks, servers, storage, software, applications, etc. all over the Internet since it is an Internet-based computing model [1].

It was obvious recently that a lot of organizations and enterprises are transferring their workloads to the cloud. Therefore, providing the highest computational performance for the intended users is mandatory. However, one of the most common problems that affects the performance is the overloading or underloading the cloud system with different load of tasks. In cloud systems, different nodes might be assigned with uneven loads of tasks where some nodes are overloaded and other nodes underloaded, which may affect the overall performance and cause multiple issues like longer execution time, machine failure, high power consumption, etc. Therefore, load-balancing mechanism is an essential aspect in cloud computing, which is concerned with detecting the nodes that are overloaded or underloaded with tasks and balance the load among them.

Using an efficient load balancing mechanism is an important aspect in cloud computing that assists in overcoming different performance issues. In this research, a proposal for a new approach that combines the advantages of Round robin algorithm with a threshold technique. The proposed design expects to enhance scheduling policies to overcome the resource utilization problem without compromising other performance measures: makespan and completion rate.

One of the most significant aspect of cloud computing is the virtualization technology where there is a several virtual machines that might run different operating system on one physical host. Virtualization is achieved through the help of hypervisor; also called virtual machine monitor (VMM) which can be placed over the hardware directly or upon the operating system of the host machine. Different tasks are allocated to different physical machines then allocated to different virtual machines on the respective host or physical machine. The virtual machine migration is the process of transferring a virtual machine to another physical machine in order to improve the resource utilization in case of the physical machine was overloaded. The task migration is the process of transferring a task from one virtual machine to another virtual machine either on the same physical machine or to a virtual machine on another physical machine. Virtual machine migration and task migration play an important role in load balancing in cloud computing. Generally, the load balancing in cloud

Manuscript received February 5, 2021
Manuscript revised February 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.2.18

computing is concerned with balancing the load among different nodes that can be either virtual machines or physical machines which have uneven amounts of loads [2].

The reminder of this paper organized as follows: section 2 includes a detailed background about the cloud computing architecture, and load balancing algorithms. Section 3 presents a review of the existing works on load balancing in cloud systems. Section 4 includes the proposed approach for achieving the load balance among different targeted nodes. Section 5 explains the results and discussion of the research . Section 6 is the conclusion of the paper.

2. BACKGROUND

In this section, we have highlighted a variety of concepts related to the load balancing in cloud computing. Section 2.1 presents cloud computing architecture and its components. Section 2.2 includes a review of the existing load balancing algorithm.

2.1. Cloud Computing Architecture

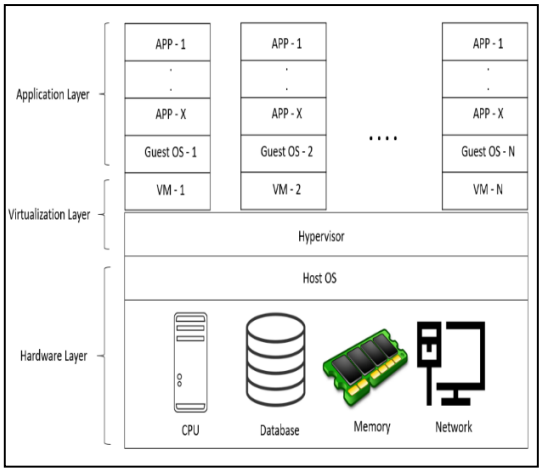


Figure 1. Architecture of cloud computing

Figure 1 shows the architecture of cloud computing that consists of the layers of hardware resources, which includs network bandwidth, main memory, processor, and secondary storage. Hypervisor such as Denali, VMWare, Xen, UML, or Virtual Machine Monitor will act as a crossing point between the Virtual machines and the guest operating system. This Virtual Machine supports numerous operating systems that simultaneously execute applications in only one hardware platform. Various heterogeneous applications run on every virtual machine. {VM1; VM2; VM3; ….; VMn} be the set of VMs used in the cloud hosts [1].

The data center of the cloud consists of fixed number of various physical hosts. The identification of every host is through its lists of processing elements, bandwidth, identification number, memory size, and speed for processing in terms of MIPS. Every host contains numerous virtual machines. Just like the host, the virtual machine has the same attribute. The activities arriving from various users to the serial loader or central load balancer for mapping of the cloud resources. Every computing node that is the virtual machine performs tasks execution at a time. In case there is a request, the load balancer allocates it to one of the virtual machines, if there are enough resources are available to complete within a specific period or else the task will have to wait if the SLA allows. Once the task's execution is completed, the resources that are used in the corresponding virtual machine is released and can be used to form a new virtual machine that can be used to work on a new task [1].

Cloud computing is concerned with services, software, and hardware provisioning from third parties via the use of a network. Any cloud computing system contains substantial components like datacenter, distributed servers, as well as clients, Figure 2 shows the components of cloud computing [3]:

a) Datacenter: is a group of servers hosting various applications. The end-user links to the datacenter to subscribe to various applications. It may be on a big remoteness from the clients' position.

b) Distributed servers: are part of a cloud that are accessible through the internet. The server hosts various applications, while using cloud applications, the users may feel that they are using these applications from their own computers.

c) Client: the end-users communicate with the clouds for information management associated with the cloud.

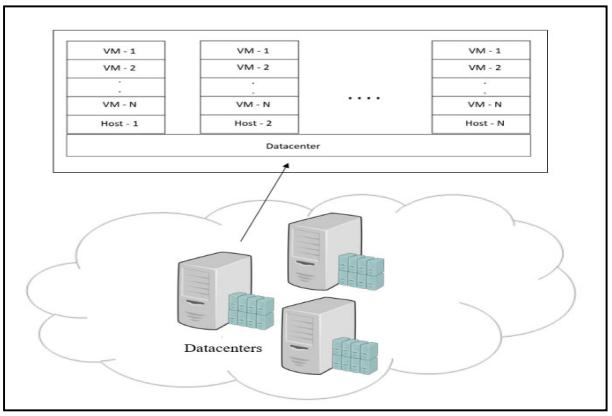


Figure 2. Overview of cloud components

2.3. Load Balancing Algorithms

Based on the current state of the system, load-balancing algorithms can be divided into two categories [4]:

- Static: decisions on the load balancing doesn't depend on the current state of the system. It needs prior information about the average behavior of the system.
- Dynamic: decisions on load balancing are made depending on the current state of the system, which means that more benefits can be achieved compared to the static policies. It is more complex since no prior information is used.

