IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 4, 2021, accepted February 4, 2022, date of publication February 9, 2022, date of current version February 17, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3149955

Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning

BOONHATAI KRUEKAEW AND WARANGKHANA KIMPAN, (Member, IEEE)
Department of Computer Science, School of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
Corresponding author: Warangkhana Kimpan (warangkhana.ki@kmitl.ac.th)

This work was supported by the School of Science, King Mongkut's Institute Technology Ladkrabang.

ABSTRACT Workload balancing in cloud computing is still challenging problem, especially in Infrastructure as a Service (IaaS) in the cloud model. A problem that should not occur during cloud access is a host or server being overloaded or underloaded, which may affect the processing time or may result in a system crash. Therefore, to prevent these problems, an appropriate schedule of access should be considered so that the system can distribute tasks across all available resources, which is called load balancing. The load balancing technique should ensure that all Virtual Machines (VMs) are used appropriately. In this paper, an independent task scheduling approach in cloud computing is proposed using a Multi-objective task scheduling optimization based on the Artificial Bee Colony Algorithm (ABC) with a Q-learning algorithm,which is a reinforcement learning technique that helps the ABC algorithm work faster, called the MOABCQ method. The proposed method aims to optimize scheduling and resource utilization, maximize VM throughput, and create load balancing between VMs based on makespan, cost, and resource utilization, which are limitations of concurrent considerations. Performance analysis of the proposed method was compared using CloudSim with the existing load balancing and scheduling algorithms: Max-Min, FCFS, HABC_LJF, Q-learning, MOPSO, and MOCS algorithms in three datasets: Random, Google Cloud Jobs (GoCJ), and Synthetic workload. The experimental results indicated that the algorithms used MOABCQ approach outperformed the other algorithms in terms of reducing makespan, reducing cost, reducing degree of imbalance, increasing throughput and average resource utilization.

INDEX TERMS Cloud computing, hybrid artificial bee colony algorithm, multi-objective task, Q-learning, task scheduling.

I. INTRODUCTION

Recently, cloud computing has played an important role in many organizations. Cloud computing started out to provide of users requirements for accessing resource computing or to enable users to purchase cloud services as required within the concept of on-demand resource sharing through highly internet-based applications. Cloud computing can also process many different services depending on the service and working platforms that are needed by the users [1]. Cloud computing is a combination of distributed and parallel computing with the sharing of resources such as software and hardware that will be utilized as ''pay-as-you-go'' [2]. To use it, users do not need to purchase any platform or software; they only have an internet connection to access and pay for their usage.

Infrastructure as a Service (IaaS) is one of the technology models behind the management of servers, data centers, and Virtual Machines (VMs). IaaS is a cloud service that provides a cloud computer or server for processing and storing data in the cloud. Users can run any operating system or application on a rental server free of maintenance and operating costs of hardware. Since the cloud infrastructure is run on VMs, IaaS has another advantage, including providing the access of the user to the servers in nearby locations. This service depends on the needs of the user, called Quality of Service (QoS) and Service Level Agreements (SLAs). The payment for the services depends on the agreement between the user and

The associate editor coordinating the review of this manuscript and approving it for publication was Gerard P. Parr.

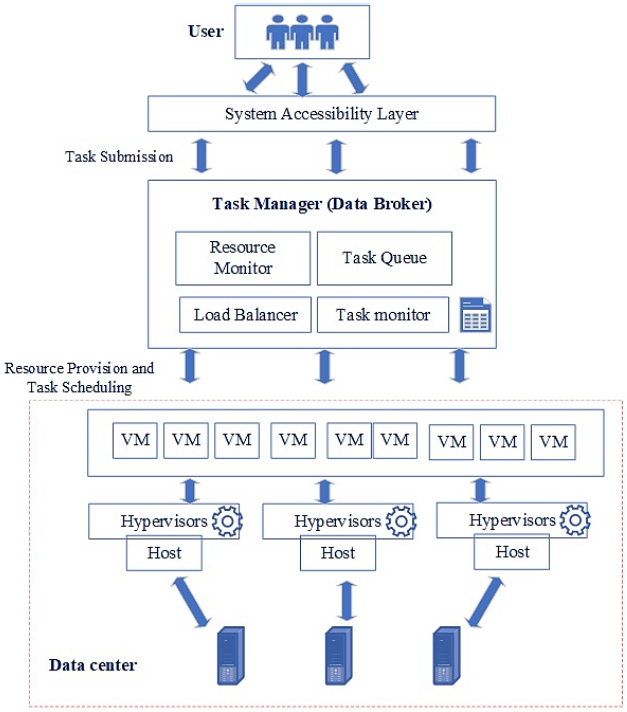


FIGURE 1. Cloud resource management framework.

the Cloud Service Providers (CSPs) [3]. Moreover, Cloud computing enables CSPs to provide data centers with high-performance computing resources to support users accessing cloud services.

A VM consumes the resources of the host machine: RAM, hard disk, or CPU. The resources on the VM are required by the requested resource for operation. As a result, the cloud network has an unequal resource distribution, and some VMs may not access the resources they require because many VMs have preemptive and non-preemptive connections to resources [4].

When a task is sent off to be processed in the cloud, the VM should be able to operate quickly to reduce waiting time. However, tasks should be distributed among all VMs in parallel to balance the system and ensure efficient use of available resources. For this reason, task scheduling to be assigned and distributed across existing resources is necessary. When multiple tasks are assigned to one VM or multiple VMs, the assigned tasks will run concurrently to complete the assignments. Therefore, the assignment must be ensured that not all tasks are loaded on only a single VM, which will cause other VMs to become inaccessible or imbalanced in the system. To avoid this, other conditions must be considered in scheduling, such as makespan, cost, and resource utilization. Several researchers have proposed ideas to manage load balancing in both heterogeneous [5] and homogeneous environments [6]. The main goal of allocating tasksof a system in a load balance state is to optimize the distribution of tasks across existing resources and to reduce the system processing time.

The cloud resource management framework is shown in Fig. 1. When a user submits a request into the system, the task is passed to the cloud broker, which is the main objective that researchers should focus on providing effective algorithms. The proposed method should efficiently submit tasks to the appropriate VM by the user depending on the required parameters, such as the deadline or other requirements. It must ensure that the user submitted the requests that are fulfilled in accordance with the requirements specified in the SLA document. Users make their requests over the internet, and the requests are stored in VMs. Then, the CSPs deliver QoS-based queries to ensure that the requests of the user can be processed as intended. This process depends on the performance of the scheduling policy (Data Broker). In this step, the workload balance must be adjusted between the machine and the server. Cloud computing depends highly on Virtual Machine Monitor (VMM) or hypervisors. Hypervisors help operate multiple VMs in the same hardware layer [7]. VMware is one of the most famous software services efficiently used for managing server resources in organizations. Even though virtualization plays an important role in cloud technology, problems still often arise, such as improper scheduling or handing over tasks to VMs that cannot meet the requests. To solve this problem, scheduling and load balancing between nodes in cloud computing should be implemented to optimize resource utilization. Scheduling and load balancing between nodes also resulted in a reduction in processing time and an imbalance in the system.

Task allocation to match the right resource, such as time, cost, and success rate, must be considered in cloud computing. Many studies have discussed scheduling or optimizing resources in the cloud by single or bi-objective optimization of QoS parameters [8]–[10], and several articles have discussed multi-objective optimization [11]. Examples of algorithms that were used to increase the efficiency of on-demand tasks such as Particle Swarm Optimization (PSO) [12], [13], Genetic Algorithm (GA) [14], niched Pareto genetic algorithm (NPGA) [15], and strength Pareto evolutionary algorithm (SPEA) [16].

The Artificial Bee Colony (ABC) algorithm is a meta-heuristic method used to solve problems and find solutions for an optimization answer that is close to the appropriate value developed by Karaboga [17], [18]. The ABC algorithm mimics the foraging behavior of bee colonies, which require adaptation to habitat and food environments. Several studies have demonstrated the effectiveness of the ABC algorithm in solving problems such as traveling salesman problem [19], and job shop scheduling problems [20]. The ABC algorithm works in a strategy similar to Reinforcement Learning (RL) with environment-based learning, which derives from observation to predict or decide a good solution. Agents learn how to behave in an environment by taking action and then observing the results for themselves. In this research, we use the Q-learning algorithm to help the system make predictions and decisions about choosing the appropriate schedule. Therefore, we propose a multi-objective optimization-scheduling

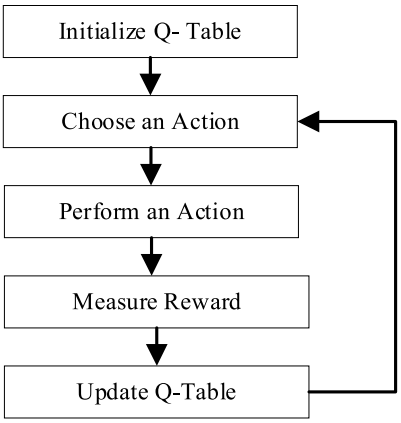


FIGURE 2. Q-learning algorithm process.

model using the ABC algorithm and Q-learning (MOABCQ) method for solving the scheduling problem in a cloud computing environment. The paper focuses on developing a multi-objective scheduling approach to optimize job scheduling with the goal of minimizing makespan, cost, and simultaneous use of resources, also considering the load balance of the system.

The main contributions of the proposed method can be described as follows:

- Formulate mathematical models for calculating resource utilization, makespan and cost for scheduling problems in cloud computing.
- Propose a multi-objective optimization scheduling model. This model consists of 3 objective functions defined as 1) execution time, 2) cost, and 3) utilization of resources. These are constraints in the proper task scheduling and the load balancing of the system.
- Propose a new approachthat is multi-objective optimization for the task scheduling problem using the Artificial Bee Colony algorithm (ABC) adapted with the implementation of the Q-learning algorithm, namely, MOABCQ, to optimize the multi-objective scheduling problem in cloud computing and to reduce the makespan, cost, and simultaneous use of resources. This new proposed method is combined with the First Come First Serve (FCFS) heuristic task scheduling called ''MOABCQ_FCFS'' and Largest Job First (LJF) heuristic task scheduling called ''MOABCQ_LJF''.
- Performance evaluation of our proposed method examined through CloudSim simulation. In this paper, we improved the classical Q-learning algorithm for load balancing in the cloud environment by combining it with ABC. This idea can improve the convergence rate, gain efficiency in load balancing, determine individual VM loads, and balance them through the fitness function.
- The proposed method was compared with other optimization algorithms, such as: First Come First Serve (FCFS) algorithm, Max-Min algorithm, Heuristic Task Scheduling with ABC with Largest Job First algorithm (HABC_LJF), Q-learning algorithm, multi-objective Particle Swarm Optimization (MOPSO) algorithm, and multi-objective Cuckoo Search (MOCS) algorithm. The three datasets used in this study are Random, Google Cloud Jobs (GoCJ), and Synthetic workload to observe in terms of makespan, cost, Average Resource Utilization Ratio (ARUR), throughput, and Degree of Imbalance (DI).

