Hybridization of the Q-learning and honey bee foraging algorithms for load balancing in cloud environments

Adeyinka Ajao Adewale, Oghorchukwuyem Obiazi, Kennedy Okokpujie, Omiloli Koto
Department of Electrical and Information Engineering, College of Engineering, Covenant University, Ota, Nigeria

Article Info	ABSTRACT
	Load balancing (LB) is very critical in cloud computing because it keeps nodes from being overloading while others are idle or underutilized. Maintaining the quality of service (QoS) characteristics like response time, throughput, cost, makespan, resource utilization, and runtime is difficult in cloud computing due to load balancing. A robust resource allocation strategy contributes to the end user receiving high-quality cloud computing services. An effective LB strategy should improve and deliver required user satisfaction by efficiently using the resources of virtual machines (VM). The Q-learning method and the honey bee foraging load balancing algorithm were combined in this study. This hybrid combination of a load balancing algorithm and a machine learning method has reduced the runtime of load balancing activities and makespan, and increased task throughput in a cloud computing environment thereby enhancing routing activities. It achieved this by continuously tracking the usage histories of the VMs and altering the usage matrix to send jobs to the VMs with the best usage histories. *This is an open access article under the [CC BY-SA](#) license.*

Corresponding Author:

Adeyinka Ajao Adewale
Department of Electrical and Information Engineering, College of Engineering, Covenant University
Km 10 Idoroko Road, Ota, Ogun State, Nigeria
Email: ade.adewale@covenantuniversity.edu.ng

1. INTRODUCTION

In cloud computing, the data being accessed is situated remotely in the cloud. Customers of cloud service providers can access information online and save files and applications on remote servers. This allows the user to access it from any place since they do not need to be in a particular place. The three major services model provided in cloud computing are platform as a service (PAAS), software as a service (SAAS), and infrastructure as a service (IAAS) [1]. Also, it has been said cloud computing is a distributed computing system that provides dynamically scaled computational resources, such as storage, processing power and applications as a service through the internet. Among the advantages are location independence, cost savings, management, and adaptability, on-demand self-service, scalability, multi-tenancy and resource pooling just a few [2], [3]. Cloud computing environments utilize virtual machines to process tasks and requests. A virtual machine (VM) is a simulated instance of a computer that can carry out the same tasks as the physical device. With virtualization, VMs are created. The process of creating a virtual computer with a dedicated central processing unit (CPU), memory, and storage from a physical machine known as the host machine is known as virtualization. The program running within the VM cannot communicate with the host computer's main operating system (OS) since it is partitioned off from the rest of the system [4]. The VMs are portable and easy to move between hypervisors in a process called migration. As a result, scalability is made simpler. VMs are the cloud computing units that perform and distribute resources as and when needed during task execution [5], [6]. The consumer and the cloud service provider (CSP) create a service level agreement

(SLA). The expected quality of service (QoS) necessary for a particular service is established using the performance criteria and dependability attributes [7], [8]. Cloud service providers can satisfy workload needs by allocating incoming jobs across several servers, networks, or other resources while maximizing efficiency and preventing service outages [9], [10]. The workload can be distributed among multiple geographical regions thanks to load balancing (LB). The scalability and adaptability of the cloud are used by LB to meet the demands of distributed jobs with numerous client relationships. Additionally, it boosts overall availability while reducing latency and increasing throughput [11].

Load balancing algorithms (LBA) can be categorized into two categories: static LBAs and dynamic LBAs. Methods of static-based balancing perform best in environments with a uniform or homogenous system or where there is little change in the load among nodes. Algorithms for dynamic load balancing can operate in both homogeneous and heterogeneous systems. They are therefore appropriate for cloud systems where the load variation will change over time [12] and LB in the cloud is a typical problem that makes it challenging to achieve SLAs [13]. Researchers have proposed several algorithms, both static and dynamic, in an attempt to improve LB amongst VMs. Yet, there is still the need to increase task execution speed. This research proposes a LBA by hybridizing a dynamic LBA and a machine learning algorithm to decrease algorithm runtime, makespan, and increase algorithm throughput. The goal is to address the speed of a LBA in task execution. This study's findings are summarized as follows: LB for independent jobs in the cloud has been examined and three goal functions are developed based on the following metrics: algorithm runtime, makespan, and throughput, which measure the number of tasks executed each millisecond. The proposed algorithm has been explored for LB in cloud environments, and the results are compared with those of existing optimization algorithms like ant colony optimization (ACO) and shortest job first (SJF). This research provides a novel approach to solving LB in cloud environments utilizing the proposed honey bee foraging Q-learning algorithm (HBFQL). The hybridization aims to complement the Q-learning algorithm using the honey bee algorithm. A simple illustration of cloud load balancing activities is provided in Figure 1.

Various algorithms, both static and dynamic, have been proposed to enhance load balancing among virtual machines. For instance, Manikandan and Pravin [14] introduced an improved weighted round-robin (WRR) algorithm, which effectively reduced response time but struggled with makespan due to a lack of consideration for VM availability status. Another attempt was made by Jeyalaksshmi *et al.* [15] who developed a hybrid LBA based on modified honey bee and round robin (RR) algorithms. While it reduced reaction time, it did not sufficiently address the challenge of achieving efficient throughput.
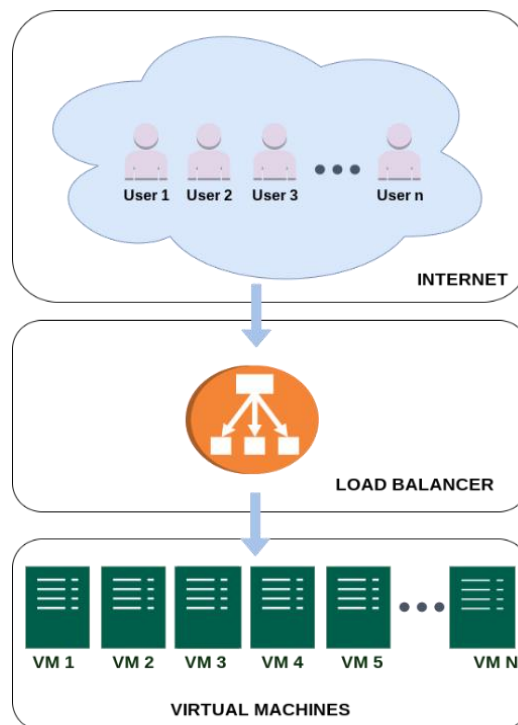

Figure 1. Load balancing overview

This study proposes a novel approach to enhance task execution speed by combining a dynamic algorithm with a reinforcement learning algorithm. Although there are multiple metrics, this paper addresses the issue of cloud load balancing by improving three critical metrics: algorithm runtime, makespan (the time it takes to complete all submitted tasks), and throughput (the number of tasks processed per millisecond). This hybrid solution tackles the issue of overloaded VMs in a cloud computing environment. By leveraging the strengths of both algorithms, resource allocations are optimized, ultimately improve the overall efficiency of cloud computing services.