2.3.1. Brief review of the current algorithms of load balancing

- Round Robin

The processes in this algorithm are divided between all processors. In a round robin order, every task is allocated to a processor. The allocation process kept locally autonomous of the distributions from distant processors. The workload allocation amongst the processors are the same but the processing time of each job is different, one job may take longer execution time than the others, at any time some nodes may be loaded heavily, and others continue to be idle or lightly loaded. Round robin is a static algorithm that does not need inter process communication, which results in a less system overhead [5].

- Central queuing

Central queuing algorithm is a dynamic allocation algorithm. Each new arriving activity is inserted into the queue. Once the queue manager receives the requests for an activity, it deletes the first activity from the queue and directs it to the requester. If there is no activity ready in the queue, the request is buffered till a new activity is available. In case a new activity comes to the queue whereas there are requests that have not been answered, the first request is detached from the queue and a new activity is allocated to it [6].

- Minimum execution time

It is also known as User Directed Assignment as well as Limited Best Assignment. Both dynamic and static strategies are used in this heuristic technique. The minimum execution time algorithm was created to assign every task to a virtual machine based on the least execution time by using the computation of Expected time to Compute (ETC) in order to execute the entire tasks within a given time for execution [1]. Min–Min

The primary procedure of this technique is to select the task with the smallest size as well as a virtual machine with the least resource or capacity. Once the task allocation to a virtual machine is done, the task is deleted from the queue and carry on with the allocation of all tasks that are unassigned. This algorithm is appropriate for distributed systems that are only small-scale. The improved Min-min algorithm will optimize the make-span as well as improve the utilization of resource. The load balance improved Min-Min algorithm will divide tasks at first into two categories A, B according to their priority. B is only for the tasks that are lower, and A is for tasks of greater priority. All tasks of A are scheduled by the algorithm, and then the tasks in B. The function of load balancing is to maintain the load on every specific machine to produce a better schedule [1].

- Min-Max algorithm

It is the same as the Min-Min. In the cloud computing, the primary procedure of Max-Min is to assign the task that is larger in size to a virtual machine with the least resource or capacity. Once the tasks distribution to a virtual machine is done, it is deleted from the queue then continue with the rest of unallocated tasks. This algorithm is appropriate for distributed systems of small-scale only. To accomplish load balancing, the improved Max-Min that holds a task table status to estimate the virtual machine's load in real-time and estimated completion time of the tasks. The proposed algorithm Elastic Cloud Max-Min is betters as compared to the Round Robin method in term of average task pending time [1].

3. PREVIOUS WORK

Achieving load balancing across large, distributed servers is considered a complex optimization problem with an essential importance in cloud computing systems and data centers. Existing schedulers often incur a high communication overhead when collecting the data required to make scheduling decisions, hence delaying job requests on their way to the executing servers.

Many approaches have been developed to overcome the problem of uneven load distribution among nodes in cloud computing. An approach for load balancing is proposed in [7], which is based on Bat Algorithm where it is supposed to find the optimal host as well as VM for any incoming task. The BAT algorithm is enforced by a load balancer in case there is a task that has arrived in the job pool where the algorithm will select a server that matches the incoming task. The main factor considered when applying the BAT algorithm is the type of the task and the required resources for the excellent assignment execution. The current server is assigned the task after the necessary server identification. In case of a load that is higher than servers available, there is task redistribution to more than one server. The approach of the BAT algorithm has been applied for load balancing performance and reducing the time of response without delay. The proposed BAT algorithm was evaluated by comparing it with fuzzy and GSO and round-robin.

In [8] , the authors introduced a scheduling approach based on Load Balancing Ant Colony Optimization (LBACO) algorithm. The main goal was trying to balance the system load and reducing the makespan, which is the total time needed to complete all tasks submitted to the system. For the evaluation process, the results of the proposed LBACO algorithm was compared to FCFS (First Come First Serve) and the basic ACO (Ant Colony Optimization) using simulations.

Different approach is proposed in [9] that consider combining the fuzzy logic method and the Glow Worm Swarm Optimization method (GSO). Fuzzy machine logic is used for assigning the fixed-rate arrived tasks to virtual machines; contrary, the GSO technique is applied. Excellent results have been realized by using Fuzzy logic and GSO load balancing than when using the round-robin technique.

The work in [10] intends to measure the optimization of cloud computing performance. Hence, there are two significant approaches to scheduling. There are two primary scheduling techniques applied with the help of Cloudsim simulator: Round Robin and the first come first served. The round-robin approach is a time-shared concept where a fixed amount of time is assigned to a given job for all resources. The FCFS approach works depending on the space shared manner whereby tasks are allocated depending on their appearance in the array. The two techniques' performance are compared and measured. The comparison of the performance calculations is done by the use of a simulation trace where the average processing time, average waiting time and utilization of the CPU computations are considered. The round-robin approach is proven to be more efficient in all criterions.

In [11] the authors proposed a hybrid scheduling for the different application type: workflows and batch jobs. The suggested algorithm takes into consideration the clustering of available resources. Execution of jobs takes place in two phases: there is tasks allocation to resources in groups in the first phase. The second phase algorithm scheduling is classical for every resources group. The algorithm is recommended for heterogeneous distributed computing, such as high-performance systems where exists various requirements for modelling applications. Authors evaluated the performance in a Cloudsim tool with respect to load-balancing, cost savings, workflow assurance dependency and efficiency of computation, and investigated these metrics at runtime.