The rest of this article is structured as follows: Section II discusses the literature review. In Section III, the problem definition and formulation of the objective function will be described. The hybrid ABC algorithm for load balancing is presented in Section IV. In Section V, experimental evaluation and discussion will be explained, and finally, Section VI presents conclusions and future work.

II. LITERATURE REVIEW

Optimizing task scheduling and load balancing in cloud computing environments is known as an NP-hard problem [21]. Many studies have proposed optimization algorithms in cloud environment in various aspects, such as resource allocation, scheduling and load balancing procedures to reduce uptime, and system power consumption. In terms of efficient use of existing resources or fast processing, we need to consider an idea in scheduling multiple objectives or cloud environments.

Therefore, many ideas have been proposed using heuristic algorithms [22]–[29], meta-heuristic algorithm [30]–[37], hybrid meta-heuristic algorithm [14], [38], or even machine learning methods [39]–[41] to solve task scheduling and load balancing problems in the cloud computing environment. The related works are described as follows:

Many studies have used a heuristic algorithm to solve problems in a cloud computing environment. For example, A Max-Min based task scheduling algorithm was proposed by Mao *et al.* [22] to balance the load in the cloud and to predict the execution time of tasks. Then, the tasks were submitted to the VM based on their proposed Max-Min algorithm. The VM utilized more resources and decreased response time. Patel *et al.* [24] proposed an Enhanced Load balanced Min-Min (ELBMM) algorithm for static task scheduling. Jobs were assigned to VMs based on execution time, and the tasks were rescheduled to distribute to resources that idle.

Zhang *et al.* [25] studied the application of heuristic algorithms in cloud scheduling problems. They proposed a method for scheduling tasks to minimize the task completion time and execution cost (MCTE) in a smart grid-cloud system. Then, they performed mathematical modeling for the grid-cloud task scheduling problem.

Hussain *et al.* [26] proposed a resource-aware load-balancing algorithm (RALBA), which is a dynamic scheduling technique that maps independent and non-preemptive tasks to VMs. The process consisted of 2 parts. The first part was selecting the maximum size for the VM with the highest computational share. The second part was mapping the remaining tasks to the fastest working VM to perform. The results showed that RALBA was able to reduce makespan. However, there were some problems with

load imbalance and reducing resource utilization. The load balancing approach was proposed to be modified using soft computing-based stochastic hill climbing based on load balancing by Mondal *et al.* [27].

A dynamic multi-workflow scheduling algorithm named the competent dynamic multi-workflow scheduling (CDMWS) algorithm was proposed by Adhikari and Koley [28]. This method aimed to improve CPU utilization, reduce makespan, and improve the makespan-deadline meeting ratio. This method was classified into 2 parts. The first part served to estimate the processing time for each task based on deadline and task dependencies. The second part was responsible for allocating VMs to reduce power consumption and increase resource utilization. The experiments showed that CDMWS outperformed the Enriched Workflow Scheduling Algorithm (EWSA) and Round Robin (RR) algorithm.

Efficient resource allocation with a focus on solving energy efficiency problems using the multi-objective optimization (MOO) method was discussed by Shrimali and Patel [29]. A VM allocation approach has also been proposed using MOO. According to the experimental results, MOO led to energy savings due to the efficient allocation of resources without affecting the performance of the data center.

Many studies have used a meta-heuristic algorithm and a hybrid meta-heuristic algorithm to solve problems in a cloud computing environment. For example, Tawfeek *et al.* [35] discussed the adoption of the Ant Colony Optimization (ACO) algorithm for reducing the execution time of tasks in a cloud computing environment. The authors demonstrated that using ACO was efficient. ACO worked better than other methods, such as the RR algorithm and FCFS algorithm. To reduce the makespan and optimize resource access in cloud computing, an improved PSO algorithm has been developed. The method used updating the weights of particles with the evolution of the number of iterations and to inject some random weights in the final stages of the PSO. The objective was to avoid local optimum solutions being generated in the final stages of PSO was proposed by Luo *et al.* [12].

Chen *et al.* [36] proposed the dynamic objective genetic algorithm (DOGA) by focusing on optimizing the execution time according to the deadline constraint to help reduce the cost of work and to work within the specified time. Amini *et al.* [37] proposed the resources allocation process for virtual machines in cloud computing using dragonfly optimization algorithm. The experimental showed that using dragonfly optimization algorithm helped improve task scheduling, load balancing, and resource allocations better than ACO algorithm and Hybrid Algorithm Based on Particle Swarm Optimization and Ant Colony Optimization Algorithm (ACO-PSO).

Li and Han [42] proposed a hybrid discrete ABC algorithm for flexible task scheduling problems in a cloud system. The objective of this approach was to minimize the maximum completion time, total workloads of all devices, and maximum device workload. In terms of reducing the completion time and balancing load in the cloud computing environment, the Heuristic Task Scheduling with Artificial Bee Colony (HABC) Algorithm was proposed to schedule and manage cloud resources by Kruekaew and Kimpan [43]. The experimental results showed that HABC with the Largest Job First heuristic algorithm (HABC_LJF) was the most efficient in scheduling and managing cloud resources.

Gan *et al.* [44] proposed job scheduling using a genetic simulated annealing algorithm. The primary purpose was to optimize the execution time of data center tasks. Basu *et al.* [14] introduced a hybrid meta-heuristic algorithm called GAACO, which was a combination of the GA algorithm and ACO algorithm to solve the task scheduling of IoT applications in a multiprocessor cloud environment. The method guaranteed appropriate convergence when tested with sizes of task graphs and number of different processors in terms of makespan and efficiency. They found that the GAACO algorithm performed better than the GA algorithm and ACO algorithm in a heterogeneous multiprocessor environment.

For a multi-objective task scheduling problem in a cloud environment that is solved using a meta-heuristic algorithm and a hybrid meta-heuristic algorithm, Alsadie [45] proposed a meta-heuristic framework for dynamic virtual machine allocation with optimized task scheduling in a cloud computing environment (MDVMA). MDVMA was a multi-objective scheduling method using a non-dominated sorting genetic algorithm to help reduce cost, makespan, and energy usage. In the experiments, theHeterogeneous Computing Scheduling Problems (HCSP) dataset was used with the CloudSim Simulator. The results showed that MDVMA was able to optimize task scheduling better than the ABC algorithm, Whale Optimization Algorithm (WOA), and PSO algorithm in terms of reducing the cost, makespan, and energy usage of the cloud data center.

Guo [34] proposed cloud computing multi-objective task scheduling optimization based on a fuzzy self-defense algorithm, which had good performance in terms of the shortest completion time, deadline violation rate, and utilization of virtual machine resources when compared with A Multi-Objective Optimization Scheduling Method Based on the Ant Colony Algorithm in Cloud Computing (PBACO) [33] and Task Scheduling Algorithm Based on RL. Zuo *et al.* [33] proposed a multi-objective optimization scheduling method in terms of efficiency and budget costs. This approach used an ant colony algorithm (PBACO) to help determine the optimal solution. The experiment was compared with the classical heuristic algorithm, Min-Min algorithm, and FCFS scheduling. PBACO was found to be superior to the other comparison methods.

He *et al.* [46] proposed Adaptive Multi-objective Task Scheduling (AMTS) to try to improve resource utilization, cost, energy consumption, and task completion time. AMTS used a PSO-based approach for multi-objective scheduling that considered process time and transmission time.

The proposed method applied the adaptive acceleration coefficient for particle diversity. After improving the PSO

TABLE 1. Simulation environment.

Type	Parameter	Value
Host	Number of Host	20
	MIPS	177,730
	Bandwidth	10 GB/s
	Storage	2 TB
	RAM	16 GB
	VM Monitor	Xen
Data Center	Number of Data Center	1
	VmScheduler	Time Shared
	Cost per Memory	0.1 - 1.0
	Cost per Storage	0.1 - 1.0
	VM Monitor	Xen
Cloudlet (Task)	Length of Tasks	1k – 900k
	Number of tasks	200-1,000
Virtual Machine (VM)	Number of VMs	5 - 100
	Processor speed	3,500 - 100,000 MIPS
	Memory	1 – 4 GB
	Bandwidth	1,000 – 10,000
	Cost per Memory	0.1 - 1.0
	Cost per Storage	0.1 - 1.0
	Cloudlet Scheduler	Time Shared
	Number of PEs	1
	VM Monitor	Xen

algorithm, AMTS was able to find the best solution for the cloud-based scheduling problem.

Jena [47] proposed an ABC-based approach for energy efficiency, processing time, cost, and computing resource utilization in the cloud computing environment. Tasks were allocated to the data center using a multi-objective ABC algorithm. An ant-based genetic algorithm was used to solve multi-objective scheduling problems to reduce latency and completion time while maximizing throughput in the cloud environment by Kumar and Venkatesan [48]. The multi-objective WOA was proposed by Reddy and Kumar [49] to develop scheduling in cloud computing. The authors tried to create a fitness-based schedule based on three conditions: quality of service, energy, and resource utilization. After considering the given parameters, the processing time and cost of the virtual machines were found to be reduced.

Madni *et al.* [50] proposed an innovative Multi-objective Cuckoo Search Optimization (MOCSO) algorithm for the resource scheduling problem in IaaS cloud computing environment. The major goal of the resource scheduling challenge was to reduce cloud user costs and enhance the performance by reducing makespan time. The simulation results showed that MOSCO algorithm outperformed Multi-objective ACO (MOACO), Multi-Objective GA (MOGA), Multi-Objective Min-Max (MOMM), and Multi-objective PSO (MOPSO) algorithm.

Pang *et al.* [51] developed an EDA-GA hybrid scheduling algorithm to solve the multi-objective task scheduling problem based on EDA (estimation of distribution algorithm) and GA (genetic algorithm). The constraints of scheduling problem in this model were scheduling performance and time. The advantages of this algorithm were the fast convergence speed and strong search ability. The algorithm was compared with EDA and GA using the CloudSim simulation experiment platform. According to the testing results, the EDA-GA hybrid algorithm effectively reduced job completion time and improved load balancing ability.

Neelima and Reddy [52] proposed a load balancing task scheduling algorithm in cloud using Adaptive Dragonfly algorithm (ADA) which was a combination of dragonfly algorithm (DA) and firefly algorithm (FA). The development of a multi-objective function was based on three parameters: completion time, processing costs, and load. The performance of this method was measured in terms of execution cost and time. When compared to DA and FA, the experimental results showed that the proposed approach achieved better load balancing results.