2.   RELATED WORKS

Cloud resources should be available to users on demand because consumers want to use a provider's computing resources in a way that matches their changing demands and providers want to optimize the consumption of their resources and, as a result, their revenue. Users desire to have no downtime and less waiting. Effective resource management has become vital and the associated problem is caused mostly by VM allocation. With the aid of load balancing algorithms, efficient on-demand VM allocation on the cloud is accomplished with no downtime [16], [17]. After receiving requests from users, the load balancer employs load-balancing algorithms to divide the requests among the VMs. The load balancer selects the VM that will be given the next request. Task management is supervised by the data center controller. The load balancer receives jobs, which it then distributes to the appropriate VM using an LBA. The user requests are sent in a random order. It is necessary to delegate requests to VMs for processing. Task distribution is thus a major problem in cloud computing. If certain VMs are overwhelmed while others are idle or underutilized, QoS will be reduced. The declining QoS causes users to lose interest [18].

Researchers have proposed many techniques to tackle the problem of LB through static and dynamic methods. Static algorithms like RR and shortest job first (SJF) have been explored. Some dynamic algorithms such as ACO, honey bee foraging algorithm, and Q-learning are explored as well. The study in [19] presented a genetic algorithm (GA) which used the RR algorithm. The algorithm offers a LB method to boost the capacity of the data center. In order to solve the issues with RR, the approach distributed requests by scanning the hash map, which effectively includes all VMs. The tasks were assigned if a VM is available; otherwise, the best VM is chosen using GA to examine the best-fit workloads. Results shows that the method significantly reduces servers' response times.

Based on honey bee LBA and a modified RR load balancing approach, the authors proposed a hybrid LBA. The algorithm used the honey bee Influenced LB technique to transfer jobs from the overloaded to the underloaded virtual computers after identifying overloaded and underloaded VMs. It chooses available VMs for jobs with a higher priority based on each VM's current job. The SJF algorithm was modified by Alworafi et al. [20] to achieve work scheduling with the least amount of makespan. The enhanced approach considers both the time it takes for a single activity to finish and the total time it takes for all tasks to finish. Modified SJF is more efficient than SJF at improving responsiveness and makespan. A hybrid algorithm was proposed in [21] called SJF-ELM that combined the SJF buffering and the extreme learning machine (ELM) scheduling techniques. According to the characteristics of cloud computing, the research [22] implemented the ACO by the initiation and update of a probabilistic table called pheromone in place of routing tables. Such improvements greatly accelerated the search for candidate nodes for load balancing operations. The suggested approach was useful and effective. A study by Ragmani et al. [23] created a hybrid fuzzy-ACO method by using the fuzzy logic module to estimate the pheromone value to optimize the algorithm's parameters, improving the ant colony algorithm. To avoid an earlier convergence to less-than-ideal solutions, this algorithm makes use of an evaporation process from the experiment.

A method known as the enhanced honey bee inspired LB algorithm (EHILB) was developed by George et al. [24] and is based on the honey bee Inspired LB technique. EHBLB takes the workload from overcrowded VMs and distributes to the ones with less workloads-based tasks priority. Based on the total amount of tasks that are currently being given to each VM, it selects the next available VM. On the other hand, the Enhanced honey bee Inspired LB solution takes tasks away from VMs that are overworked and assigns them to less busy ones by taking into account the priority and resource needs of activities. The HB algorithm is enhanced in [25], [26]. Finding the appropriate jobs to assign takes less time thanks to the honey bee behavior LB (HBB-LB) technique. After determining the load on each VM, they are classified as underloaded, balanced, and overloaded; if the difference between the loads is more than zero, no VM movement between sectors is required. The LBA's capability to take into consideration each resource's capabilities and accessibility to service providers increases the effectiveness of the cloud system.

In study [26], an intelligent multi-agent reinforcement model (IMARM) for cloud resource distribution optimization was proposed. Since it approaches the issue of resource allocation from various perspectives, it offers a comprehensive solution for cloud service providers. The suggested architecture

achieves LB and fault tolerance using checkpointing and VM migration techniques. To evaluate the system response, sensor agents periodically report workload, energy consumption, and system failure status [27], [28]. Accordingly, Q-learning identifies the appropriate action in each system state. To address realistic methods to improve energy efficiency in cloud data centers, Ding *et al.* [29] developed task scheduling framework known as Q-learning based task scheduling framework for energy-efficient cloud computing. The centralized task dispatcher used at the cloud level pushed each task that a user submits to the appropriate server by allocating the work to that server's request queue. During the task scheduling phase, the Q-learning based scheduler on each server used a time frame to determine when to process the tasks in the request queue. The scheduler assigned tasks to virtual machines based on a continuously changing policy, rewarding assignments that can reduce task response time and increase CPU utilization on each server.

3.   METHOD

Cloud service providers will increasingly be required to guarantee that the SLA contract is upheld and that a high-level QoS is provided. A cloud environment is made up of VMs. Let the set of n number of VMs in a cloud network be represented by $V = \{V_1, V_2, \dots V_n\}$. Each VM has its own set of resources, such as CPU cores, memory, and storage. There are S number of servers, and one serves as the central server, posting requests originating from the VMs [30]–[33]. The number of independent tasks being run by n number of VMs is represented by m. To balance the load, a scheduler is needed to map m tasks to n VMs. The equations used are derived from study [34]–[36]. A single VM's memory load is determined through:

$$LV_n = (Mv_n + \frac{Vpm_n}{Vm_n}) \times 100\% \tag{1}$$

where LV_n is memory load on the VM V_n, Mv_n is memory required to complete the task, Vpm_n is percentage of memory available in V_n, and Vm_n is percentage of total memory in V_n. The amount of CPU load that the task in the VM V_n is calculated as (2).

$$LC_n = (Cv_n + \frac{Vpc_n}{Vc_n}) \times 100\% \tag{2}$$

where LC_n is CPU load on the VM V_n, Cv_n is CPU required to complete the task, Vpc_n is percentage of CPU available in V_n, Vc_n is percentage of total CPU in V_n. The total load in terms of memory load and CPU load on a VM V_n is given as (3).

$$L_n = LC_n + LV_n \tag{3}$$

The total load on a host j machine containing n number of VMs is given as (4).

$$LH_j = \sum_{n=0}^{m} L_{jn} \tag{4}$$

A single VM V_n processing time PT is shown as (5).

$$PT_n = \frac{L_n}{C_n} \tag{5}$$

where L_n is total load on V_n, C_n is capacity of V_n

The capacity C of a single VM V_n is calculated as (6).

$$C_n = P_{nn} \times P_{mipsn} + VM_{bwn} \tag{6}$$

where P_{nn} is number of processors in VM V_n, P_{mipsn} is million instructions per second of all processors in VM V_n and VM_{bwn} is communication bandwidth ability of VM V_n. The capacity of all VMs in the data center is calculated as (7).

$$C = \sum_{i=o}^{n} C_i \tag{7}$$

The processing time of all the VMs is shown in (8).