As virtualization technologies are being adopted and broadly utilized in many fields and at different levels, it has a huge significance and there must be an efficient virtual machine load balancing and migration schemes to serve as a tool for managing cloud resources and achieve the primary goals like maintaining the load balance, reducing the failure possibilities, resource utilization and so on.

4. METHODOLOGY

In cloud computing, load balancing is required to distribute the workload evenly across all the nodes. Using a proper load balancing algorithm leads to minimizing the resource consumption and overprovisioning of resources, which improves the performance of the system in different aspects and helps to achieve higher user satisfaction.

In this paper, we propose a method that combines the advantages of Round robin with a threshold technique. Thus, the proposed scheduler enhances scheduling policies to overcome the resource utilization problem and to reduce the makespan of the system. An overview of the environment and its components is presented in the next subsection.

4.1. System overview and main components

A typical environment for modeling cloud systems usually involves datacenters, set of host machines and a set of virtual machines.

- Datacenter: is centralized place that involves numerous servers where computing and networking is taking place in the cloud.
- Host: Host machines are connected to the datacenter. Hosts involves set of virtual machines.
- VM: is virtual machine that is applied on a physical machine. Every virtual machine may run different OS than the physical machine and have their own resources.

There are an n input of tasks (T) from $1\ to\ n$ $(T_1, T_2, \ldots, T_n)$, and N numbers of virtual machines (VM) from $1\ to\ N$ $(VM_1, VM_2, \ldots, VM_N)$. Load balancer is essentially based on two characteristics. First, assign load to the best candidate node, and second, migration of load from heavily loaded VMs to lightly loaded VMs.

4.2. Proposed work

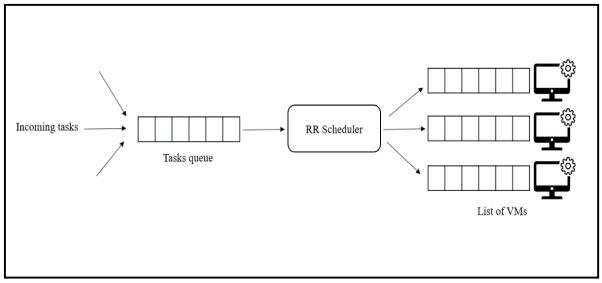


Figure 3. Tasks distribution using RR in a host

Figure 3 shows the distribution of tasks across virtual machines using round robin in one host. Round Robin is a simple scheduling algorithm as it distributes the load evenly among the existing VMs in a round robin style.

Round Robin results in less system overhead; however, it does not take into consideration the different amount of resources that were allocated to different VMs, which in return might cause some lacking in other performance metrics like makespan, resource utilization and completion rate of the tasks.

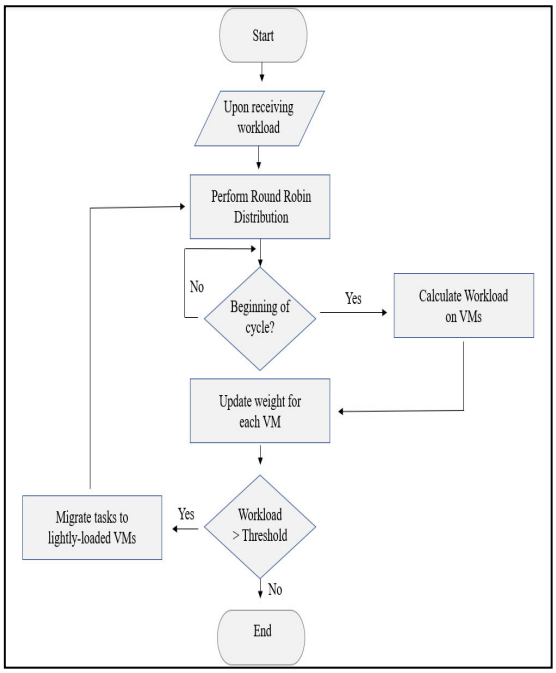


Figure 4. Flowchart of the proposed work in one host

Round robin can be adequate in small systems when all VMs have the same resource configurations. However, in this approach we assume that all VMs have different resources. Thus, after the distribution of the first load in a round robin style, the utilization of the resources is calculated, then we are going to use the weighted round robin algorithm, where each VM is assigned with a weight that indicates its amount of available resources and based on this weight the load will be distributed in a round robin style. This assures the even distribution of load. However, it doesn't necessarily improve the makespan or completion rate as the load might be so heavy on some or all existing VMs.

Therefore, a threshold is defined, when this threshold is exceeded, then the tasks should be migrated either to another VMs in the same host, or to a VM in another host. First, the VMs in the same host will be checked out as some tasks might have finished their execution, if not then, the tasks should be migrated to another host.

In case all the VMs on the same host are overloaded, the queue of the incoming tasks should stop assigning these tasks to the VMs until they get lighter in load or migrate these tasks to another VM on another host. The host will be picked up according to its load-level state, which is kept and updated in a table in the datacenter. The state of the host is represented by the number of the idle VMs divided by the total number of VMs on that host. which will provide a percentage that represent the state of the host. The states of different hosts in the datacenter will be arranged in ascending order where the lightest-loaded host is placed on the top, and this order is updated in every time step.

4.3. Evaluation

There are different performance metrics that can be affected from the mapping of tasks to the virtual machines. The following performance metrics are considered in analyzing the performance of the proposed load balancing approach:

- Completion Rate:

The rate of the completed jobs will be calculated as the ratio between the number of requested jobs and the number of completed jobs:

$$Completion\ Rate - \frac{Number\ of\ Requested\ Jobs}{Number\ of\ Completed\ Jobs}$$

- Makespan:

This is the amount of time spent between the start and end of all tasks executed by the system. In this case, makespan is the total or maximum time taken by the host to run all tasks. Makespan (MS) is the maximum of ET_f, calculated as follows in [1] :

$$MS = Max.[ET_f]_{i=1}^{n}$$

First the Expected Time to Compute is calculated as:

$ETC_{ij} = \frac{L_i}{P_j}$, where L_i is the length of the task in terms of Million instructions (MI), and P_j is the processing speed of the VM in terms of MIPS. Then the Execution Time of VM is calculated as:

$$X_{ij} - \begin{cases} 1 & if\ T_i\ \text{is allocated to}\ VM_j \\ 0 & if\ T_i\ \text{is not allocated to}\ VM_j \end{cases}$$

$$ET_j = \sum_{i-1}^{n} X_{ij} \times ETC_{ij}$$

We assume that the metrics of the proposed system will provide better results compared to the original round robin as it takes into consideration the dynamic state of the system.

