Received: February 16, 2023. Revised: March 10, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Dynamic Cost-Optimized Resources Management and Task Scheduling with Deadline Constraint for Mobile Crowd Sensing Environment

Abbas M. Ali Al-muqarm[1,2]* **Naseer Ali Hussien[3]**

[1]*Department of Computer Science, Faculty of Computer Science and Mathematics, University of Kufa, Iraq*
[2]*Computer Technical Engineering Department, The Islamic University, Najaf 54001, Iraq*
[3]*Alayen University, Iraq*
* Corresponding author's Email: abbasm.almuqarm@student.uokufa.edu.iq

Abstract: The unpredictable and huge data generation nowadays by smart devices from IoT and mobile crowd sensing (MCS) applications like sensors, and smartphones requires processing power and rapid responses in a specific time frame (deadline), and user cost, i.e., budget. Cloud provides these capabilities to serve organizations and customers, but when using cloud appear some limitations, the most important is task scheduling. Nevertheless, the most of earlier scheduling algorithms have focused on only one evaluation parameter such as makespan or cost and so on, as the scheduling process goal. Optimizing a single metric may not ensure an improvement in the cloud performance in general, so the need may arise for algorithms that concentrate more broadly on enhancing more than one parameter. Therefore, this paper presents a deadline-budget multi-objective dynamic scheduling scheme (DB-MODS) to execute user tasks on VMs within QoS limits in order to finish the task within the deadline and budget if possible. DB-MODS using K-Means based on task length and deadline for clustering incoming tasks into groups and categories VMs based on capacity thresholds. In addition, the task belonging to every cluster is assigned to a suitable VM in VMs groups. based on objective functions, which can be one for user request depending on (deadline and budget) and the second is for system depending on (execution time and cost). This paper used CloudSim plus to simulate and evaluate our approach. The DB-MODS approach performance was compared to scheduling algorithms from the previous literature. Both random and Google cloud jobs (GOCJ) workloads are used to evaluate the efficiency of DB-MODS. The results show that DB-MODS outperforms the other algorithms by minimizing makespan by 41%, energy by 46%, and increasing success ratio by 35%, in comparison to existing load balancing and scheduling methods: EEVS, LAS, Greedy-R, DTS, Random, and Greedy-P in the first scenario. Minimizing makespan by 46%, cost by 44%, and maximize throughput by 80% when compared with MOCS, Max-Min, HABC-LJF, FCFS, MOPSO, Q-learning, MOABCQ-FCFS, and MOABCQ-LJF in a second scenario, and minimizing makespan by 46%, and increase throughput by 39% when compared with HESGA, GA, and ACO in a third scenario in average. In addition to the possibility of applying and employing DB-MODS for scheduling deadline-critical tasks in a heterogeneous cloud system.

Keywords: Cloud computing, Resources management, Task scheduling, IoT, Mobile crowd sensing.

1. Introduction

Cloud computing is a developing computational method that relies on virtualization hardware to distribute resources dynamically in response to user requests made over the internet [1]. This virtualization of cloud computing improves accessibility while lowering maintenance costs. Task scheduling becomes a key problem in the cloud when the user requests service for efficient resource allocation. The purpose of scheduling is to establish the best relationship between tasks and machine resources. The scheduling process is simple when there are few user tasks and resources. The scheduling becomes complicated if a user sends out several requests to get the appropriate level of service [2, 3].

In the cloud context, task schedule has recently

been a fascinating topic that has garnered a lot of concentration in the research literature. Task scheduling is allocating tasks to the resources with more availability based on the specifications and requirements of the tasks. Cloud broker receives the tasks of the users and assigns them to the available resources, taking into account the characteristics of the task and the resources. These resources must be used correctly and perfectly in order to allocate the best suitable physical resources to the task to achieve good quality of service (QoS) and an optimal allocation of resources. To address the challenge of user satisfaction and provider revenue [4]. Before using any services or renting out resources, cloud users always negotiate with cloud service provider (CSP) to ensure service-level-agreements (SLAs). In a computing paradigm, a user only pays for the services and resources they really use. The two most crucial aspects that affect a consumer are turnaround time and budget. Consequently, deadline and budget are the trade-off issues for task scheduling. Task constraints like deadlines and budgets are used to assign user tasks to cloud resources that differ in their characteristics, based on cost and time. A provider can reduce operating expenses and provide excellent QoS by guaranteeing that each user has fair access to resources shared by all users [5].