$$PT = \frac{L}{C} \tag{8}$$

Hybridization of the Q-learning and honey bee foraging algorithms for … (Adeyinka Ajao Adewale)

This research presents a novel dynamic load balancing algorithm by utilizing the Q-learning ML algorithm and the honey bee foraging algorithm to form a hybrid new algorithm called the honey bee foraging Q-learning algorithm (HBFQL). The hybridization aims to complement the Q-learning algorithm using the honey bee algorithm. The Q-learning algorithm does a lot of computation to get the Q-value for every possible action in each state and requires a lot of storage capacity to store the values. To mitigate this, the honey bee classifies nodes into overloaded, balanced, and underloaded categories. The Q agent then computes the Q values for VMs in each category, making it easier and faster to migrate tasks between VMs. The honey bee foraging algorithm is dynamic and adaptable, enabling it to work in all types of cloud computing environments, which helps offset the inability of the Q-learning algorithm to operate in environments with discreet and finite state and action spaces. The honey bee algorithm has the drawback that VM status change after categorization is not quickly considered. The Q-learning helps offset that by keeping a constantly updated table of Q-values that monitor the state of each VM within the categories meaning it can detect changes faster. This hybrid algorithm aims to improve performance and speed by improving throughput while minimizing makespan and runtime.

The hybrid loads balancing algorithm developed in this study builds on the Q-learning ML algorithm and the honey bee foraging algorithm. It uses the honey bee load balancing algorithm to keep track of the virtual machine's status. The VMs are grouped into groups with a status of under-utilized, balanced, or over-utilized. The Q-learning algorithm determines the Q-value of the VMs within the different categories. It uses the Q-value to evaluate which VMs are running at optimal capacity and which are available for task migration. At the beginning of the session, all VMs in the data center are indexed, and all VM attributes are set to default. The VMs are initialized just the way worker bees are initialized in HBF. Once tasks arrive at the data center, the broker forwards the tasks to the assigned VMs until they are tagged unavailable to new jobs. The task scheduling algorithm decides which VM will handle a task. The CPU utilization of the VM is used as the criteria for over-utilized and under-utilized. This forms the state of the VMs. This utilization history is logged continuously as long as the algorithm keeps running. A VM is selected from a set of VMs denoted as $VM = \{VM_1, VM_2, ... VM_3\}$. At the beginning of the training, the states and actions associated with each condition are randomly chosen until specific criteria are met (*Epsilon=0/Q-value=0*). This process is called exploitation and is done because there is no history yet. After exploitation is exploration.

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(PT_i - PT)^2} \tag{9}$$

where σ is standard deviation of load used to compare threshold value, PT_i is processing time of a single VM and PT is processing time of all VMs. The standard deviation (σ) is compared to the threshold value set and used to categorize VMs into overloaded, balanced, and underloaded categories. The Q-value lets the agent (DC broker) know if a particular state action provides benefits to prioritize those actions. That is where the learning comes from. The idea is to evaluate if using a specific VM will be profitable in terms of execution time for the task. This is done by constantly updating the Q-value of the VMs, thereby training the algorithm. The utilization history records the previous state for the next iteration. What changes every iteration is the reward/penalty for each VM based on the associated Q-value. The virtual machine with the best reward, as computed by the Q-learning algorithm, is selected as the following path (VM) to send a task (Cloudlet) when the data center is initialized, the Q matrix is also initialized. The Q-matrix also known as the Q-table, has the form [state, action] with all values initialized to zero (0). The agent's memory is the Q-table. It maps out all the reward matrix's values on a new table, demonstrating how to get the best payout based on prior states. The Q-matrix has an $8 \times n$ dimension given as (10).

$$Q = \{[a_1], [a_2], [a_3], ... [a_n]\} \tag{10}$$

where $[a_1]$ is column vector of VM utilization, n is number of VMs and the Q-table formula is given as (11).

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma maxQ(s',a') - Q(s,a)) \tag{11}$$

where s is current state, s' is next state, a is current action, a' is next action, α is learning rate, r is penalty or reward, $\gamma(0 < \gamma < 1)$ is discount factor and effect on successive state, $Q(s,a)$ is Q-value for current state/action and $Q(s',a')$ is Q-value for next state/action. Each VM column vector has a size of eight. The column vector stores the utilization history of the virtual machine. Once the Q-matrix has been initiated, the Q-values are computed. The Q-learning algorithm computes the Q-value with a reward and penalty basis. In this study, a VM with the highest CPU utilization is considered a penalty, while a VM with low CPU

utilization is regarded as a reward. Once the Q-values are computed and the Q-matrix is updated, the maximum correlation of values in the Q-matrix is resolved to determine the next VM for the waiting task. The reward matrix, also known as the reward table, mirrors the Q-matrix except that it displays the reward or penalty associated with each action. The steps in the proposed HBFQL algorithm are as follows:

Algorithm. honey bee foraging Q-learning algorithm (HBFQL)
Step 1: Start
Step 2: Initialize VM population with CPU utilization history
Step 3: Create a Q matrix for VMs set to zero
Step 4: Build VM reward matrix using VM availability
Step 4.1: Utilized VM has a negative reward, under-utilized VM has a positive reward
Step 5: Set threshold value
Step 6: While tasks exist, calculate σ using (9)
Step 7: Check current VM load > σ
Step 8: Group VMs into overloaded, balanced, and underloaded using threshold value
Step 9: Get VMs CPU utilization history
Step 10: Update reward matrix
Step 11: Choose VM Q-value from Q-matrix
Step 12: Measure q-value for each VM using (11)
Step 13: Compute updated Q-matrix
Step 14: Use maximum correlation to select best VM
Step 15: Match task to VM
Step 16: If all tasks are complete, match task to VM and go to step 17, else go to step 6
Step 17: Stop