5. RESULTS AND DISCUSSIONS

In this section, we are providing an experimental result in order to achieve the objectives of this research.

For implementation, we used CloudSim, which is a tool (library) that works with any programming IDEs that support Java, in our case, we are using Eclipse. CloudSim enables modeling and simulation of cloud computing systems [5],[19].

Table 1. Simulation Parameters

Simulation Parameters		
Parameter	Value	Range
Number of tasks	(60, 100, 1000)	(25 to 8000 KB)
Number of VMs	(5,10,15)	(600 to 1000 MIPS)
Bandwidth	1000 MBPS	
Number of CPU per VM	1	
RAM per VM	512	

5.1 Simulation Setup

We run simulations using different number of tasks (60, 100, 1000) with random complexities, first with 5 VMs, 10 VMs , and then with 15 VMs. These VMs have different processing power in terms of CPU (600 to 1000 MIPS). The bandwidth is 1000 MBPS, and for each VM there is 1 CPU. We run these parameters consecutively to measure the makespan of the system and the average execution time in different scenarios. Table1 summarizes these parameters.

5.2 Simulation Results

We performed eight experiments using different values of tasks and VMs. In each experiment, we will input different number of tasks between 60 to 1000 task with: 5 VMs, 10 VMs, 15 VMs consecutively. In the first experiment, we used Round Robin mechanism to allocate the tasks to VMs and using the VM utilization as a threshold with sorting the hosts in ascending order according to their utilization. In the second experiment, we used Round Robin mechanism to allocate the tasks to VMs, and using the VM utilization as a threshold, but without sorting the hosts. In the third experiment, we used Random mechanism to allocate the tasks to VMs, and using the VM utilization as a threshold with sorting the hosts in ascending order according to their utilization. In the forth experiment, we used Random mechanism to allocate the tasks to VMs, and using the VM utilization as a threshold, but without sorting the hosts. In the fifth experiment, we used Random mechanism to allocate the tasks to VMs, and using the VM's allocation count as a threshold with sorting the hosts in ascending order according to their utilization. In the sixth experiment, we used Random mechanism to allocate the tasks to VMs, and using the VM's allocation count as a threshold without sorting the hosts. In the seventh experiment, we performed the basic LCM algorithm. In the eighth experiment, we performed the original Round Robin algorithm. Table 2 shows the experiments that we are simulating in this section.

Experiment	Allocation Mechanism	Threshold Value	Host Sorting
1	Round Robin	VM CPU Utilization	Yes
2	Round Robin	VM CPU Utilization	No
3	Random	VM CPU Utilization	Yes
4	Random	VM CPU Utilization	No
5	Random	VM's Allocation Count	Yes
6	Random	VM's Allocation Count	No
7	Least Connection Mechanism	-	-
8	Original Round Robin	-	-

Table 2. Simulation experiments

5.2.1 Experiment 1 (RRH)

In this experiment, we used Round Robing Allocation with Host Sorting (RRH). The load distribution is performed using Round Robin. Then the VM utilization is calculated. Then the VM's CPU utilization is calculated by summing all the amount of CPU utilized by each task on that VM, if the amount of utilized resources is bigger than the threshold (75% of VM's CPU) then it is considered overloaded, and if the VM utilization is less than (25% of VM's CPU) then the VM is considered under loaded. In case, of overload, the tasks will be migrated to another VM, either on the same or different host, and the hosts are sorted in ascending order according to their utilization. Table 3 and figure 5 show the results of this experiment.

Table 3. RRH Results in term of makespan and average execution time

Makespan in Seconds for RRH			
Number of VMs	Number of tasks		
	60	100	1000
5 VMs	6.382	8.743	63.827
10 VMs	5.574	7.650	28.115
15 VMs	6.989	6.098	16.382
Average Excusión Time in Milliseconds for RRH			
Number of VMs	Number of tasks		
	60	100	1000
5 VMs	315.419	324.454	376.053
10 VMs	266.600	326.064	384.568
15 VMs	385.693	296.226	248.695

Figure 5. Representation of RRH Results in term of makespan and average execution time

5.2.2 Experiment 2 (RRV)

In this experiment, we used Round Robing Allocation to VMs (RRV). The load distribution is performed using Round Robin. Then The VM's CPU utilization is calculated by summing all the amount of CPU utilized by each task on that VM, if the amount of utilized resources is bigger than the threshold (75% of VM's CPU) then it is considered overloaded, and if the VM utilization is less than (25% of VM's CPU) then the VM is considered underloaded. In case, of overload, the tasks will be migrated to another VM either on the same or different host. In this experiment, the tasks will be allocated to any Idle VM without host sorting. Table 4 and figure 6 show the results of this experiment.