There is also research combining osmotic behavior with bio-inspired algorithms, for example, Gamal *et al.* [53] presented Osmotic hybrid artificial bee and ant colony (OH_BAC) algorithm which derives from the osmosis theory in the chemistry science. This algorithm was used to form osmotic computing and find load balancing for VM placement. OH_BAC used an osmosis approach to create a low-energy cloud computing environment. In this algorithm, ACO and ABC collaborated to find the optimum VM for migrating to the best physical machine (PM). To reduce power consumption, OH_BAC activated the most appropriate osmotic host among all PMs in the system. The algorithm was compared with ACO, ABC, H_BAC, and host overloading detection algorithms. The experimental results showed that when compared to other algorithms in fixed and variable loads, OH_BAC improved energy consumption, service level agreement violation (SLAV), number of virtual machine migration, and number of host shutdowns.

Machine learning algorithms have been used to solve challenging problems in cloud computing environments [54]. For example, Farahnakian *et al.* [39] proposed the Reinforcement Learning-based Dynamic Consolidation method (RL-DC) to reduce energy consumption and optimize resource usage in cloud data centers.

Caviglione *et al.* [55] introduced a deep reinforcement learning-based approach for the placement of VMs in cloud data centers. DRL was used to select the best location possible for each VM. Jena *et al.* [56] proposed a hybrid meta-heuristic algorithm called QMPSO for balancing loads between VMs in cloud computing using hybridization of modified particle swarm optimization (MPSO) and an improved Q-learning algorithm. QMPSO helped to improve the makespan, throughput, and power consumption during load balancing and effectively reduced the waiting time of tasks.

A framework for cloud resource allocation with the goal of deploying resources in green cloud was proposed by Thein *et al.* [57]. The proposed framework was based on a reinforcement learning mechanism and fuzzy logic. Its efficiency was measured by CPU utilization at the data center,

which measuring Power Usage Effectiveness (PUE) and Data Center infrastructure Efficiency (DCiE). Simulation results using CloudSim showed that this framework can achieve effective performance for high data center energy efficiency and prevent SLA violation. The goal of the task scheduling and resource allocation model by hybrid ant colony optimization and deep reinforcement learning is to reduce the completion time and improve the utilization of idle resources with the use of a binary in-order traversal tree using weighted values. For task scheduling, a DRL algorithm was used to find idle resources and ACO to find the most suitable VM for each task by Rugwiro *et al.* [41].

As mentioned previously, developing effective algorithms for task scheduling and selecting the right resources in cloud computing environments was found to be very important. Many studies put much effort into organizing tasks or allocating proper resources for each task considering single objective, bi-objective and multi-objective scheduling. Therefore, this paper proposed the multi-objective optimization of the task scheduling problem, while previous studies focused mainly on the objectives of makespan or processing time and cost because both of these objectives meet the needs of the users. However, to work in cloud computing, it is necessary to consider various conditions or objectives. In addition, load balancing in the cloud must be considered. A hybrid meta-heuristic algorithm has been introduced to help optimize performance in the cloud. Nevertheless, some algorithms are weak in local search, while some algorithms have a weakness in global search optimization. According to related studies, the ABC algorithm can be helpful in solving the multi-objective task scheduling problem because it is capable of explorative behavior. In contrast, exploitative behavior, which is a part of the onlooker bee, was weak. In this case, reinforcement learning can help solve this problem because Q-learning can improve the solution quality. Therefore, we propose a method that can solve the multi-objective task scheduling problem using the ABC algorithm and Q-learning (MOABCQ). MOABCQ helps determine the order of tasks for suitable available resources environments to find the most appropriate task scheduling solution. Moreover, MOABCQ also provides a load balancing system.

III. PROBLEM DEFINITION AND FORMULATION OF OBJECTIVE FUNCTION

Many users operate their works on the cloud environment, which makes the scheduling process play an important role in resource utilization, response time, latency, and load balancing. Task scheduling problems can be described as follows.

Input:

- Let $V = \{v_1, v_2, v_3, \ldots, v_m\}$ where V is a collection of VMs, and m is the total number of VMs in the cloud network. Each VM has its own resources (e.g., CPU, RAM, and bandwidth) and the cost of usage and the computing power are defined differently; v_i presents the i^{th} VM.
- Let $T = \{t_1, t_2, t_3, \ldots, t_n\}$ where T is the set of assigned tasks, and n is the number of independent tasks performed on the VMs (m). t_j represents the j^{th} task. Each task submission contains the number of instructions, memory required, CPU required, etc.

Output:

- Optimize scheduling, map n tasks and m VMs (resources) to minimize makespan, cost, and resource utilization and to increase load balancing in resource utilization.

In general, multi-objective optimization problem consists of several fitness functions (Objective function) as: $F(x) = [f_1(x), f_2(x), \ldots, f_{obj}(x)]$ where obj is number of objectives, and $f_i(x)$ represents i^{th} objective function. This problem does not have a single solution and a set of non-dominated solutions can be found, known as a pareto optimal solutions.

We propose the task scheduling algorithm in cloud computing using conditions of multi-objective scheduling approaches to increase the efficiency of schedule optimization and resource utilization, to maximize VM throughput, and create load balancing between VMs. The most common evaluation factors are cost, energy consumption, task completion time, task waiting time, flow time, failure rate, profit, carbon emission, makespan, and reliability [58]. The proposed method determines the suitability conditions using cost, resource utilization, and makespan as factors that help in proper scheduling. Scheduling and load balancing in a cloud computing environment requires a fitness function design to provide optimal solutions and also considers the multi-objective used to find the answer. We used weighted sum approach [59], [60] for solving the multi-objective optimization problem. This approach is used to convert a multi-objective optimization problem to a single objective with weights which represents preferences among objectives by the decision maker. We have considered three objectives in the article as follows:

1) *The first objective* is defined in the condition of makespan or task execution time, which is the time that the system completes its last task. The makespan is a useful factor for multi-objective scheduling approaches that can reduce the execution time and allow tasks to be completed prematurely. Each VM has a different execution time for completing the task determined by the makespan. If a maximum of the execution time value is high and the makespan value is also high, the system is considered poorly distributed tasks to the VMs. However, if the maximum execution time value is low, the makespan value is also low. Thus, the system can evenly distribute tasks among the resources in the system.

Considering where each task t_j $(t_j \in T)$ is assigned to the VM, v_i ($v_i \in V$) is represented by t_{ji}, so the VM task is represented by $v_i = \{t_{xi}, t_{yi}, \ldots, t_{zi}\}$.

The total execution time (ET) of task processing on v_i can be obtained by (1).

$$ET(v_i) = \sum_{t_{ji} \in v_i} ExtTime(t_{ji}) = \frac{\sum_{t_{ji} \in v_i} length(t_j)}{CPU(v_i)} \quad (1)$$

where $ExtTime\left(t_{ji}\right)$ is the execution time of t_j processing in v_i. This value can be calculated by (2).

$$ExtTime\left(t_{ji}\right) = \frac{length(t_j)}{CPU(v_i)} \tag{2}$$

where $length(t_j)$ is the length of the j^{th} task, the length of the task is defined in terms of the number of instructions (million instructions), and $CPU(v_i)$ is the CPU rate used to process the j^{th} VM in the cloud. Makespan is the maximum value of the execution time of all VMs and can be calculated by (3).

$$Makespan = Max(ExtTime(v_i)), \quad 1 \leq i \leq m \tag{3}$$

MinMakespan is a lower bound of makespan, which is the minimum time required by the system to complete all tasks. MinMakespan is calculated by (4).

$$MinMakespan = Min(ExtTime(v_i)), \quad 1 \leq i \leq m \tag{4}$$

Fitness function in terms of makespan (F_1) can be calculated by (5).

$$F_1 = \frac{MinMakespan}{Makespan} \tag{5}$$

2) *The second objective* is defined in terms of cost which is the cost of requesting to be processed by the task and can be calculated from CPU cost, memory usage cost, and bandwidth usage cost. Estimated cost when t_j is processed at v_i can be calculated by (6).

$$Cost\left(t_{ji}\right) = (c_1 * ExtTime(t_{ji})) + (c_2 * ExtTime(t_{ji})) + (c_3 * ExtTime(t_{ji})) \tag{6}$$

where c_1, c_2, c_3 are CPU usage cost per unit, memory usage cost per unit, and bandwidth usage cost per unit in v_i, respectively.

The total cost ($TCost$) is calculated as the sum of all tasks processing on all VMs, which can be calculated from (7).

$$TCost = \sum_{j=1}^{n} \sum_{i=1}^{m} Cost(t_{ji}) \tag{7}$$

MinTCost is the lowest cost when the set of assigned tasks T is processed in the VM where the VM process task t_j gives the lowest cost, called the $MinTCost(t_j)$ value, which means that MinTCost is only for task t_j and can be calculated from (8).

$$\begin{aligned} MinTCost &= \sum_{t_j \in T} MinCost\left(t_j\right) \\ &= \sum_{t_{ji} \in T} min_{1 \leq i \leq m}(Cost\left(t_{ji}\right)) \end{aligned} \tag{8}$$

The fitness function in terms of cost (F_2) can be calculated by (9)

$$F_2 = \frac{MinTCost}{TCost} \tag{9}$$

3) *The third objective* is defined in terms of the utilization of resources (CPU and memory) sent to a different number of processing units than in the cloud network. If the requested task is sent to v_i, we can calculate the memory load of v_i by (10).

$$LM_i = AM_i + \frac{RM_j}{TM_i} \tag{10}$$

where AM_i is the amount of memory usage before executing task t_j at the i^{th} VM

RM_j is the memory containing the request of the j^{th} task

TM_i is the total memory available at the i^{th} VM

The next parameter is the CPU. If the requested task is sent to v_i, we can calculate the CPU load of v_i (LC_i) using (11).

$$LC_i = AC_i + \frac{RC_j}{TC_i} \tag{11}$$

where AC_i is the amount of CPU usage before executing task t_j at the i^{th} VM

RC_j is the CPU that requests the j^{th} task

TC_i is the total CPU available at the i^{th} VM.

VM utilization evaluation (VU_i) [61] can be calculated by (12).

$$F_3 = VU_i = \frac{\omega_1}{1 - LM_i} * \frac{\omega_2}{1 - LC_i} \tag{12}$$

where ω_1 and ω_2 are the weights for the CPU and memory, respectively. $\omega_1 + \omega_2 = 1$ in this paper, given ω_1, ω_2 are both equal to 0.5 because CPU and memory are equally important.

Total load on k host (LH), where k is the total number of hosts in the system, can be calculated from (13).

$$LH_k = \sum_{i=0}^{m_k} VU_{ki} \tag{13}$$

The average load on all physical machines in cloud (AL) can be calculated by (14).

$$AL = \frac{\sum_{k=0}^{p} LH_k}{p} \tag{14}$$

where p is the number of hosts in the cloud network.