Figures 2, 3, and 4 illustrate various aspects of the HBFQL algorithm. Figure 2 is a block diagram visually representing the algorithm's process. Figure 3 presents the architectural model of the algorithm, while Figure 4 depicts the algorithm's flowchart.
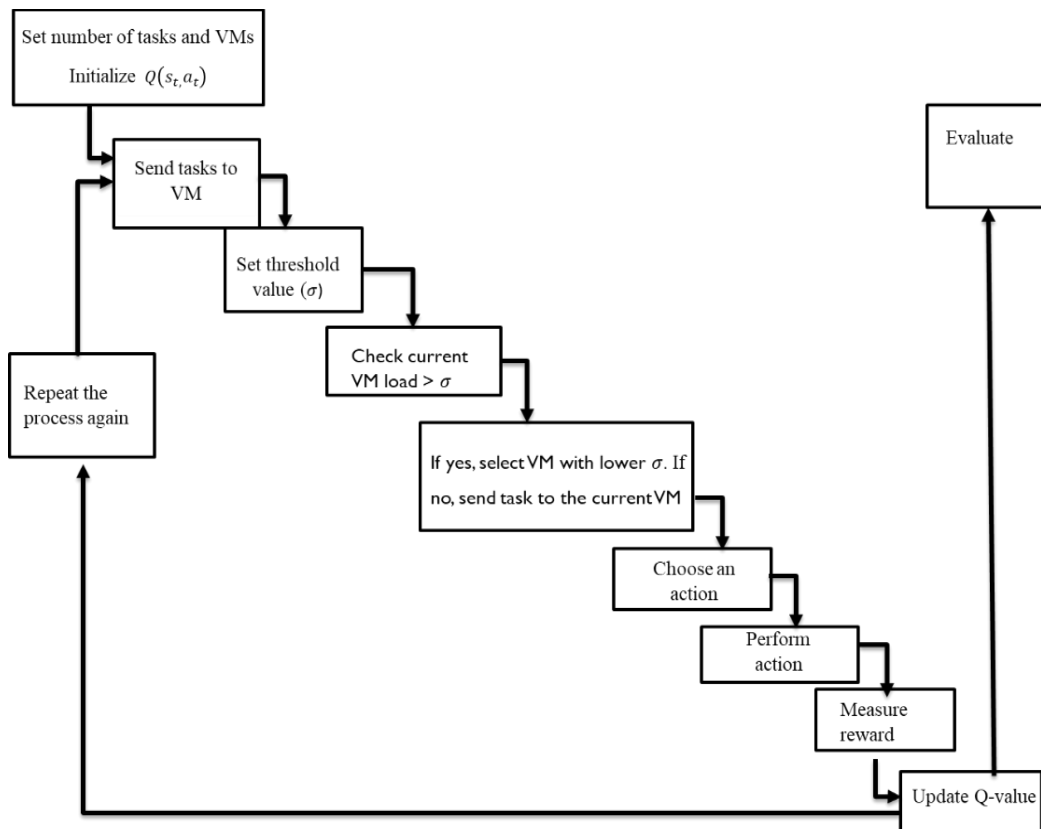
Figure 2. HBFQL algorithm block diagram
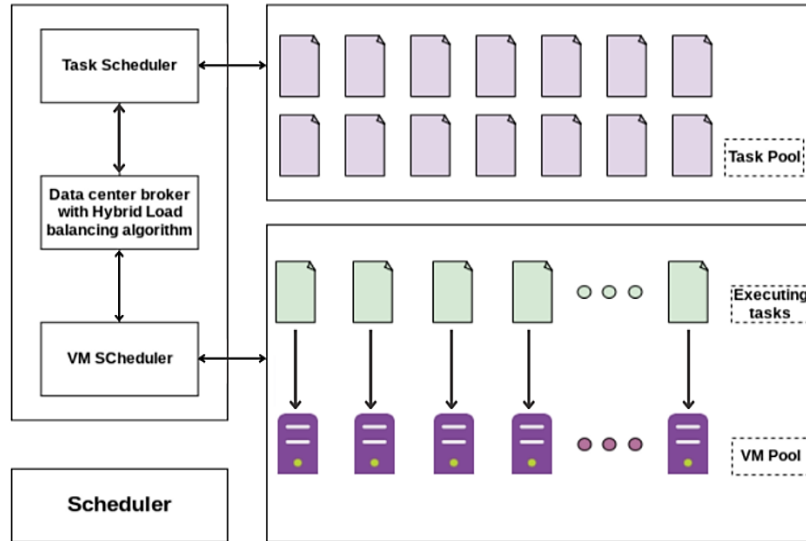
Figure 3. Architectural model for HBFQL algorithm
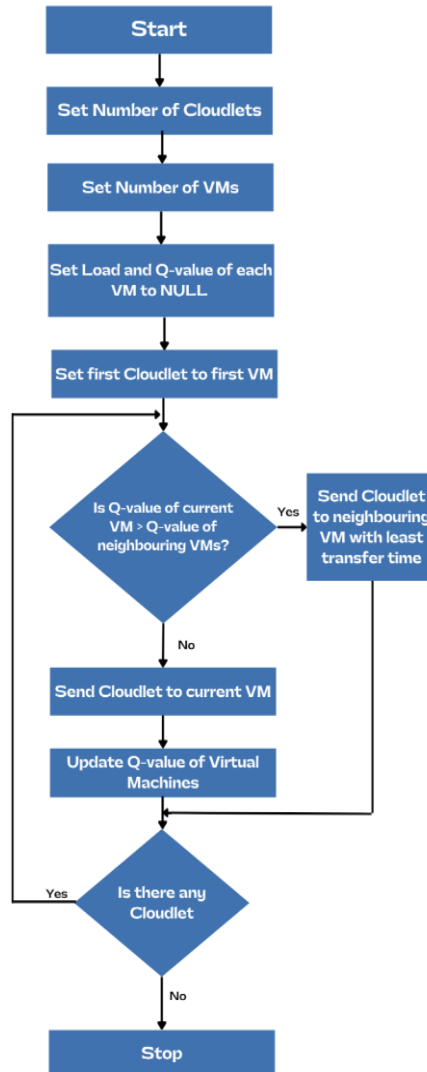
Figure 4. HBFQL algorithm flowchart

4.   RESULTS AND DISCUSSION

The HBFQL algorithm was simulated and its performance was analyzed. The simulation was done using the CloudSim simulator. The simulator was run on a machine with an Intel core i7 processor, 16 GB RAM, 2.8 GHz CPU, and Windows 11 OS. The cloud environment simulated is presented in Table 1. The algorithm was evaluated under three different scenarios presented in Table 2. In the first scenario, the number of VMs was fixed at 100 while the tasks were increased from 100 to 2,000 at intervals of 100. The second scenario has a constant number of tasks fixed at 100 while the VMs were increased at intervals of 10 from 10 to 100. The third scenario has both varying tasks and varying VMs. The VMs and tasks were steadily increased from 10 to 100 at intervals of 10. The algorithm was evaluated in terms of throughput, makespan, and runtime. These three metrics measured task execution time and would indicate the speed of algorithm execution. Figures 5, 6, and 7 illustrate the throughput, makespan, and runtime respectively. The performance of the proposed HBFQL algorithm was measured against the ACO algorithm as well as the SJF algorithm.