Table 4. RRV Results in term of makespan and average execution time

Makespan in Seconds for RRV			
Number of VMs	Number of tasks		
	60	100	1000
5 VMs	7.465	6.727	8.405
10 VMs	6.726	6.028	7.914
15 VMs	5.914	7.046	7.635
Average Execution Time in Milliseconds for RRV			
Number of VMs	Number of tasks		
	60	100	1000
5 VMs	289.182	348.369	352.882
10 VMs	246.979	313.959	380.734
15 VMs	387.352	321.232	237.965

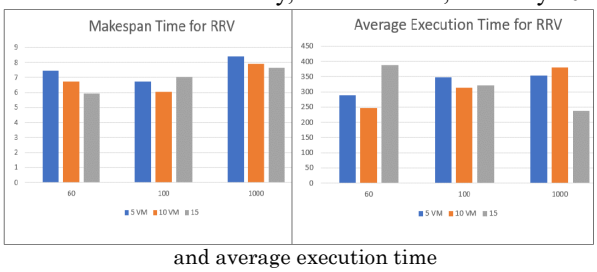


Figure 6. Representation of RRV results in term of makespan and average execution time

5.2.3 Experiment 3 (RAH)

In this experiment, we used Random Allocation with Host Sorting (RAH). The load distribution is performed by using Random Allocation. Then The VM's CPU utilization is calculated by summing all the amount of CPU utilized by each task on that VM, if the amount of utilized resources is bigger than the threshold (75% of VM's CPU) then it is considered overloaded, and if the VM utilization is less than (25% of VM's CPU) then the VM is considered underloaded. In case, of overload, the tasks will be migrated to another VM either on the same or different host, and the hosts are sorted in ascending order according to their utilization. Table 5 and figure 7 show the results of this experiment.

Table 5. RAH Results in term of makespan and average execution time

Makespan in Seconds for RAH			
Number of VMs	Number of tasks		
	60	100	1000
5 VMs	6.528	6.320	8.061
10 VMs	5.938	7.709	6.984
15 VMs	7.649	6.670	7.983
Average Execution Time in Milliseconds for RAH			
Number of VMs	Number of tasks		
	60	100	1000
5 VMs	258.716	286.693	363.984
10 VMs	224.732	309.513	363.158
15 VMs	376.878	295.851	258.197

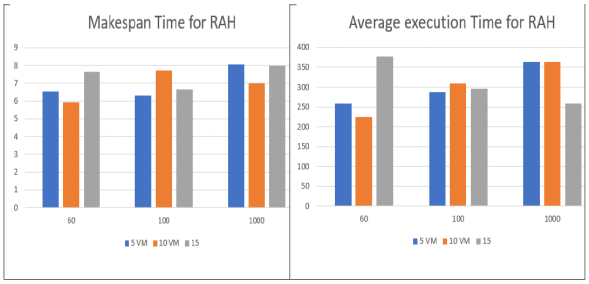


Figure 7. Representation of RAH Results in term of makespan and average execution time

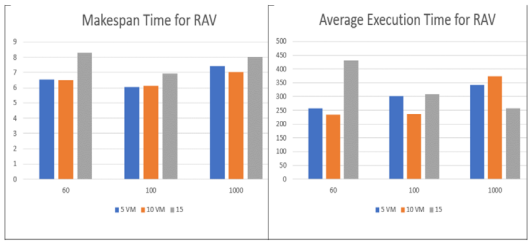
5.2.4 Experiment 4 (RAV)

In this experiment, we used Random Allocation to VMs (RAV). The load distribution is performed by using Random Allocation. Then The VM's CPU utilization is calculated by summing all the amount of CPU utilized by each task on that VM, if the amount of utilized resources is bigger than the threshold (75% of VM's CPU) then it is considered overloaded, and if the VM utilization is less than (25% of VM's CPU) then the VM is considered underloaded. In case, of overload, the tasks will be migrated to another VM either on the same or different host, in this experiment, the tasks will be allocated to any Idle VM without host sorting. Table 6 and figure 8 show the results of this experiment.

Table 6: RAV Results in term of makespan and average execution time

Makespan in Seconds for RAV			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	6.548	6.055	7.432
10 VMs	6.515	6.105	7.038
15 VMs	8.277	6.915	8.014
Average Execution Time in Milliseconds for RAV			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	256.806	301.283	341.367
10 VMs	233.791	235.911	374.731
15 VMs	431.172	307.669	257.110

Figure 8. Representation of RAV results in term of makespan and average execution time



5.2.5 Experiment 5 (RCH)

In this experiment, we used Random Allocation and Count with Host Sorting (RCH). Table 7 and figure 9 show the results of this experiment. The load distribution is performed by using Random Allocation. This mechanism uses the count of allocations on each VM to define the threshold, if the allocation count is bigger than 75% of the average allocations to all VMs, then the VM is overloaded, if it is less than 25% then it is underloaded.

Table 7. RCH Results in term of makespan and average execution time

Makespan in Seconds for RCH			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	6.998	7.131	7.612
10 VMs	6.797	6.254	6.937
15 VMs	6.964	7.080	8.044
Average Execution Time in Milliseconds for RCH			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	257.997	305.167	352.740
10 VMs	229.232	286.378	396.967
15 VMs	368.160	332.968	246.896

Figure 9. Representation of RCH Results in term of makespan and average execution time

It uses the number of allocations counts on each VM and assign the new tasks to the VM that has the least number of requests. In case, of overload, the tasks will be migrated to another VM either on the same or different host, and the hosts are sorted in ascending order according to their utilization.

5.2.6 Experiment 6 (RCV)

In this experiment, we used Random Allocation and Count to VMs (RCV). The load distribution is performed by using Random Allocation. This mechanism uses the count of allocations on each VM to define the theshold, if the allocation count is bigger than 75% of the average allocations to all VMs, then the VM is overloaded, if it is less than 25% then it is underloaded. it uses the number of allocations counts on each VM and assign the new tasks to the VM that has the least number of requests. In case, of overload, the tasks will be migrated to another VM either on the same or different host, in this experiment, the tasks will be allocated to any Idle VM without host sorting. Table 8 and figure 10 show the results of this experiment.