The difference in load among each host and the average load on the cloud network is calculated from $|LH_k - AL|$. The fitness function is defined in terms of the utilization of resources, which can be calculated by (15).

$$F_3 = \sum_{k=0}^{p} |LH_k - AL| \tag{15}$$

Fitness function is created by calculating the weighted average of each individual fitness function. The proposed fitness function (F) is shown in (16).

$$F = (\gamma_1 * F_1) + (\gamma_2 * F_2) + (\gamma_3 * F_3) \tag{16}$$

where γ_1, γ_2, γ_3, and $\gamma \epsilon [0, 1]$ are the balance coefficients between makespan, total cost, and resource utilization. Maximizing the utility function (F) results in a better solution.

IV. HYBRID ABC ALGORITHM FOR LOAD BALANCING

A. Q-LEARNING ALGORITHM

Q-learning [62] is one of the Reinforcement Learning (RL) algorithms. RL is one of the machine learning methods that allows agents to learn in their environment and action by changing their state to receive rewards or penalties based on the feedback obtained from the environment. The main purpose of RL is to learn the agent through trial and error between the agent and the environment. The agent is able to receive the environment situation through a state and choose an action that affects the environment to obtain the best reward and learn through past mistakes. The Markov decision process (MDP) is a framework for the decision-making of agents because of the uncertain environment, and the result is sometimes stochastic. The agent chooses to perform the same action for the same situation or state, but it may not always obtain the same result. The Q-learning algorithm process is shown in Fig. 2.

Given that the set of states $S = \{s_1, s_2, s_3, \ldots, s_n\}$ is in the environment, each state has a set of actions. The $A = \{a_1, a_2, a_3, \ldots, a_m\}$ agent selects action $a_t \epsilon A$ at time t in state $s_t \epsilon S$ to pass to the next state $s_{t+1} \epsilon S$ through the transition process and receives a reward r_{t+1} from the environment. To process the tasks, it is necessary to select the appropriate action to maximize the Q-value of each state, which is the primary objective of finding the optimal policy in cloud computing. The Q-value function depends on the selection of action in the state. Given the agent in state s_t and selecting action a_t, the Q-value function is expected to move to the best state and gain to maximize the total expected reward in the environment. The Q-value can be calculated by (17)

$$Q(s_t, a_t) = (1-\alpha)\, Q(s_t, a_t) + \alpha\, [r_t + \gamma max_{a+1} Q(s_{t+1}, a_{t+1})] \quad (17)$$

where α is the learning rate, $\gamma (0 < \gamma < 1)$ is the discount factor and effect on the successive state by the previous action, and r_t is the penalties or rewards awarded for performing actions in the state s_t. The Q-value derives from creating a Q-table that stores all possible states, Q-values, and appropriate actions. The Q-learning algorithm attempts to establish the optimal state from their experience, and the greedy algorithm is implemented in the Q-learning algorithm. Q-value can be computed using (18).

$$Q_{t+1}(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_t + \gamma max_{\acute{a_t}} [Q_t\left(\delta(s_t, a_t), \acute{a_t}\right)] - Q_t(s_t, a_t)\right] \quad (18)$$

where α is the learning rate, calculated from $\alpha = 1/(1 + \text{total number of visits to state } s_t)$, and δ is the transition function.

$max_a Q(s_{t+1}, a)$ is an estimate of optimal future value. A possible action in cloud computing involves load balancing, which allocates resources in VM and can be described in Algorithm 1.

Algorithm 1 Q-Update(S)

1. Set values for learning rate α, discount factor γ, reward r
2. Initialize Q-values
3. For $i = 1$ to n # n is the number of states
4. For $j = 1$ to p # p is the number of actions in each state
5. $Q\left(s_i, a_j\right) = 0$
6. End for
7. End for
8. Select an action a_i from $A = \{a_1, a_2, a_3, \ldots, a_m\}$ and execute it and go to next state s_{t+1}
9. Calculate the learning rate
10. Calculate the reward
11. Update Q_{t+1} by (18)
12. Repeat this step for new state until it converges

B. MODIFIED ABC ALGORITHM

The ABC algorithm [19] is a meta-heuristic optimization algorithm based on swarm intelligence. The swarm system consists of agents that communicate with other agents and their environment. In this algorithm, the goal of the agent is to find the best food source, where the food source represents a set of possible answers in the search space and each agent is represented by a bee.

In ABC, agents are categorized according to the functions of the bees and can be classified into 3 groups: employed bees, onlooker bees, and scout bees. Initially, each employed bee finds a random food source, whereas in each iteration, employed bees find a new food source near a current food source. After collecting the nectar, each employed bee assesses the best food source. Then, the bee will move to a new food source only if the bees have determined that new food sources are better than the previous one. The employed bees share information with the onlooker bees. The onlooker bees decide to choose new food sources based on information obtained from the employed bees. If any food source has much food or high quality, it will have a high chance of being chosen by onlooker bees. Then, each onlooker bee will find a new food source around it. They select the food source and move to a new food source. The number of iterations is also predetermined, meaning that if no better food source is found, the employed bees who own the selected food source become scout bees and are responsible for exploring the new food source in a new area of search space.

Each group of bees has different explorative and exploitative behaviors [63]. The explorative behavior of a search agent involves searching for a new food source in the search space to avoid a local optimum. In contrast, exploitative behavior involves searching for a better food source near the current food source. In this paper, we use Q-learning to improve our solution to provide more appropriate solutions to problems. The ABC algorithm has both strong explorative behavior and weak exploitative behavior; therefore, we aim to

optimize exploitative behavior. The process can be described as follows:

In the initialization phase, the ABC algorithm represents the location of the food source with possible solutions to the problem. Initially, the food source is randomly generated as shown in (19). Then, employed bees are associated with food sources. The initial value of the Q-table is 0.

$$x_{i,j}^{0} = x_{j}^{min} + rand\ (0, 1) * (x_{j}^{max} - x_{j}^{min}) \tag{19}$$

where x_j^{min} is the lower bound of the j^{th} optimization parameter, and x_j^{max} is the upper bound of the j^{th} optimization parameter.

In addition to the initialization phase, the ABC algorithm separates the algorithm into three sub-phases: employed bee phase, onlooker bee phase, and scout bee phase. The algorithm repeats all three phases until a certain maximum number of values is reached.

In the employed bee phase, the employed bees [64] find the location of the neighboring food source $v_{i,j}^t$ of the current food source ($\bar{x}_i^t$) using (20).

$$v_{i,j}^{t} = x_{i,j}^{t} + rand\ [-1, 1] * (x_{i,j}^{t} - x_{k,j}^{t}) \tag{20}$$

where $v_{i,j}^t$ is the j^{th} optimization parameter of $\bar{v}_i^t$, and k is the index of the food source.

If the new food source $\bar{v}_i^t$ returns a fitness value greater than the current food source x_i^t, $Fit\left(\bar{v}_i^t\right) > Fit\left(\bar{x}_i^t\right)$, employed bees forget the current food source and remember the new location. In this step, the Q-table (reward-penalty scheme) value is updated using (18). If the new food source provides a better fitness value, the employed bee will not only replace the current food source with the new food source but also update the Q-value by rewarding the selected new food source and penalty with the current food source.

In contrast, if the new food source does not provide a better fitness value, the new food source receives a penalty, and the current food source receives a reward. The Q-table is updated every time that an employed bee finds a suitable food source. Therefore, if the number of employed bees is Emp, Q-table will update Emp once.

In the onlooker bee phase, the onlooker bee selects the employed bee's food source from the Q-value in the Q-table. In contrast, the onlooker bee searches for the new food source using (21) and replacing the current food source with the new food source, if the new food source has a higher fitness value, then the Q-value will also be updated.

$$v_{i,j}^{t} = x_{i,k}^{t} + \emptyset_{i,j}^{t} \cdot r_{j}^{t} \cdot (x_{i,k}^{t} - x_{RFS,k}^{t}) \tag{21}$$

where $v_{i,j}^t$ is the optimization parameter of a neighboring food source $\bar{v}_i^t$, $\emptyset \in [-1, 1]$, and $x_{RFS,k}^t$ is the optimization parameter of the optimal food source caused by random selection.

Using (21) to improve exploitation instead of updating values in dimension, onlooker bees exploit current food and update all dimensions with different weight values, and in this step, Q-values (reward and penalty) are updated.

In the scout bee phase, if there are a number of unsuccessful attempts to find a better neighbor, that food source will be discarded, and the scout bee will randomly search for a new food source.

The pseudo-code of the MOABCQ method is presented in Algorithm 2.

Algorithm 2 MOABCQ Method

Initialization:
1. Initialize the population and calculate individual fitness values
2. Set up the parameter: best solution, maximum number of iterations, the population size
3. Find the best solution
4. while stopping criteria satisfied

Employed Bees Phase:
5. For each position do
6. Update position of employed bee by (20)
7. Estimate the new position
8. If fitness value of new position is better
9 Replace the current position with the new position
10 End if
11. Calculate probability and update the Q-table for select position in the onlooker bee phase
12. End for

Onlooker Bees Phase:
13. For each onlooker do
14. Select a position based on Q-value and probability
15. Update position of onlooker bee by (21)
16. Estimate the new position
17. If fitness value of new position is better
18. Replace the current position with the new position
19. End if
20. Update the Q-table using (18)
21. End for
22. Find the best solution (Q-table)

Scout Bees Phase:
23. For each position do
24. Abandon the solution that have not been updated and generate new solutions randomly
25. update the Q-table using (18)
26. End for
27. End while

V. EXPERIMENTAL EVALUATION AND DISCUSSIONS

In this section, the parameter settings and experimental results are described. To evaluate the effectiveness of the proposed methodology (MOABCQ), we compared it with well-known heuristic task scheduling algorithms, such as Max–Min task scheduling algorithm [22], First Come First Serve (FCFS) algorithm, and Largest Job First (LJF) algorithm. Moreover, we compared MOABCQ method with the

TABLE 2. Parameter settings of meta-heuristic algorithms.

Algorithms	Parameter	Values
HABC [43]	Number of population (n)	960
	Number of sites selected out of n visited sites (m)	96
	Number of best sites (e)	1
	Number of Employed bees	192
	Number of Onlooker bees	768
	Maximum iterations	1000
MOABCQ	Number of population (n)	960
	Number of sites selected out of n visited sites (m)	96
	Number of best sites (e)	1
	Number of Employed bees	192
	Number of Onlooker bees	768
	Maximum iterations	1000
MOPSO [67]	Number of particle size	100
	Weight (w_{min}, w_{max})	0.1, 0.9
	Self-recognition coefficients (c_1, c_2)	1.49445
	Maximum iterations	1000
MOCS [50]	Number of population size	20
	Abandon probability (P_a)	0.25
	Step size (λ)	0.01, 1
	Maximum iterations	1000

popular meta-heuristic task scheduling algorithms such as the PSO algorithm and CS algorithm, and with our previous method called Heuristic Task Scheduling with Artificial Bee Colony algorithm and largest job first (HABC_LJF) [43].

We considered evaluating the performance of the proposed method, which consists of a simulation environment, a benchmark datasets, parameter settings for the proposed method and the comparison algorithms, experimental results, and the time complexity of MOABCQ method.