Table 1. The simulated cloud environment

Type	Number	Specification	Value
Datacenter	1	Architecture	x86
		Operating system	Linux
		Cost	3.0
		Memory	2048 MB
		Storage	10000000 MB
		Bandwidth	10000 MB
		Cost per memory	0.05
VM	10 to 100	Name	Xen
		MIPS	1000
		Storage	150 GB
		Memory	16 GB
		Bandwidth	10 GB/s
		Cores (Pes)	1
		ID	0
Cloudlet	100 to 2000	ID	0
		Length	400,000
		File size	300
		Output size	300

Table 2. Algorithm test scenario

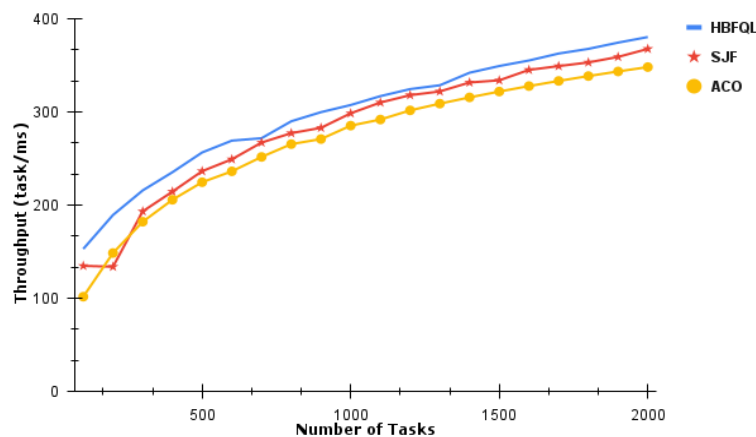| S/N	Name	Type
1	Experiment 1	Constant VMs (100) Varying tasks (100 to 2000, increments of 100)
2	Experiment 2	Varying VMs (10 to 100, increments of 10) Constant tasks (100)
3	Experiment 3	Varying VMs (10 to 100, increments of 10) Varying tasks (10 to 100, increments of 10)

Figure 5. Throughput results for experiment 1

Figure 5 shows the throughput performance of all three algorithms side by side. From the initial stage to the end of the cycle, the HBFQL performs better. This is because the HBFQL can execute more tasks per second than the other algorithms. The HBFQL has a 10.86% increase in throughput compared with ACO

and a 5.5% increase compared to SJF. The makespan comparison is displayed in Figure 6. Although the algorithms begin with similar values, the HBFQL gradually performs better as the number of tasks increases. This is because the HBFQL algorithm employed monitoring VMs with the q-value and categorizing the VMs based on their current load size highly decreasing the chances of having an overloaded VM. The HBFQL has a 26.52% decrease in makespan compared to ACO and a 9.58% decrease compared to SJF. Figure 7 shows the runtime comparison between HBFQL, SJF, and ACO. The algorithms start with identical values. As the number of tasks increased, the ACO and the HBFQL algorithms seemed to tie in values until the number of tasks increased to 1800, and the HBFQL had a lower runtime. The HBFQL has a 24.66% decrease in runtime compared with SJF and a 0.85% difference with ACO.
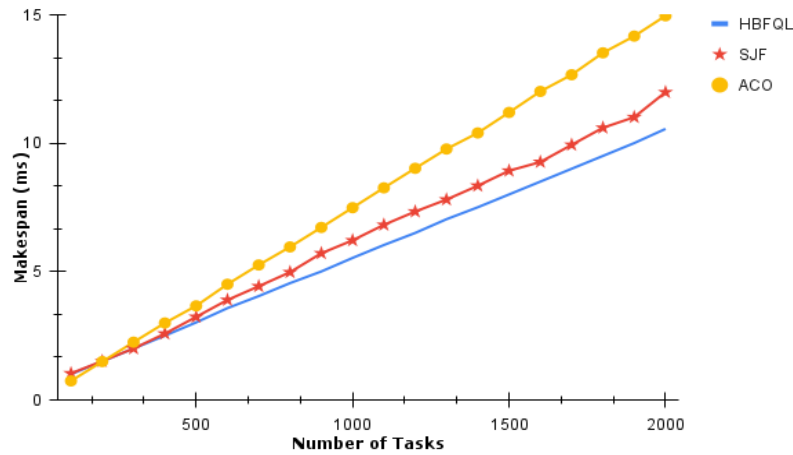
Figure 6. Makespan results for experiment 1
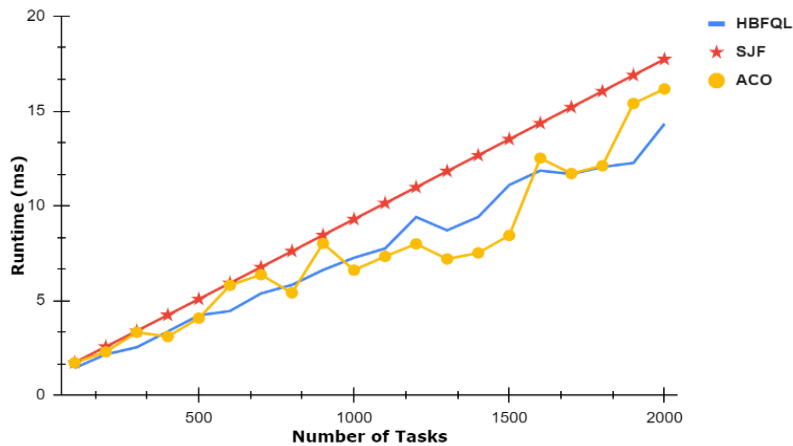
Figure 7. Runtime results for experiment 1

The results from experiment 2 with constant tasks and varying VMs are displayed in Figures 8, 9, and 10. The throughput performance of all three algorithms is compared in Figure 8. The HBFQL has a 3.54% decrease in throughput when compared with ACO and a 13.59% increase when compared with SJF. Figure 9 displays the makespan comparison. Although the HBFQL initially reached a plateau, it gradually improved as the number of VMs rose. This dramatically reduces the likelihood of a VM becoming overwhelmed. The HBFQL improves speed in execution by reducing the time it takes to complete tasks in milliseconds. The HBFQL has a 4.21% increase in makespan compared to ACO and a 4.75% decrease compared to SJF. ACO, SJF, and HBFQL runtime comparison is shown in Figure 10. From beginning to end, the HBFQL algorithm performs significantly better. The HBFQL has a 7.79% increase in runtime when compared with ACO and a 19.81% decrease when compared with SJF.
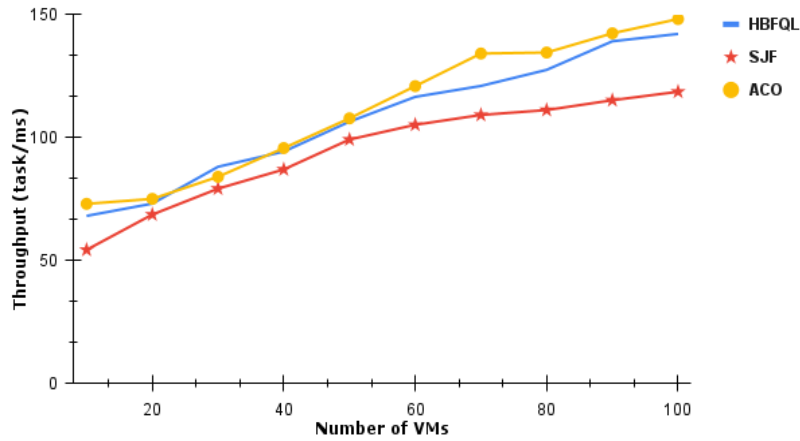
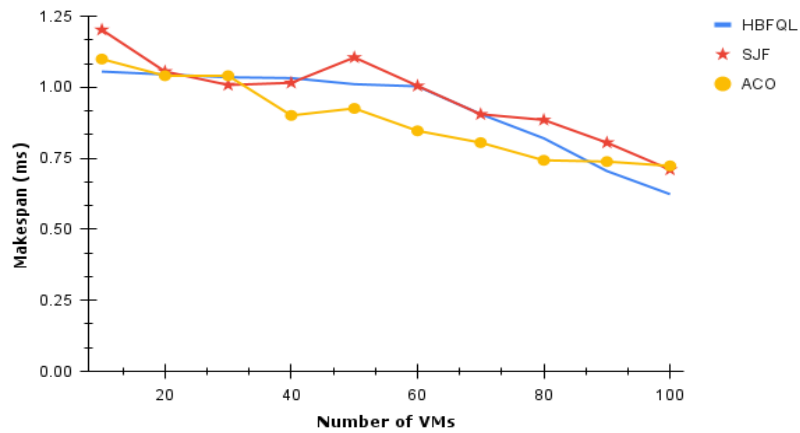Figure 8. Throughput results for experiment 2