Table 8: RCV Results in term of makespan and average execution time

Makespan in Seconds for RCV			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	7.160	7.252	61.840
10 VMs	5.871	7.565	7.176
15 VMs	7.262	7.895	9.624
Average Execution Time in Milliseconds for RCV			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	314.597	279.944	317.757
10 VMs	268.107	307.766	344.363
15 VMs	359.530	312.466	257.176

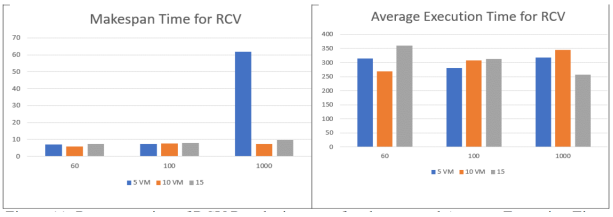


Figure 10. Representation of RCV Results in term of makespan

and average execution time

5.2.7 Experiment 7 (LCM)

In this experiment, we used Least Connection Mechanism (LCM). which is an algorithm for load balancing that is considered as a dynamic scheduling algorithm. It has to calculate the number of connections (allocation count) to each VM dynamically then the allocation count number of every VM is recorded by the load balancer. The allocation count number grows when a new task is transmitted to the VM and is decreased when the task finishes its execution. Multiple simulation runs were performed with different values of tasks and VMs. In each run we input different number of tasks (60 -100 - 1000) tasks with different number of VMs (5 VMs, 10 VMs, 15 VMs) consecutively, in order to measure the makespan and the average execution time. Table 9 and figure 11 show the results of this experiment.

Table 9. Least Connection Results in term of makespan and average execution time

Makespan in Seconds for Least Connection Mechanism			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	7.465	6.727	8.405
10 VMs	6.726	6.028	7.914
15 VMs	5.914	7.046	7.635
Average Execution Time in Milliseconds for LCM			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	251.749	346.954	401.593
10 VMs	236.586	302.347	347.504
15 VMs	355.762	304.028	243.268

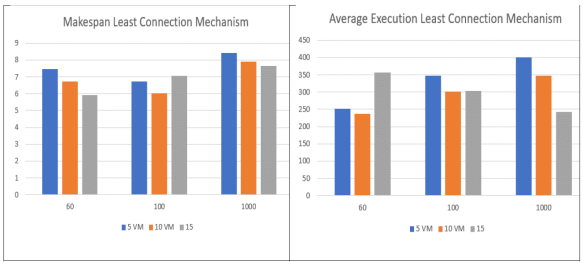


Figure 11. Representation of LCM eesults in term of makespan and average execution time

5.2.8 Experiment 8 (RR)

In this experiment, we used the Original Round Robin (RR), which is a simple scheduling algorithm, it distributes the load evenly among the existing VMs in a round robin style, multiple simulation runs were performed with different values of tasks and VMs. In each run we input different number of tasks (60 -100 - 1000) tasks with different number of VMs (5 VMs, 10 VMs, 15 VMs) consecutively, in order to measure the makespan and the average execution time in Round Robin case. Table 10 shows the results of this experiment. Figure 12 represents the RR results mentioned in the Table 10.

Table 10. RR Results in term of makespan and average execution time

Makespan in Seconds for RR			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	7.086	8.154	68.375
10 VMs	6.658	8.808	29.368
15 VMs	6.132	7.281	19.563
Average Execution Time in Milliseconds for RR			
Number of VMs	**Number of tasks**		
	60	**100**	**1000**
5 VMs	338.995	297.693	359.559
10 VMs	284.079	320.256	369.045
15 VMs	376.580	291.143	252.090

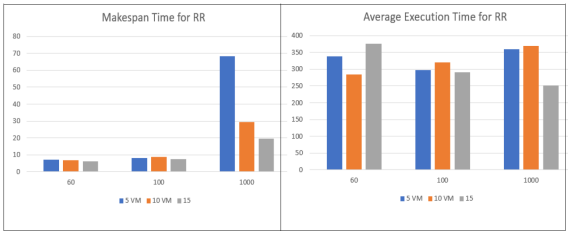


Figure 12. Representation of RR Results in term of makespan and average execution time

5.3. Comparison between the results of all experiments

We performed the eight experiments using different values of tasks and VMs. We run simulations for each experiment using (60, 100, 1000) of tasks that have random complexities, with (5 VMs, 10 VMs and 15 VMs), and these VMs have different processing power in term of CPU (600 to 1000 MIPS).

1) 5 VMs

Table 11 shows the results of the eight experiments that were performed on 5 VMs. These results are compared in term of Makespan. The algorithm provided a quite good performance compared to Round Robin which provided a close performance to other algorithms when the number of tasks was little (60 - 100), However the performance of RR changed dramatically when we executed 1000 tasks, which indicates that Round robin can be adequate in small systems and when all VMs have the same resource configurations. Random allocation to VM (RAV) provided the best performance in term of makespan. This algorithm starts with distributing the load across VMs randomly, Then The VM's CPU utilization is calculated and used as threshold (25% -75% of VM's CPU) to maintain the balance on the VMs, this algorithms does not sort the host according to their idle state, it migrate the tasks to any underloaded VM, for which we assume it provided the best makespan when performed on 5 VMs.

Table 11: Makespan time in seconds comparison between all experiments using 5 VMs

Experiment	Number of tasks		
	60	100	1000
Original RR	7.086	8.154	68.375
Least Connection	7.465	6.727	8.405
RRH	6.382	8.743	63.827
RRV	6.588	9.437	58.110
RAH	6.528	6.320	8.061
RAV	6.548	6.055	7.432
RCH	6.998	7.131	7.612
RCV	7.160	7.252	61.840

For better understanding of the differences, figure 13 shows the results of the eight experiments that were performed on 5 VMs.

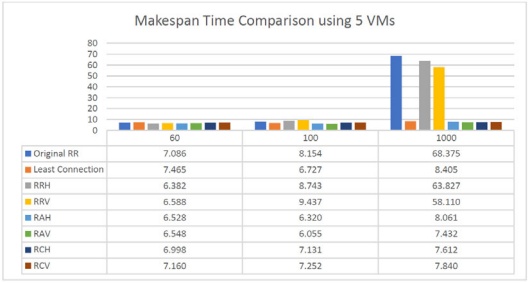


Figure 13. Makespan time in seconds comparison using 5 VMs

Table 12 and figure 14 show the results of the eight experiments that were performed on 5 VMs. These results were compared in terms of average execution time in milliseconds.