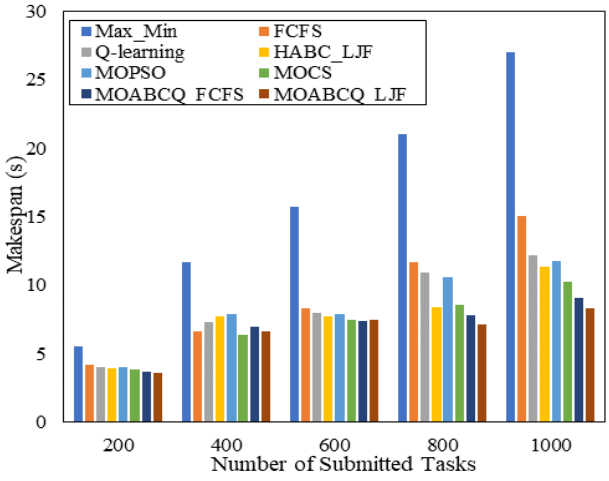
A. SIMULATION ENVIRONMENT

This section presents an experiment conducted to evaluate the performance of the proposed method (MOABCQ) when comparing task scheduling with other methods in a heterogeneous cloud computing environment. In this paper, a simulation was designed and developed using the CloudSim 3.0.3 simulator [65]. CloudSim is the most widely used simulator to implement clouds. CloudSim is a tool that can simulate virtual resources, and CloudSim can also support modeling, simulation, and experimentation of virtualized cloud-based data. This experiment was simulated on a computer with an Intel Core i7-8750H CPU (clock speed of 2.20 GHz) and 16 GBs of RAM.

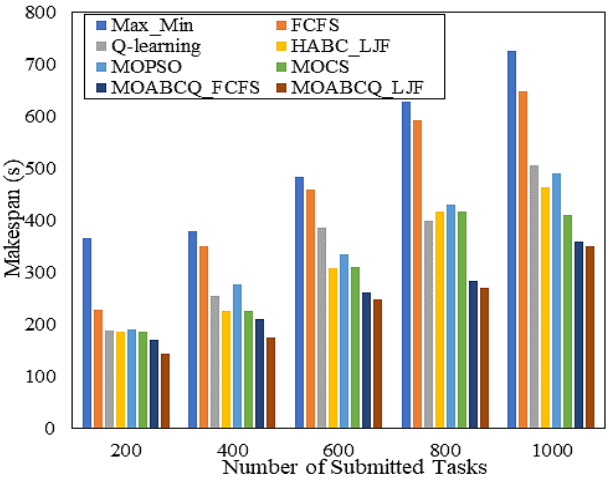
In this experiment, a virtual environment was simulated to demonstrate the efficiency of the proposed method in terms of scheduling and load balancing in a cloud computing environment. The simulation environment of this experiment was defined as shown in Table 1.

B. BENCHMARK DATASETS

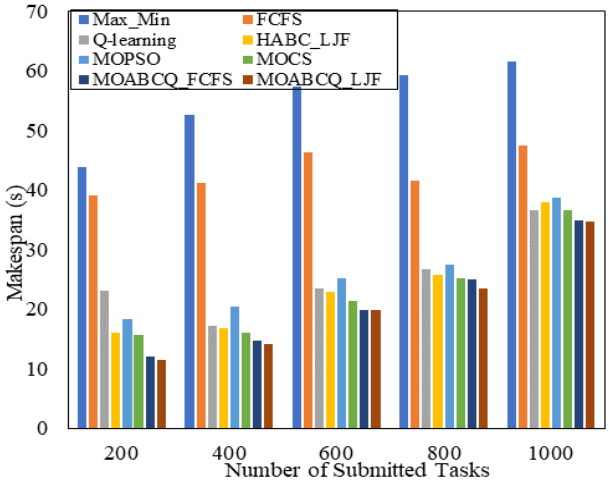
To evaluate the scheduling efficiency of the proposed method, three different datasets were used: 1) Random dataset, 2) Google Cloud Jobs (GoCJ) dataset [66], and 3) Synthetic workload dataset [26], which are described as:



(a) Makespan comparison using the random dataset



(b) Makespan comparison using the GoCJ dataset



(c) Makespan comparison using the Synthetic workload dataset

FIGURE 3. Comparison of the performance in terms of makespan on various datasets.

TABLE 3. Comparison of the performance in terms of ARUR on various datasets.

Dataset	Task Scheduling Approach							
	Max-Min	FCFS	Q-learning	HABC_LJF	MOPSO	MOCS	MOABCQ_FCFS	MOABCQ_LJF
Random	0.200	0.332	0.660	0.642	0.637	0.698	0.762	**0.812**
GoCJ	0.190	0.252	0.610	0.692	0.684	0.702	0.742	**0.792**
Synthetic workload	0.178	0.505	0.755	0.723	0.722	0.761	0.796	**0.801**

1) RANDOM DATASET

We generated task sizes ranging from 1k – 70k Million Instructions (MIs). The randomly generated dataset contains a total of 1000 tasks. The dataset contains task size, number of requested CPUs, and amount of RAM being requested.

2) GOCJ DATASET

GoCJ dataset is considered a Google-like realistic dataset generated from the workload behaviors witnessed in Google cluster traces using bootstrapped Monte Carlo, a well-known simulation method. The task sizes in the GoCJ dataset range from 15k – 900k MIs, and the datasets are classified as: small size jobs (15k – 55k MIs), medium size jobs (59k – 99k MIs), large size jobs (101k – 135k MIs), extra-large size jobs (150k – 337.5k MIs), and huge size jobs (525k – 900k MIs).

3) SYNTHETIC WORKLOAD DATASET

Synthetic workload dataset iscreated by random-number generator mechanism using Monte Carlo simulation method. It consists of different tasks sizes from 1 – 45K MIs which are tiny size jobs (1-250 MIs), small size jobs (800–1200 MIs), medium size jobs (1800–2500 MIs), large size jobs (7k–10k MIs), and extra-large size jobs (30k–45k MIs).

C. PARAMETER SETTINGS FOR THE PROPOSED METHOD AND THE COMPARISON ALGORITHMS

In this experiment, we defined population parameter and other conditions from related articles which are: HABC algorithm [43], PSO algorithm [67], and CS algorithm [56]. As it is already recognized that parameter setting has effects on the efficiency of algorithms and it depends on size or nature of the problem. For this reason, tuning the parameters must be done to ensure that we use the appropriate parameters for the problem type and dataset. We did not claim, however, that the proposed method or its parameters outperform alternative algorithms for all type of problems and datasets. In the experiment, the parameters of the ABC algorithm proposed by Kruekaew and Kimpan [43] are defined in Table 2. In addition, to compare the scheduling performance, the proposed method was compared with the HABC algorithm, MOPSO algorithm, and MOCS algorithm. The parameters are defined as shown in Table 2 which based on the original papers.

D. EXPERIMENTAL RESULTS

This section describes an experimental evaluation of a proposed scheduling approach using the benchmark dataset to compare experimental results in terms of makespan, throughput, ARUR [26], cost, and DI. In this experiment, we assessed the effectiveness of the proposed method (MOABCQ). We combined the MOABCQ method with the First Come First Serve (FCFS) heuristic task scheduling called "MOABCQ_FCFS" and Largest Job First (LJF) heuristic task scheduling called "MOABCQ_LJF" and compared them with other well-known algorithms: FCFS scheduling algorithm, Max–Min task scheduling algorithm [22], Heuristic Task Scheduling with Artificial Bee Colony algorithm and largest job first (HABC_LJF) [43], Q-learning algorithm, multi-objective particle swarm optimization (MOPSO) algorithm, and multi-objective cuckoo search (MOCS) algorithm. Each dataset was run for 20 rounds, and the average of the results is proposed in the following section.

The first section presents a comparison of the performance of the proposed method in terms of makespan. This experiment was assigned to 100 VMs, and 200, 400, 600, 800, and 1000 tasks were assigned to the system. In the experiment, we used the 3 datasets mentioned earlier. The experimental results are shown in Fig. 3. According to the experimental results in Fig. 3(a), when the random dataset was tested, the MOABCQ method was found to reduce the average makespan better than the Max-Min method, FCFS, HABC_LJF, Q-learning, MOPSO, and MOCS. However, when 400 datasets were tested, MOCS gave the lowest average makespan compared to the other methods. MOCS took 3.69%, 3.77%, 8.82%, 13.85%, 20.77% 23.41%, and 82.39% less time to complete than FCFS, MOABCQ_LJF, MOABCQ_FCFS, Q-learning, HABC_LJF, MOPSO, and Max-Min, respectively.

If we consider the insight of MOABCQ, after comparing MOABCQ_FCFS and MOABCQ_LJF, MOABCQ_LJF can be found to give a lower average makespan than MOABCQ_FCFS in the case of 200, 800, and 1000 tasks. However, in the case of 600 tasks, MOABCQ_FCFS had an average makespan 0.51% less than MOABCQ_LJF.

Considering the experimental results in Fig. 3(b), in which the GoCJ dataset was used, and Fig. 3(c), in which the Synthetic workload dataset was used, MOABCQ_LJF gave the lowest average makespan. When using the GoCJ dataset, MOABCQ_LJF gave average makespan approximately 117.80%, 92.15%, 46.17%, 42.25%, 34.94%, 30.62%, and 8.21% less than the makespan of Max-Min, FCFS, Q-learning, MOPSO, HABC_LJF, MOCS, and

MOABCQ_FCFS, respectively. When using the Synthetic workload dataset, MOABCQ_LJF yielded average makespan of approximately 150.75%, 98.61%, 25.53%, 22.87%, 15.35%, 10.76%, and 2.97% less than the makespan of Max-Min, FCFS, MOPSO, Q-learning, HABC_LJF, MOCS, and MOABCQ_FCFS, respectively. It can be indicated that MOABCQ_LJF outperformed the other methods in running both datasets with all test conditions except the percentage of the average makespan reduction.

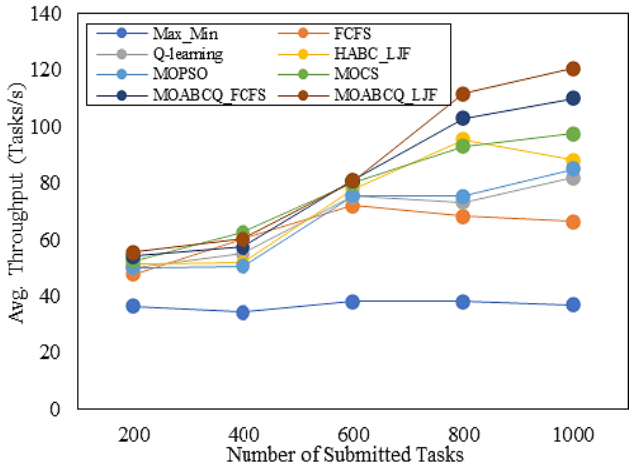
However, overall examinations showed that the proposed method can provide the lowest makespan value because MOABCQ_LJF can allocate the task to the appropriate resource. According to all these results, we can conclude that the proposed method has the potential to efficiently allocate resources in the system.