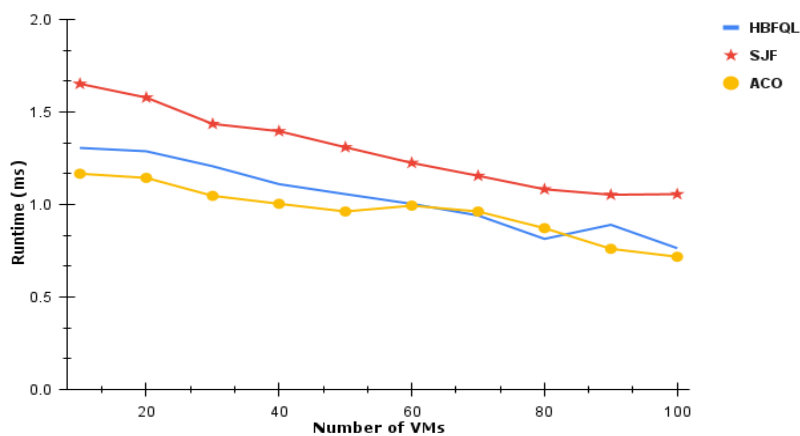
Figure 9. Makespan results for experiment 2

Figure 10. Runtime results for experiment 2

Experiment 3 results are presented in Figures 11, 12, and 13 charts. This time both the VMs and the tasks were varied. In Figure 11, the throughput performance of the three algorithms is contrasted. The HBFQL starts on average and then steadily improves when the tasks and VMs are 60 until it outperforms the other algorithms in terms of throughput. The HBFQL has a 2.97% difference in throughput when compared

with ACO and a 6.02% increase when compared with SJF. The Makespan comparison is shown in Figure 12. During the testing, the HBFQL algorithm performed better. The HBFQL has a 19.81% difference in makespan when compared with ACO and a 12.53% decrease when compared with SJF. Figure 13 compares the runtimes for ACO, SJF, and HBFQL. The algorithms start with similar values until the jobs and VMs are 40, then the HBFQL algorithm performs better than average. From this point on, the HBFQL algorithm runs faster than the ACO and much faster than the SJF. The HBFQL has an 11.25% difference in runtime compared with ACO and an 18.63% decrease compared to SJF.

It is noteworthy to mention that the performance of the proposed hybrid LBA is evident. The results show that the proposed algorithm performs better than some known LBA like the SJF and ACO. It can also be inferred from the result that the proposed algorithm performs better under certain conditions than others; hence the algorithm has its best use case, just like other traditional algorithms. In experiment one, the proposed algorithm in this study performed better overall than the other simulated algorithms. As expected, the runtime of the algorithms in experiment one increase as the number of tasks increase. Although the algorithms follow closely in runtime, there is an evident performance gulf between the HBFQL algorithm and other simulated algorithms as the workload in the data center increases. Experiment two shows the difference between the HBFQL and other simulated algorithms, although the evidence is relatively marginal.

According to the analysis in experiment 3, the proposed HBFQL method performed better than the compared algorithms. This investigation concludes that HBFQL has improved speed and efficiency compared to its rival algorithms. As a result, it efficiently and successfully balanced the load in the cloud network simulation. The analysis demonstrates that when it comes to task load balancing on the cloud network, the HBFQL performed better than the ACO and SJF. The summary of the improvements of the proposed HBFQL algorithm over the SJF and ACO is presented in Table 3.
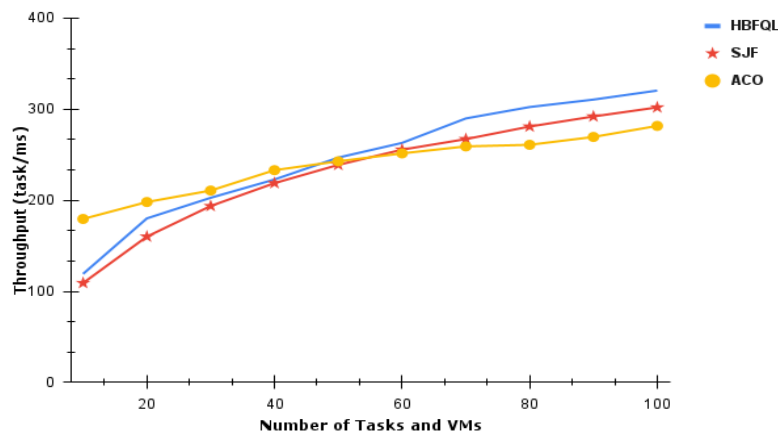

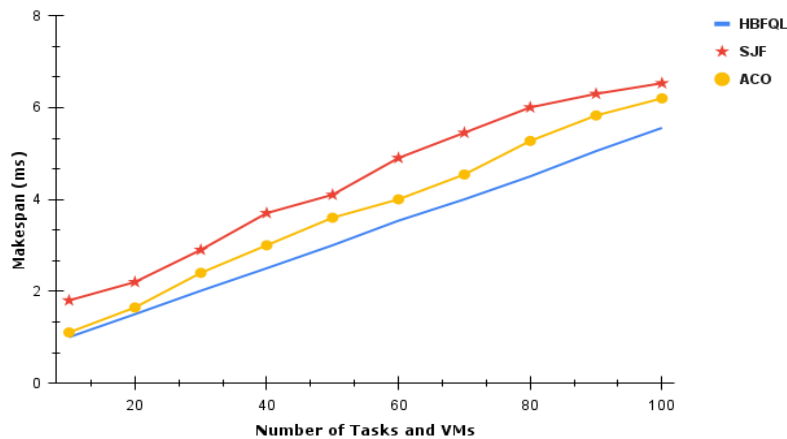Figure 11. Throughput results for experiment 3
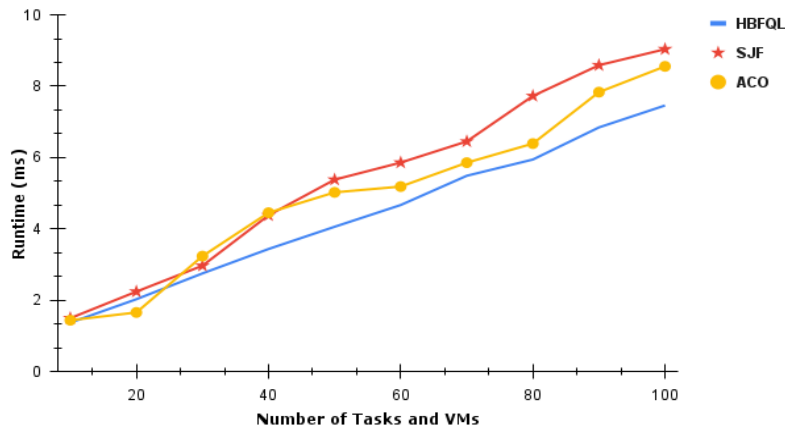