Table 12. Comparison using 5 VMs of average execution time in milliseconds

Experiment	Number of tasks		
	60	100	1000
Original RR	338.995	297.693	359.559
Least Connection	251.749	346.954	401.593
RRH	315.419	324.454	376.053
RRV	289.182	348.369	352.882
RAH	258.716	286.693	363.984
RAV	256.806	301.283	341.367
RCH	257.997	305.167	352.740
RCV	314.597	279.944	317.757

In these results we realized that the Least Connection Mechanism (LCM) has provided the best performance in terms of average execution time when

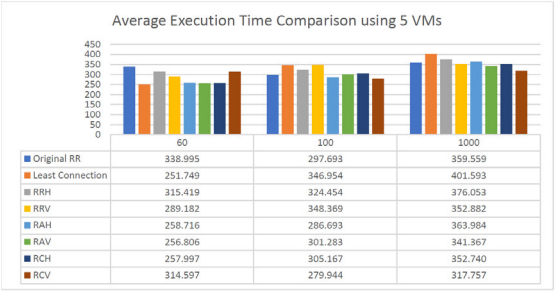


Figure 14. Comparison of average execution time in milliseconds using 5 VMs

the number of tasks were 60. However, when the number of tasks increased to 1000, the performance became the worst among others. LCM calculates the number of connections (allocation count) to each VM and allocate tasks to the VM that has the least number of connections (tasks). LCM does not take into consideration the different processing power between VMs, it only considers the number of allocations, and that why tasks were taking longer execution time in LCM. Least connection algorithm can be a perfect choice for small systems that cares more about the execution time that has a number of tasks between 60-100. However, for systems that needs to execute up to 1000 tasks, RCV would provide the best performance in terms of average execution time as shown in figure 14.

2) 10 VMs

Table 13 and figure 15 show the results of the eight experiments that were performed on 10 VMs. These results were compared in terms of average execution time.

Table 13. Comparison of average execution time in milliseconds using 10 VMs

Experiment	Number of tasks		
	60	100	1000
Original RR	284.079	320.256	369.045
Least Connection	236.586	302.347	347.504
RRH	266.600	326.064	384.568
RRV	246.979	313.959	380.734
RAH	224.732	309.513	363.158
RAV	233.791	235.911	374.731
RCH	229.232	286.378	396.967
RCV	268.107	307.766	344.363

We realized that the algorithms that start with random allocation has the least execution time, RAV and RCH provided the best performance in terms of average execution time when the number of tasks are between 60 – 100, and RAH and RCV outperform when the number of tasks are more that 100 up to 1000 tasks. LCM provided better performance in terms of average execution time when the number of VMs increased from 5 – 10. Figure 15 shows the comparison of average execution time using 10 VMs.

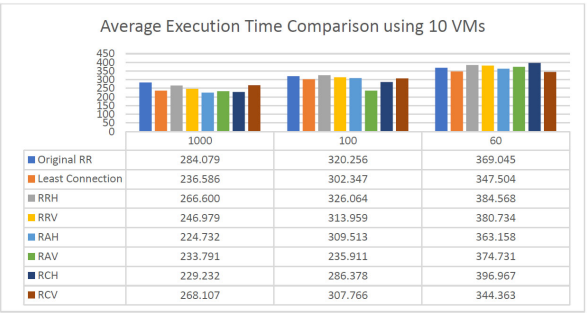


Figure 15. Comparison of average execution time in milliseconds using 10 VMs

Table 14 and figure 16 show the results of the eight experiments that were performed on 10 VMs. These results are compared in terms of Makespan. RCH is providing the best performance in terms of makespan when number of tasks are between 100 -1000 tasks. However, Round Robin did not perform well in terms of makspan when tasks are up to1000.

Table 14. Comparison of makespan time in seconds between all experiments using 10 VMs

Experiment	Number of tasks		
	60	100	1000
Original RR	6.658	8.808	29.368
Least Connection	6.726	6.028	7.914
RRH	5.574	7.650	28.115
RRV	6.454	7.449	26.512
RAH	5.938	7.709	6.984
RAV	6.515	6.105	7.038
RCH	6.797	6.254	6.937
RCV	5.871	7.565	7.176

Figure 16. Comparison of makespan time in seconds using 10 VMs

3) 15 VMs

Table 15 and figure 17 show the results of the eight experiments that were performed on 15 VMs. These results are compared in term of Average execution time.

Table 15: Comparison of average execution time in milliseconds between all experiments using 15 VMs

Experiment	Number of tasks		
	60	100	1000
Original RR	252.090	291.143	376.580
Least Connection	243.268	304.028	355.762
RRH	248.695	296.226	385.693
RRV	237.965	321.232	387.352
RAH	258.197	295.851	376.878
RAV	257.110	307.669	431.172
RCH	246.896	332.968	368.160
RCV	257.176	312.466	359.530

Figure 17. Comparison of average execution time in milliseconds using 15 VMs

Table 16 and Figure 18 show the results of the eight experiments that were performed on 15 VMs. These results are compared in terms of Makespan.

Experiment	Number of tasks		
	60	100	1000
Original RR	6.132	7.281	19.563
Least Connection	5.914	7.046	7.635
RRH	6.989	6.098	16.382
RRV	6.409	6.358	15.543
RAH	7.649	6.670	7.983
RAV	8.277	6.915	8.014
RCH	6.964	7.080	8.044
RCV	7.262	7.895	9.624

Table 17. Comparison of Makespan time in seconds between all experiments using 15 VMs

Figure 18 shows the performance of Round Robin and the algorithms that are extended to Round Robin is also outperformed by others, even though we have increased the number VMs, yet the performance of RR and its extensions did not improve.