The second section presents a comparison of the efficiency of the method presented in terms of throughput, which is the number of tasks executed per unit of time. The 3 previously used datasets were used in this experiment. The experimental results of testing the efficiency with a random dataset are shown in Fig. 4(a), which indicates that MOABCQ had better throughput than other algorithms. However, when testing in 400 tasks, MOCS gave greater values than the others at 3.56%, 3.63%, 8.10%, 12.16%, 17.20%, 18.97%, and 45.17% when compared to FCFS, MOABCQ_LJF, MOABCQ_FCFS, Q-learning, HABC_LJF, MOPSO, and Max-Min, respectively. Considering the MOABCQ itself in depth, MOABCQ_LJF has a higher throughput than MOABCQ_FCFS in 200, 800, and 1000 tasks. In the case of 600 tasks, MOABCQ_FCFS provided 0.50% higher throughput than MOABCQ_LJF.

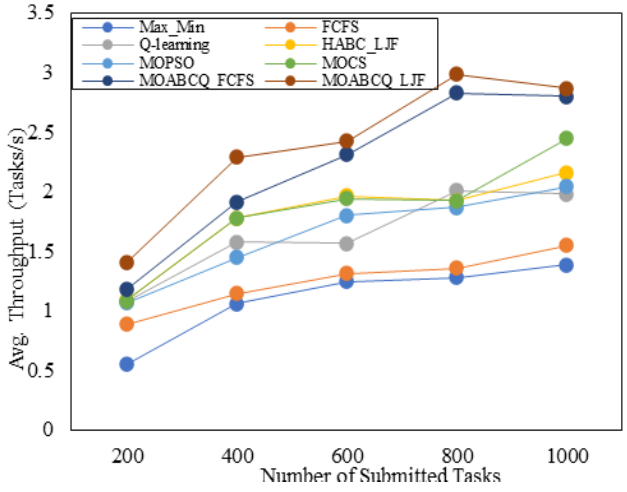
After using the GoCJ and Synthetic workload datasets to test throughput, we found that MOABCQ had better performance than the other algorithms. The results are shown in Fig. 4(b) and 4(c). When MOABCQ_LJF and MOABCQ_FCFS were considered, MOABCQ_LJF provides throughput approximately 6.14% more than MOABCQ_FCFS on average. We can conclude from the overall throughput test experiments that MOABCQ_LJF has the potential to allocate the tasks to appropriate resources in the same direction as the testing result in the first section.

The third section presents a comparison of the efficiency of the proposed method in terms of Average Resource Utilization Ratio (ARUR), which is another important condition for task scheduling in the system. In this experiment, we used 1000 tasks, 100 machines of VMs, and 3 dataset tests. The experimental results are shown in Table 3.

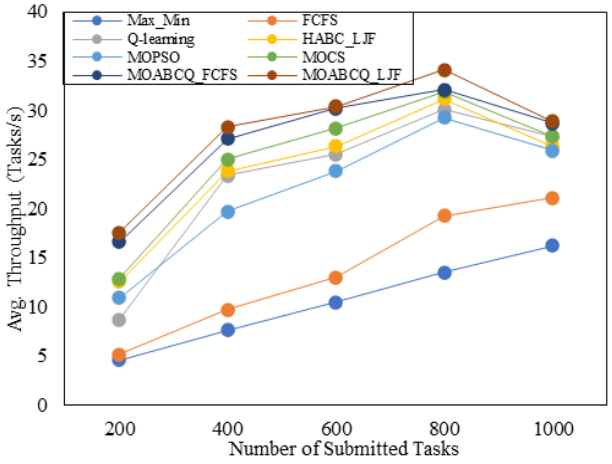
We found that MOABCQ gave higher ARUR values than the other methods. To consider in depth all 3 datasets, the algorithms used to perform in this experiment that gave the similarity of ARUR results were found to be Q-learning, HABC_LJF, MOPSO, MOCS, MOABCQ_FCFS, and MOABCQ_LJF, unlike Max-Min and FCFS. When testing with the random dataset, MOABCQ_LJF provided larger ARUR values of 6.15%, 14.01%, 18.72%, 20.94%, and 21.53% compared to MOABCQ_FCFS, MOCS, Q-learning, HABC_LJF, and MOPSO, respectively.



(a) Throughput comparison using the random dataset



(b) Throughput comparison using the GoCJ dataset



(c) Throughput comparison using the Synthetic workload dataset

FIGURE 4. Comparison of the performance in terms of throughput on various datasets.

When using the GoCJ dataset, MOABCQ_LJF gave greater ARUR values then the others at 6.31%, 11.34%, 12.63%, 13.61%, and 22.99% when compared to MOABCQ_FCFS, MOCS, HABC_LJF, MOPSO, and Q-learning. After experimenting with the Synthetic workload dataset, MOABCQ_LJF gave higher ARUR values than the MOABCQ_FCFS, MOCS, Q-learning, HABC_LJF, and MOPSO methods at 0.62%, 4.99%, 5.74%, 9.74%, and 9.86%, respectively.

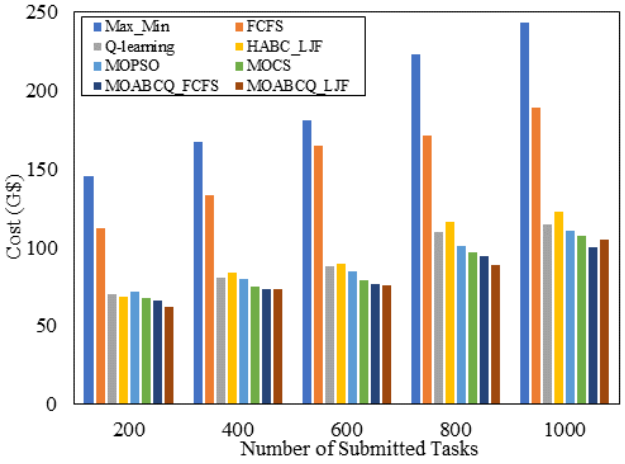
The testing with all three datasets revealed that MOABCQ_LJF provides the highest ARUR value compared to the other methods. Thus, we can conclude that the MOABCQ_LJF method can efficiently schedule tasks in the system and can help equally distribute tasks across available resources, which can help the system stay in balance mode.

The fourth section presents a comparison of the performance of the proposed method in terms of the degree of imbalance (DI) to assess the load balancing of the system. The experiments were conducted with the same datasets as in the previous experiment sections. These experiments were tested on 100 VMs and 200, 400, 600, 800, and 1000 tasks. The proposed method (MOABCQ) was compared with Max-Min, FCFS, Q-learning, HABC_LJF, MOPSO, and MOCS. The results are shown in Table 4. When testing on a random dataset and GoCJ dataset, the DI values of MOABCQ_LJF were the lowest, indicating that MOABCQ_LJF can distribute tasks more equally to existing resources in the system than the other methods. However, when using the Synthetic workload dataset, we found that in the case of setting tasks in the system equal to 200, 600, 800 and 1000 tasks, MOABCQ_LJF gave a lower DI value than the other methods. Unless testing with 400 tasks, MOABCQ_FCFS had the lowest DI value. When compared to MOABCQ_LJF, MOABCQ_FCFS can distribute tasks better than MOABCQ_LJF at 4.37%.

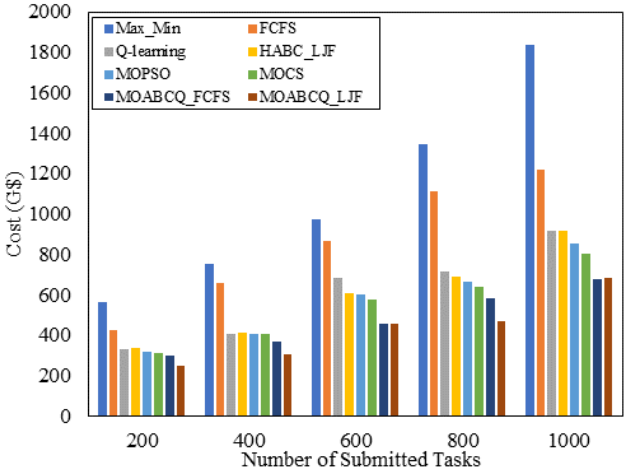
According to the testing with all three datasets, the MOABCQ method was found to be able to equally distribute the task to the available resources in the system, which resulted in a low DI value. If we consider the proposed method in depth, it reveals that MOABCQ_LJF performed more efficiently than the other compared methods. However, it depends on the dataset being tested.

The final section presents a comparison of the performance of the proposed method from a cost perspective to assess the costs or overheads when accessing cloud computing by executing 3 datasets. The experimental results are shown in Fig. 5. Considering Fig. 5(a), using the random dataset, MOABCQ was found to be able to reduce cost more than the MOCS, MOPSO, Q-learning, HABC_LJF, FCFS, and Max-Min. When MOABCQ_FCFS was compared with MOABCQ_LJF, it indicated that MOABCQ_LJF has lower cost than MOABCQ_FCFS by approximately 3.38%. However, with 1000 tasks, the MOABCQ_LJF algorithm costs 4.88% more than the MOABCQ_FCFS algorithm.

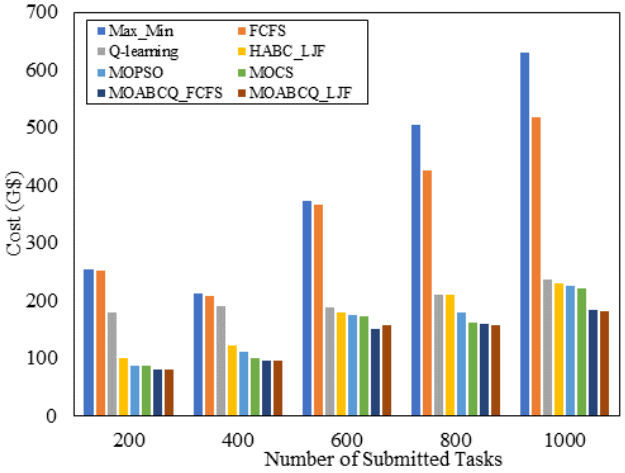
When testing with the GoCJ dataset, the results in Fig. 5(b) show that MOABCQ was able to reduce costs more than



(a) Cost comparison using the random dataset



(b) Cost comparison using the GoCJ dataset



(c) Cost comparison using the Synthetic workload dataset

FIGURE 5. Comparison of the performance in terms of cost on various dataset.

the other methods in the same way as when using the random dataset. When comparing MOABCQ_FCFS with MOABCQ_LJF and testing on 200, 400, and 800 tasks,

TABLE 4. Comparison of the performance in terms of DI on various datasets.