Figure 12. Makespan results for experiment 3

Figure 13. Runtime result for experiment 3

Table 3. Percentage improvement of HBFQL compared with SJF and ACO

Experiment	Metric	SJF (%)	ACO (%)
1	th1	5.5	10.86
	mk2	9.58	26.52
	ru3	0.85	24.66
2	th2	13.59	-3.54
	mk2	4.75	-4.21
	ru2	19.81	7.79
3	th3	6.02	2.97
	mk3	12.53	19.81
	ru3	18.63	11.25
Avg	TH	8.37	3.43
	MK	8.95	13.71
	RT	13.1	14.57

5.   CONCLUSION

The hybrid dynamic LBA that utilized the honey bee foraging LBA and the Q-learning algorithm as part of its objectives has been proposed in is study. The resulting algorithm was tested in a simulated cloud environment, and its performance was evaluated in comparison with other LBAs. The approach adopted selects the node to forward requests to by categorizing VMs based on the current load status and then calculating the Q-value of each node based on its utilization history. The honey bee Foraging algorithm groups the VMs into under-loaded, balanced, and overloaded. The Q-learning used a Q-table of Q-values to determine the best node to distribute tasks. The Q-table was continually updated based on the utilization history of each VM. The different experiments' results showed that the proposed hybrid dynamic algorithm can work in multiple kinds of environments. It can also be inferred that this proposed algorithm enables faster task distribution during LB activities. The different algorithms were tested under conditions such as a constant number of VMs and varying tasks, a constant number of tasks with varying VMs, and, a varying number of tasks and VMs. The HBFQL algorithm outperformed the SJF and ACO algorithms in scenarios where tasks varied and performed averagely when the number of tasks was constant. Nodes' status change after categorization was not quickly considered.

ACKNOWLEDGEMENTS

REFERENCES
[1]   F. Ye, S. Wu, Q. Huang, and X. A. Wang, "A novel QoS-aware load balancing mechanism in cloud environment," in *2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, Sep. 2016, pp. 298–301, doi: 10.1109/INCoS.2016.97.
[2]   I. Arpaci, "Antecedents and consequences of cloud computing adoption in education to achieve knowledge management," *Computers in Human Behavior*, vol. 70, pp. 382–390, May 2017, doi: 10.1016/j.chb.2017.01.024.
[3]   D. Puthal, B. P. S. Sahoo, S. Mishra, and S. Swain, "Cloud computing features, issues, and challenges: a big picture," in *2015*

International Conference on Computational Intelligence and Networks, Jan. 2015, pp. 116–123, doi: 10.1109/CINE.2015.31.

[4]   L. Helali and M. N. Omri, "A survey of data center consolidation in cloud computing systems," *Computer Science Review*, vol. 39, Feb. 2021, doi: 10.1016/j.cosrev.2021.100366.

[5]   M. Masdari, S. S. Nabavi, and V. Ahmadi, "An overview of virtual machine placement schemes in cloud computing," *Journal of Network and Computer Applications*, vol. 66, pp. 106–127, May 2016, doi: 10.1016/j.jnca.2016.01.011.

[6]   A. U. Adoghe, E. C. Owuama, V. Oguntosin, and B. Morawo, "Design and implementation of a low-cost cloud-powered home automation system," *Journal of Engineering Science and Technology Review*, vol. 15, pp. 177–192, 2022.

[7]   J. K. Meena and R. Kumar Banyal, "Efficient virtualization in cloud computing," in *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, Apr. 2021, pp. 227–232, doi: 10.1109/ICCMC51019.2021.9418425.

[8]   C. Sauvanaud, M. Kaâniche, K. Kanoun, K. Lazri, and G. Da Silva Silvestre, "Anomaly detection and diagnosis for cloud services: practical experiments and lessons learned," *Journal of Systems and Software*, vol. 139, pp. 84–106, May 2018, doi: 10.1016/j.jss.2018.01.039.

[9]   S. K. Mishra, B. Sahoo, and P. P. Parida, "Load balancing in cloud computing: A big picture," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 2, pp. 149–158, Feb. 2020, doi: 10.1016/j.jksuci.2018.01.003.

[10]   D. A. Shafiq, N. Z. Jhanjhi, and A. Abdullah, "Load balancing techniques in cloud computing environment: A review," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 3910–3933, 2022, doi: 10.1016/j.jksuci.2021.02.007.

[11]   S. K. Panda and P. K. Jana, "Load balanced task scheduling for cloud computing: a probabilistic approach," *Knowledge and Information Systems*, vol. 61, no. 3, pp. 1607–1631, Dec. 2019, doi: 10.1007/s10115-019-01327-4.

[12]   N. Shah and M. Farik, "Static load balancing algorithms in cloud computing: challenges & solutions," *International Journal of Scientific & Technology Research*, vol. 4, no. 10, 2015.

[13]   K. Wakunuma and R. Masika, "Cloud computing, capabilities and intercultural ethics: implications for Africa," *Telecommunications Policy*, vol. 41, no. 7–8, pp. 695–707, Aug. 2017, doi: 10.1016/j.telpol.2017.07.006.

[14]   N. Manikandan and A. Pravin, "An efficient improved weighted Round Robin load balancing algorithm in cloud computing," *International Journal of Engineering & Technology*, vol. 7, no. 3.1, Aug. 2018, doi: 10.14419/ijet.v7i3.1.16810.

[15]   S. Jeyalaksshmi, J. Anita Smiles, D. Akila, D. Mukherjee, and A. J. Obaid, "Energy-efficient load balancing technique to optimize average response time and data center processing time in cloud computing environment," *Journal of Physics: Conference Series*, vol. 1963, no. 1, Jul. 2021, doi: 10.1088/1742-6596/1963/1/012145.

[16]   I. Pietri and R. Sakellariou, "Mapping virtual machines onto physical machines in cloud computing: a survey," *ACM Computing Surveys*, vol. 49, no. 3, pp. 1–30, Sep. 2017, doi: 10.1145/2983575.

[17]   B. A. Al Amal Murayki Alruwaili, M. Humayun, and N. Z. Jhanjhi, "Proposing a load balancing algorithm for cloud computing applications," *Journal of Physics: Conference Series*, vol. 1979, no. 1, Aug. 2021, doi: 10.1088/1742-6596/1979/1/012034.