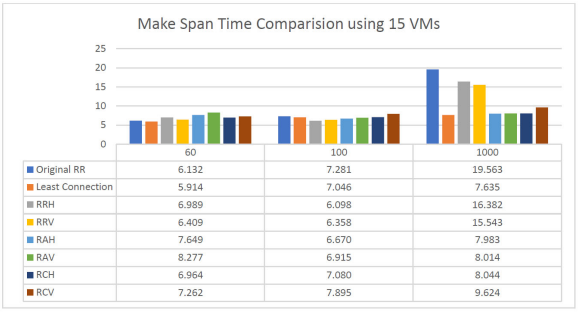


Figure 18. Comparison of Makespan time in seconds using 15 VMs

5.4. Findings

In this section we performed many of simulations in eight different algorithms. In the first and second experiments (RRH) and (RRV), we used Round Robin mechanism to allocate the tasks to VMs, and used the VM utilization as a threshold with sorting the hosts in ascending order according to their utilization in (RRH) and without host sorting in (RRV). In the third and forth experiments (RAH) and (RAV), we used Random mechanism to allocate the tasks to VMs, and used the VM utilization as a threshold , with sorting the hosts in ascending order according to their utilization in (RAH) and without host sorting in (RAV). In the fifth, and sixth experiments (RCH) and (RCV), we used Random mechanism to allocate the tasks to VMs, we used the count of allocations on each VM to define the theshold which is a percentage of the average allocations, with sorting the hosts in ascending order according to their utilization in (RCH) and without host sorting in (RCV). In the seventh experiment we performed the basic LCM, and the last experiment was the original Round Robin. The algorithms that are extended to Random allocation provided better performance than the algorithms that are extended to

Round Robin. (RAV) provided the best performance in terms of makespan when there is a number of tasks up to 1000 tasks with all different numbers of VMs, and LCM provided the best performance in terms of execution time when the number of VMs is more than 10 for all number of tasks between 60- 1000. Both RAV and LCM outperform Round Robin in terms of Makespan and average execution time.

6. CONCLUSION

Cloud Computing has become a real trend in information technology, one of the most significant aspect of cloud computing is the virtualization technology where there is a several virtual machines that might run different operating system on one physical host. Different tasks are allocated to different physical machines then allocated to different virtual machines on the respective physical machine. The critical issue in cloud computing is the balancing of loads across these VMs and achieving an excellent utilization of resources.

The workload must be managed equally across all VMs. There is a need for the development of a new approach for load balancing that overcomes the existing methods drawbacks. Scheduling is an aspect that should be improved to obtain an efficient performance. The objective of scheduling is mapping tasks to resources to optimize one or several objectives. In cloud computing, scheduling belongs to NP-hard problem category as a result of wide solution space. There is no existence of an algorithm that can find the optimal solution within polynomial time. It is recommended to find a suboptimal solution technique for achieving solutions within a reasonable time.

In this research, we proposed a new approach that combines the advantages of different task allocation algorithms like Round Robin algorithm, and Random allocation with different threshold techniques like the VM utilization and the number of allocation counts using least connection mechanism. We performed extensive simulations and experiments with the goal of augmenting different scheduling policies to overcome the resource utilization problem without compromising other performance measures like makespan and execution time of the tasks. To evaluate the proposed work, the results are discussed in the context of makespan and average execution time metrics. The proposed system provided better results compared to the original round robin as it takes into consideration the dynamic state of the system. For future work, we intend to use more than one CPU for each VM, also we will consider the priority of tasks and execute the tasks with higher priority first without preemption and predict the incoming tasks by utilizing a prediction technique.

REFERENCES

[1] M. Mishra, S. Kumar, B. Sahoo and P. P. Parida, "Load balancing in cloud computing: A big picture," Journal of King Saud University-Computer and Information Sciences, 2018.

[2] M. H. Shirvani, A. M. Rahmani and A. Sahafi, "A survey study on virtual machine migration and server consolidation techniques in DVFS-enabled cloud datacenter: taxonomy and challenges," Journal of King Saud University-Computer and Information Sciences, 2018.

[3] Ray, S. and De Sarkar, A., 2012. Execution analysis of load balancing algorithms in cloud computing environment. *International Journal on Cloud Computing: Services and Architecture (IJCCSA)*, *2*(5), pp.1-13.

[4] K. Ramana and M. Ponnavaikko, "AWSQ: an approximated web server queuing algorithm for heterogeneous web server cluster," International Journal of Electrical and Computer Engineering, vol. 9, no. 3, p. 2083, 2019.

[5] D. L. Eager, E. D. Lazowska and J. Zahorjan, "A comparison of receiver-initiated and sender-initiated adaptive load sharing," Performance evaluation , vol. 6, no. 1, pp. 53-68 , 1986.

[6] Z. M. Elngomi and K. Khanfar, "A Comparative Study of Load Balancing Algorithms: A Review Paper," International Journal of Computer Science and Mobile Computing , vol. 5, no. 6, pp. 448-458, 2016.

[7] S. Sharma, A. K. Luhach and S. S. Abdhullah, "An optimal load balancing technique for cloud computing environment using bat algorithm," lndian Journal of Science and Technology, vol. 9, no. 28, 2016.

[8] K. Li, G. Xu, G. Zhao, Y. Dong and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," In 2011 Sixth Annual ChinaGrid Conference, pp. 3-9, 2011.

[9] U. Singhal and S. Jain, "A new fuzzy logic and GSO based load balancing mechanism for public cloud," International Journal of Grid and Distributed Computing, vol. 7, no. 5, pp. 97-110, 2014.

[10] K. Maheshwari and V. K. Gupta, " Load Balancing in VM in Cloud Computing Using CloudSim.," Cloud Computing, 2019.

[11] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea and J. Kołodziej, "Resource-aware hybrid scheduling algorithm in heterogeneous distributed

computing.," Future Generation Computer Systems, p. 51, 2014.

Iehab Al-Rassan received his Ph.D. in Computer Science from the George Washington University. He has more than 20 years of experience in the field of IT, held many lectures and published many papers and articles in this field. His research interests include coding theories, information retrieval, string-matching algorithms, data compression, Business Process Management (BPM), Distributed Systems, Internet of Things, Cloud Computing and Mobile Computing.