Dataset	Task	Task Scheduling Approach							
		Max-Min	FCFS	Q-learning	HABC_LJF	MOPSO	MOCS	MOABCQ_FCFS	MOABCQ_LJF
Random	200	1.285	0.577	0.442	0.462	0.334	0.280	0.264	**0.200**
	400	0.902	0.425	0.285	0.241	0.309	0.177	0.177	**0.166**
	600	0.916	0.685	0.585	0.378	0.555	0.203	0.134	**0.116**
	800	1.138	0.660	0.237	0.157	0.573	0.177	0.137	**0.115**
	1000	1.330	0.526	0.297	0.175	0.221	0.159	0.097	**0.093**
GoCJ	200	1.023	0.829	0.311	0.430	0.450	0.250	0.270	**0.132**
	400	0.997	0.944	0.272	0.283	0.278	0.204	0.146	**0.123**
	600	1.114	0.652	0.299	0.291	0.299	0.281	0.276	**0.176**
	800	1.234	0.544	0.478	0.294	0.499	0.287	0.387	**0.275**
	1000	1.112	0.912	0.355	0.373	0.387	0.363	0.314	**0.251**
Synthetic workload	200	1.529	0.964	0.225	0.410	0.412	0.378	0.268	**0.187**
	400	1.314	0.905	0.342	0.463	0.472	0.315	**0.293**	0.305
	600	1.130	0.766	0.302	0.734	0.742	0.301	0.418	**0.214**
	800	1.147	0.670	0.295	0.368	0.413	0.274	0.141	**0.117**
	1000	0.834	0.679	0.211	0.235	0.337	0.210	0.209	**0.122**

we found that the MOABCQ_LJF algorithm has a lower cost than the MOABCQ_FCFS algorithm at approximately 20.79%. In addition, when testing with 600 and 1000 tasks, MOABCQ_FCFS was found to have a higher cost than the MOABCQ_LJF method by 0.48% on average.

When testing with the Synthetic workload dataset, the results in Fig. 5(c) show that the MOABCQ method has a lower cost than the other comparison methods in the same way as when testing with the previous two datasets. After comparing MOABCQ_FCFS and MOABCQ_LJF, in the case of setting tasks in the system equal to 200, 600, 800, and 1000 tasks, the MOABCQ_LJF algorithm was found to have lower cost than MOABCQ_FCFS, except for 400 tasks, MOABCQ_FCFS has lower cost than MOABCQ_LJF method at 1.90%.

According to the testing with all three datasets, the proposed method of MOABCQ was able to reduce costs more than the other comparison methods. However, when comparing MOABCQ_LJF and MOABCQ_FCFS, we found that MOABCQ_LJF can be more appropriately used for task scheduling with existing resources in the system than MOABCQ_FCFS. Considering the difference in the percentage between the two methods, MOABCQ_LJF has a smaller percentage. However, it depends on the dataset to be tested.

E. THE TIME COMPLEXITY OF MOABCQ METHOD

The time complexity of MOABCQ method can be calculated as: in ABC, an initial population of n is given and the bee is classified into Employed bees and Onlooker bee. Therefore, the number of iterations to find suitable VMs in the cloud is n and the number of updated data in the Q-table is n as well. As a result, the time complexity of MOABCQ is $O(n)$. If ABC repeats this step k times, the time complexity is equal to $k \times O(n)$. Since k is a constant, the total time complexity of MOABCQ is equal to $O(n)$.

VI. CONCLUSION

In this article, we propose the multi-objective optimization scheduling method in heterogeneous cloud computing using the MOABCQ method. This method considered the selection of suitable VMs based on calculating the fitness of each VM. Heuristic approaches which are FCFS and LJF were also included. The experiments were conducted with various datasets to observe the performance of the proposed algorithms. The proposed method helps load balancing tasks with existing resources in the system and also improves makespan reduction, DI reduction, cost reduction, and throughput and ARUR increases when compared to the Max-Min, FCFS, Q-learning, HABC_LJF, MOPSO, and MOCS algorithms. The experimental results indicated that the proposed method outperformed the others. However, we cannot guarantee that the MOABCQ_LJF algorithm is optimal. Nevertheless, the performance of the system cannot be optimized in every test dataset.

Task scheduling in a multi-cloud, fog cloud, or edge cloud environment can be challenging and interesting work in the future. We propose a scheduling arrangement method in different environments. Other machine learning algorithms may also be applied further. In addition, the proposed method can also be tested in a real-world environment to observe the performance of the MOABCQ method.

REFERENCES

[1] T. S. George and V. P. S. Kumar, "Multicloud computing for on-demand resource provisioning using clustering," in *Proc. 3rd Int. Conf. Sustain. Energy Intell. Syst. (SEISCON)*, 2012, pp. 435–440.

[2] S. Yang, L. Pan, Q. Wang, S. Liu, and S. Zhang, "Subscription or pay-as-you-go: Optimally purchasing IaaS instances in public clouds," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2018, pp. 219–226.

[3] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang, "Quality-of-service in cloud computing: Modeling techniques and their applications," *J. Internet Services Appl.*, vol. 5, pp. 1–17, Dec. 2014.

[4] K. Psychas and J. Ghaderi, "On non-preemptive VM scheduling in the cloud," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, no. 2, pp. 1–29, Dec. 2017, doi: 10.1145/3154493.

[5] S. Crago, K. Dunn, P. Eads, L. Hochstein, D. Kang, M. Kang, D. Modium, K. Singh, J. Suh, and J. Walters, "Heterogeneous cloud computing," in *Proc. IEEE Int. Conf. Clust. Comput.*, Austin, TX, USA, Feb. 2011, pp. 378–385, doi: 10.1109/CLUSTER.2011.49.

[6] J. W. M. Bush, B. A. Thurber, and F. Blanchette, "Particle clouds in homogeneous and stratified environments," *J. Fluid Mech.*, vol. 489, pp. 29–54, Jul. 2003, doi: 10.1017/S0022112003005160.

[7] R. Messier, "Virtual servers and platform as a service," in *Proc. Collaboration Cloud Comput. Secur., Social Media, Unified Commun.*, 2014, pp. 77–91, doi: 10.1016/B978-0-12-417040-7.00005-8.

[8] O. Alsaryrah, I. Mashal, and T.-Y. Chung, "Bi-objective optimization for energy aware Internet of Things service composition," *IEEE Access*, vol. 6, pp. 26809–26819, 2018, doi: 10.1109/ACCESS.2018.2836334.

[9] L. Liu, M. Zhang, R. Buyya, and Q. Fan, "Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing," *Concurr. Comput. Pract. Exp.*, vol. 29, no. 5, p. e3942, 2017, doi: 10.1002/cpe.3942.

[10] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *J. Supercomput.*, vol. 63, no. 1, pp. 256–293, 2013, doi: 10.1007/s11227-011-0578-4.

[11] D. Yagyasen, M. Darbari, P. K. Shukla, and V. K. Singh, "Diversity and convergence issues in evolutionary multiobjective optimization: Application to agriculture science," *IERI Proc.*, vol. 5, pp. 81–86, Oct. 2013, doi: 10.1016/j.ieri.2013.11.074.

[12] F. Luo, Y. Yuan, W. Ding, and H. Lu, "An improved particle swarm optimization algorithm based on adaptive weight for task scheduling in cloud computing," in *Proc. 2nd Int. Conf. Comput. S. App. Eng.*, 2018, pp. 1–5, doi: 10.1145/3207677.3278089.

[13] I. Alharkan, M. Saleh, M. A. Ghaleb, H. Kaid, A. Farhan, and A. Almarfadi, "Tabu search and particle swarm optimization algorithms for two identical parallel machines scheduling problem with a single server," *J. King Saud Univ.-Eng. Sci.*, vol. 32, no. 5, pp. 330–338, Jul. 2020, doi: 10.1016/j.jksues.2019.03.006.

[14] S. Basu, M. Karuppiah, K. Selvakumar, and K. Li, "An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment," *Future Gener. Comput. Syst.*, vol. 88, pp. 254–261, Nov. 2018, doi: 10.1016/j.future.2018.05.056.

[15] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization," in *Proc. 1st IEEE Conf. Evol. Comput. World Congr. Comput. Intell.*, vol. 1, Jun. 1994, pp. 82–87, doi: 10.1109/ICEC.1994.350037.

[16] J. Knowles and D. Corne, "Approximating the nondominated front using the Pareto archived evolution strategy," *Evol. Comput.*, vol. 8, no. 2, pp. 149–172, Jan. 2000, doi: 10.1162/106365600568167.

[17] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," ERU, Kayseri, Turkey, Tech. Rep.-tr06, Oct. 2005.

[18] B. Akay and D. Karaboga, "A modified artificial bee colony algorithm for real-parameter optimization," *Inf. Sci.*, vol. 192, no. 1, pp. 120–142, Apr. 2012, doi: 10.1016/j.ins.2010.07.015.

[19] D. Karaboga and B. Gorkemli, "A combinatorial artificial bee colony algorithm for traveling salesman problem," in *Proc. Int. Symp. Innov. Intell. Syst. Appl.*, Jun. 2011, pp. 50–53.

[20] X. Li, D. Peng, B. Du, J. Guo, W. Xu, and K. Zhuang, "Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems," *Comput. Ind. Eng.*, vol. 113, pp. 10–26, Nov. 2017, doi: 10.1016/j.cie.2017.09.005.

[21] J. D. Ullman, "NP-complete scheduling problems," *J. Comput. Syst. Sci.*, vol. 10, no. 3, pp. 384–393, 1975.

[22] Y. Mao, X. Chen, and X. Li, "Max-min task scheduling algorithm for load balance in cloud computing," in *Proc. Int. Conf. Comput. Sci. Inf. Technol.*, 2014, pp. 457–465.

[23] T. Islam and M. S. Hasan, "A performance comparison of load balancing algorithms for cloud computing," in *Proc. Int. Conf. Frontiers Adv. Data Sci. (FADS)*, Oct. 2017, pp. 130–135.

[24] G. Patel, R. Mehta, and U. Bhoi, "Enhanced load balanced min-min algorithm for static meta task scheduling in cloud computing," *Proc. Comput. Sci.*, vol. 57, pp. 545–553, Jan. 2015, doi: 10.1016/j.procs.2015.07.385.

[25] H. Zhang, J. Shi, B. Deng, G. Jia, G. Han, and L. Shu, "MCTE: Minimizes task completion time and execution cost to optimize scheduling performance for smart grid cloud," *IEEE Access*, vol. 7, pp. 134793–134803, 2019, doi: 10.1109/ACCESS.2019.2942067.

[26] A. Hussain, M. Aleem, A. Khan, M. A. Iqbal, and M. A. Islam, "RALBA: A computation-aware load balancing scheduler for cloud computing," *Cluster Comput.*, vol. 21, no. 3, pp. 1667–1680, 2018, doi: 10.1007/s10586-018-2414-6.