[18]   E. Jafarnejad Ghomi, A. Masoud Rahmani, and N. Nasih Qader, "Load-balancing algorithms in cloud computing: a survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, Jun. 2017, doi: 10.1016/j.jnca.2017.04.007.

[19]   K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, "A genetic algorithm (GA) based load balancing strategy for cloud computing," *Procedia Technology*, vol. 10, pp. 340–347, 2013, doi: 10.1016/j.protcy.2013.12.369.

[20]   M. A. Alworafi, A. Dhari, A. A. Al-Hashmi, A. B. Darem, and Suresha, "An improved SJF scheduling algorithm in cloud computing environment," in *2016 International Conference on Electrical, Electronics, Communication, Computer and Optimization Techniques (ICEECCOT)*, Dec. 2016, pp. 208–212, doi: 10.1109/ICEECCOT.2016.7955216.

[21]   S. Rekha and C. Kalaiselvi, "Load balancing using SJF-MMBF and SJF-ELM approach," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 74–86, Jan. 2021, doi: 10.32628/CSEIT21714.

[22]   P. Verma, S. Shrivastava, and R. K. Pateriya, "Enhancing load balancing in cloud computing by ant colony optimization method," *International Journal of Computer Engineering in Research Trends*, vol. 4, no. 6, 2017.

[23]   A. Ragmani, A. Elomri, N. Abghour, K. Moussaid, and M. Rida, "An improved hybrid fuzzy-ant colony algorithm applied to load balancing in cloud computing environment," *Procedia Computer Science*, vol. 151, pp. 519–526, 2019, doi: 10.1016/j.procs.2019.04.070.

[24]   M. S. George, K. C. N. Das, and B. R. Pushpa, "Enhanced honeybee inspired load balancing algorithm for cloud environment," in *2017 International Conference on Communication and Signal Processing (ICCSP)*, Apr. 2017, pp. 1649–1653, doi: 10.1109/ICCSP.2017.8286670.

[25]   P. Ehsanimoghadam and M. Effatparvar, "Load balancing based on bee colony algorithm with partitioning of public clouds," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 4, 2018, doi: 10.14569/IJACSA.2018.090462.

[26]   A. Belgacem, S. Mahmoudi, and M. Kihl, "Intelligent multi-agent reinforcement learning model for resources allocation in cloud computing," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2391–2404, Jun. 2022, doi: 10.1016/j.jksuci.2022.03.016.

[27]   M. A. Shahid, M. M. Alam, and M. M. Su'ud, "Performance evaluation of load-balancing algorithms with different service broker policies for cloud computing," *Applied Sciences*, vol. 13, no. 3, Jan. 2023, doi: 10.3390/app13031586.

[28]   M. A. Shahid, N. Islam, M. M. Alam, M. M. Su'ud, and S. Musa, "A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach," *IEEE Access*, vol. 8, pp. 130500–130526, 2020, doi: 10.1109/ACCESS.2020.3009184.

[29]   D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, and J. Zeng, "Q-learning based dynamic task scheduling for energy-efficient cloud computing," *Future Generation Computer Systems*, vol. 108, pp. 361–371, Jul. 2020, doi: 10.1016/j.future.2020.02.018.

[30]   U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2332–2342, Jun. 2022, doi: 10.1016/j.jksuci.2020.01.012.

[31]   O. Jonathan, S. Misra, and V. Osamor, "Comparative analysis of machine learning techniques for network traffic classification," *IOP Conference Series: Earth and Environmental Science*, vol. 655, no. 1, Feb. 2021, doi: 10.1088/1755-1315/655/1/012025.

[32]   V. Oguntosin, A. Akindele, and A. Uyi, "A convolutional neural network for soft robot images classification," in *2020 7th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, Nov. 2020, pp. 110–114, doi: 10.1109/ISCMI51676.2020.9311562.

[33]   L. A. Demidova and A. V Gorchakov, "Research and study of the hybrid algorithms based on the collective behavior of fish schools and classical optimization methods," *Algorithms*, vol. 13, no. 4, Apr. 2020, doi: 10.3390/a13040085.

[34]   V. L. Narasimhan, V. S. Jithin, M. Ananya, and J. Oluranti, "AI-based enhanced time cost-effective cloud workflow scheduling," 2022, pp. 277–297.

[35]   D. B. L.D. and P. Venkata Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292–2303, May 2013, doi: 10.1016/j.asoc.2013.01.025.

[36] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep Q-learning,"
      Information Sciences, vol. 512, pp. 1170–1191, Feb. 2020, doi: 10.1016/j.ins.2019.10.035.

BIOGRAPHIES OF AUTHORS

Adeyinka Ajao Adewale 🆔 ⓖ SC ⓒ is an associate professor of information and communication engineering. He has a B.Sc. degree in electrical engineering, an M.Sc. in information technology, and a Ph.D. degree in information communication engineering. He is a COREN registered engineer and a corporate member of the Nigerian Society of Engineers (NSE) and has several professional qualifications to his credit. His research areas of interest include quality of service (QoS) in mobile or wireless communication, artificial intelligence applications in mobile communication, digital signal processing in seismic operations, network planning and optimization, database design and implementation, software design, and programming. He is a Huawei Certified instructor in routing and switching, Microsoft and Oracle Certified instructor in database design and SQL. He can be contacted at email: ade.adewale@covenantuniversity.edu.ng.

Oghorchukwuyem Obiazi 🆔 ⓖ SC ⓒ is a software engineer with a B.Eng. in computer engineering from the University of Benin, and an M.Eng. in computer engineering from Covenant University. Her research interests include energy-efficient cloud computing, Software engineering, and the application of data analysis in software startup scaling. She can contacted at email: oghorchukwuyem.obiazipgs@stu.cu.edu.ng.

Kennedy Okokpujie 🆔 ⓖ SC ⓒ holds a bachelor of engineering (B.Eng.) in electrical and electronics engineering, master of science (M.Sc.) in electrical and electronics engineering, master of engineering (M.Eng.) in electronics and telecommunication engineering, and master of business administration (MBA), Ph.D. in information and communication engineering, besides several professional certificates and skills. He is currently a senior lecturer in the Department of Electrical and Information Engineering, Covenant University, Ota, Ogun State, Nigeria. He is a member of the Nigerian Society of Engineers and the Institute of Electrical and Electronics Engineers (IEEE). His research areas of interest include biometrics, artificial intelligence, digital signal processing, and network security and management. He can be contacted at email: kennedy.okokpujie@covenantuniversity.edu.ng.

Omiloli Koto 🆔 ⓖ SC ⓒ obtained his bachelor's degree in electrical and electronic engineering from Niger Delta University and a master's degree with a major in electrical engineering from Covenant University. He is currently the operation and maintenance engineer at Ikeja Electric Plc. His research interest is in nonlinear control and observer design. He can be contacted by email: andrew.omiloli@stu.cu.edu.ng.