[27] B. Mondal, K. Dasgupta, and P. Dutta, "Load balancing in cloud computing using stochastic hill climbing—A soft computing approach," *Proc. Technol.*, vol. 4, pp. 783–789, Jun. 2012, doi: 10.1016/j.protcy.2012.05.128.

[28] M. Adhikari and S. Koley, "Cloud computing: A multi-workflow scheduling algorithm with dynamic reusability," *Arabian J. Sci. Eng.*, vol. 43, no. 2, pp. 645–660, Feb. 2018, doi: 10.1007/s13369-017-2739-0.

[29] B. Shrimali and H. Patel, "Multi-objective optimization oriented policy for performance and energy efficient resource allocation in cloud environment," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 32, no. 7, pp. 860–869, Sep. 2020, doi: 10.1016/j.jksuci.2017.12.001.

[30] C.-W. Tsai and J. J. P. C. Rodrigues, "Metaheuristic scheduling for cloud: A survey," *IEEE Syst. J.*, vol. 8, no. 1, pp. 279–291, Mar. 2014, doi: 10.1109/JSYST.2013.2256731.

[31] M. Kalra and S. Singh, "A review of metaheuristic scheduling techniques in cloud computing," *Egyptian Informat. J.*, vol. 16, no. 3, pp. 275–295, 2015, doi: 10.1016/j.eij.2015.07.001.

[32] F. Ramezani, J. Lu, J. Taheri, and F. K. Hussain, "Evolutionary algorithm-based multi-objective task scheduling optimization model in cloud environments," *World Wide Web*, vol. 18, no. 6, pp. 1737–1757, 2015, doi: 10.1007/s11280-015-0335-3.

[33] L. Zuo, L. Shu, S. Dong, C. Zhu, and T. Hara, "A multi-objective optimization scheduling method based on the ant colony algorithm in cloud computing," *IEEE Access*, vol. 3, pp. 2687–2699, 2015, doi: 10.1109/ACCESS.2015.2508940.

[34] X. Guo, "Multi-objective task scheduling optimization in cloud computing based on fuzzy self-defense algorithm," *Alexandria Eng. J.*, vol. 60, no. 6, pp. 5603–5609, Dec. 2021.

[35] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, "Cloud task scheduling based on ant colony optimization," in *Proc. 8th Int. Conf. Comput. Eng. Syst. (ICCES)*, Nov. 2013, pp. 64–69.

[36] Z. Chen, K. Du, Z. Zhan, and J. Zhang, "Deadline constrained cloud computing resources scheduling for cost optimization based on dynamic objective genetic algorithm," in *Proc. CEC*, 2015, pp. 708–714, doi: 10.1109/CEC.2015.7256960.

[37] Z. Amini, M. Maeen, and M. R. Jahangir, "Providing a load balancing method based on dragonfly optimization algorithm for resource allocation in cloud computing," *Int. J. Netw. Distrib. Comput.*, vol. 6, no. 1, pp. 35–42, 2018, doi: 10.2991/ijndc.2018.6.1.4.

[38] M. S. Sanaj and P. M. Joe Prathap, "An efficient approach to the map-reduce framework and genetic algorithm based whale optimization algorithm for task scheduling in cloud computing environment," *Mater. Today, Process.*, vol. 37, pp. 3199–3208, Oct. 2021.

[39] F. Farahnakian, P. Liljeberg, and J. Plosila, "Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning," in *Proc. 22nd Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, Turin, Italy, 2014, pp. 500–507.

[40] S. Ismaeel, R. Karim, and A. Miri, "Proactive dynamic virtual-machine consolidation for energy conservation in cloud data centres," *J. Cloud Comput.*, vol. 7, no. 1, pp. 1–28, Dec. 2018, doi: 10.1186/s13677-018-0111-x.

[41] U. Rugwiro, C. Gu, and W. Ding, "Task scheduling and resource allocation based on ant-colony optimization and deep reinforcement learning," *J. Internet Technol.*, vol. 20, no. 5, pp. 1463–1475, 2019, doi: 10.3966/160792642019092005013.

[42] J.-Q. Li and Y.-Q. Han, "A hybrid multi-objective artificial bee colony algorithm for flexible task scheduling problems in cloud computing system," *Cluster Comput.*, vol. 23, no. 4, pp. 2483–2499, Dec. 2020, doi: 10.1007/s10586-019-03022-z.

[43] B. Kruekaew and W. Kimpan, "Enhancing of artificial bee colony algorithm for virtual machine scheduling and load balancing problem in cloud computing," *Int. J. Comput. Intell. Syst.*, vol. 13, no. 1, pp. 496–510, 2020, doi: 10.2991/ijcis.d.200410.002.

[44] G.-N. Gan, T.-L. Huang, and S. Gao, "Genetic simulated annealing algorithm for task scheduling based on cloud computing environment," in *Proc. Int. Conf. Intell. Comput. Integr. Syst.*, Oct. 2010, pp. 60–63.

[45] D. Alsadie, "A metaheuristic framework for dynamic virtual machine allocation with optimized task scheduling in cloud data centers," *IEEE Access*, vol. 9, pp. 74218–74233, 2021, doi: 10.1109/ACCESS.2021.3077901.

[46] H. He, G. Xu, S. Pang, and Z. Zhao, "AMTS: Adaptive multi-objective task scheduling strategy in cloud computing," *China Commun.*, vol. 13, no. 4, pp. 162–171, Apr. 2016, doi: 10.1109/CC.2016.7464133.
[47] R. Jena, "Task scheduling in cloud environment: A multi-objective ABC framework," *J. Inf. Optim. Sci.*, vol. 38, pp. 1–19, Jan. 2017, doi: 02522667.2016.1250460.
[48] A. Kumar and M. Venkatesan, "Multi-objective task scheduling using hybrid genetic-ant colony optimization algorithm in cloud environment," *Wireless Pers. Commun.*, vol. 107, pp. 1835–1848, 2019, doi: 10.1007/s11277-019-06360-8.
[49] G. N. Reddy and S. P. Kumar, "Multi objective task scheduling algorithm for cloud computing using whale optimization technique," in *Proc. Int. Conf. Next Gener. Comput., Technol.* Singapore: Springer, 2017, pp. 286–297.
[50] S. H. H. Madni, M. S. A. Latiff, J. Ali, and S. M. Abdulhamid, "Multi-objective-oriented cuckoo search optimization-based resource scheduling algorithm for clouds," *Arabian J. Sci. Eng.*, vol. 44, no. 4, pp. 3585–3602, 2019, doi: 10.1007/s13369-018-3602-7.
[51] S. Pang, W. Li, H. He, Z. Shan, and X. Wang, "An EDA-GA hybrid algorithm for multi-objective task scheduling in cloud computing," in *IEEE Access*, vol. 7, pp. 146379–146389, 2019, doi: 10.1109/ACCESS.2019.2946216.
[52] P. Neelima and A. R. M. Reddy, "An efficient load balancing system using adaptive dragonfly algorithm in cloud computing," *Cluster Comput.*, vol. 23, pp. 2891–2899, 2020, doi: 10.1007/s10586-020-03054-w.
[53] M. Gamal, R. Rizk, H. Mahdi, and B. E. Elnaghi, "Osmotic bio-inspired load balancing algorithm in cloud computing," *IEEE Access*, vol. 7, pp. 42735–42744, 2019, doi: 10.1109/ACCESS.2019.2907615.
[54] U. A. Butt, M. Mehmood, S. B. H. Shah, R. Amin, M. W. Shaukat, S. M. Raza, D. Y. Suh, and M. J. Piran, "A review of machine learning algorithms for cloud computing security," *Electronics*, vol. 9, no. 9, p. 1379, Aug. 2020, doi: 10.3390/electronics9091379.
[55] L. Caviglione, M. Gaggero, M. Paolucci, and R. Ronco, "Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters," *Soft. Comput.*, vol. 25, pp. 12569–12588, Oct. 2021, doi: 10.1007/s00500-020-05462-x.
[56] U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *J. King Saud Univ.-Comput. Inf. Sci.*, early access. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157819309267, doi: 10.1016/j.jksuci.2020.01.012.
[57] T. Thein, M. M. Myo, S. Parvin, and A. Gawanmeh, "Reinforcement learning based methodology for energy-efficient resource allocation in cloud data centers," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 32, no. 10, pp. 1127–1139, Dec. 2020, doi: 10.1016/j.jksuci.2018.11.005.
[58] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "Analysis and lessons from a publicly available Google cluster trace," EECS Dep., RAD Lab, Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2010-95, Jun. 2010.
[59] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: New insights," *Struct. Multidisciplinary Optim.*, vol. 41, no. 6, pp. 853–862, Jun. 2010, doi: 10.1007/s00158-009-0460-7.
[60] I. Y. Kim and O. L. de Weck, "Adaptive weighted-sum method for bi-objective optimization: Pareto front generation," *Struct. Multidisciplinary Optim.*, vol. 29, no. 2, pp. 149–158, 2005, doi: 10.1007/s00158-004-0465-1.
[61] A. Abdelsamea, A. A. El-Moursy, E. E. Hemayed, and H. Eldeeb, "Virtual machine consolidation enhancement using hybrid regression algorithms," *Egyptian Inform. J.*, vol. 18, no. 3, pp. 161–170, Nov. 2017, doi: 10.1016/j.eij.2016.12.002.
[62] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[63] S. Fairee, S. Prom-On, and B. Sirinaovakul, "Reinforcement learning for solution updating in artificial bee colony," *PLoS ONE*, vol. 13, no. 7, Jul. 2018, Art. no. e0200738, doi: 10.1371/journal.pone.0200738.
[64] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm," *J. Global Optim.*, vol. 39, pp. 459–471, Nov. 2007, doi: 10.1007/s10898-007-9149-x.
[65] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exper.*, vol. 41, no. 1, pp. 23–50, 2011, doi: 10.1002/spe.995.
[66] A. Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures," *Data*, vol. 3, no. 4, p. 38, 2018, doi: 10.3390/data3040038.
[67] H. Saleh, H. Nashaat, W. Saber, and H. M. Harb, "IPSO task scheduling algorithm for large scale data in cloud computing environment," *IEEE Access*, vol. 7, pp. 5412–5420, 2018.

BOONHATAI KRUEKAEW received the B.Sc. degree in computer science from the Prince of Songkla University, Hat Yai, Songkhla, Thailand, and the M.Sc. degree in computer science from the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, where she is currently pursuing the Ph.D. degree. Her research interests include cloud computing, algorithm, artificial intelligence, and swarm intelligence.

WARANGKHANA KIMPAN (Member, IEEE) received the Ph.D. degree in system information engineering from Kagoshima University, Japan. She is currently an Assistant Professor with the Department of Computer Science, School of Science, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. Her main research interests include swarm intelligence, biomedical engineering, big data, data science and analytics, cloud computing, and the Internet of Things.