Scheduling heterogeneous resources in cloud relies on some characteristics such as availability of resources, resource utilization, workload length, resource capacity, and cost. like differing processor speeds and communication between CPUs. SLA negotiation as well controls the uses of resources and scheduling. As a result, must focus on the parameters mentioned above while scheduling requested tasks on resources to ensure user satisfaction [6]. There are several scheduling algorithms such as shortest job first (SJF), first come first serve (FCFS), min-min, max-min, and round robin (RR), there are existing, nevertheless, these are not seen as being significantly superior solutions to scheduling issues in cloud environment. To address the scheduling issue, the scientific community has given a variety of solutions [7]. Traditionally, resource management has relied on static policies, but these policies have limitations in dynamic situations. As a result, CSP have turned to data-driven, and machine learning. Machine learning is employed for various resource management functions to address diverse resource management tasks, involving estimating workloads, task scheduling, optimizing resources, VM consolidation and improving energy efficiency, among others [8]. The processes of clustering and task scheduling are commonly concentrated in federated clouds for task assignment based on the resources [9]. Decision making with multi criteria is a branch of operations research that studies issues with several criteria, like priority-based workload scheduling. This way of decision method provides important support for decision-makers (DMs) by specifying the problem, also formal it, and making helpful suggestions [4]. Dynamic scheduling methods have more flexible than the earlier defined types. Dynamic schedulers possess the capability to monitor, and estimate, in addition to updating the load on the VMs at the time of execution as the tasks are being carried out in either a proactive or reactive manner [10]. These schedulers enable the priority and migration of the tasks that have already been allocated. Additionally, dynamic scheduling methods typically have the capability to create new VMs, remove existing ones, and migrate VMs while they are running [11]. However, several of the current dynamic scheduling methods continue to use a temporal-batch of incoming tasks that were received during an interval of time. Most dynamic algorithms now in use have problems with Inadequate resource utilization, an imbalanced load, and a high rate of rejection for tasks with a time constraint (i.e. deadline) [12].

In this paper, a new approach named DB-MODS is proposed to address load balancing and task scheduling issues. DB-MODS task scheduling method is based on k-means algorithm for task clustering and thresholds for classifying virtual machine (VMs). The objective function of proposed is computed the execution time and cost of every task on all VMs and return the minimum value as the fitness value (F) and computed the objective function for user parameters to choose a suitable VM to process a given task.

The primary contributions of this paper are outlined as follows:

- Introduce a (multi-objective optimization) DB-MODS approach for deadline and budget sensitive tasks scheduling problem in a cloud for IoT/MCS environment.
- To proactively and intelligently find the optimum VMs for task allocation while taking time and money restrictions into account.
- The proposed method focuses on two competing objectives, namely the shortest completion time within a deadline and the lowest cost within a budget constraint.
- The scheduling problem is defined as a multi-objective issue that aims to minimize of makespan, energy consumption, and cost.
- To validate the results of the proposed DB-MODS against methods from the literature, a statistical

test (i.e. t-test) was used. The t-test is employed using a significance test.

- A large number of tests were conducted, to show the superiority of DB-MODS over peers. The comparisons were done by measuring makespan, throughput, success ratio, cost, and energy consumption by using CloudSim plus.

The rest of the paper is organized as in the following: section 2, presents the works related to resource management and scheduling problem in a cloud system. Section 3 presents task scheduling and resource management. Section 4 introduces problem modeling. The proposed approach of task scheduling based on K-means clustering is introduced in section 5. Section 6 introduces the proposed load-balancing and scheduling algorithm DB-MODS. Section 7 introduces performance metrics. Section 8 introduces implementation and experimentation results. Section 9 introduces the conclusion and future works.

2. Related work

This section of this paper presents various works related to resource management and task scheduling issues in different environments like IoT and cloud. Some benefits and drawbacks of each related study are discussed and analyzed.

Task scheduling is an NP-hard, and considered a complex aspect when designing cloud computing systems. It plays a crucial role in achieving high performance and maximizing resource utilization by utilizing the best features from available resources. This paper's key contribution is to present a novel approach called DRRHA that focuses on the classic RR algorithm's drawbacks via optimizing the metrics of performance by reducing the average response time, waiting time and turn-around time. But it contains a lot of calculations, in addition to their use of the FCFS and SJR algorithms [13].

In this paper, the authors introduced a new framework called learning automata scheduling (LAS) which is an adaptive decision-making unit to deal with tasks sensitive to deadline in the cloud environment. They formulated the issue of scheduling tasks as a bi-objective to decrease makespan and energy consumption, but they did not use a real dataset and did not consider user budget [14].

In [15] an energy-efficient-scheduling algorithm (EEVS) was proposed, in the cloud by considering the deadline and supporting dynamic voltage and frequency scaling (DVFS) well. The EEVS involves dividing the process into distinct scheduling periods. During each period, virtual machines are assigned to appropriate physical machines, and the active cores run at the most efficient frequency. To achieve even greater energy savings, the cloud needs to be reconfigured after each period to consolidate computing resources. Despite its ability to process a greater number of virtual machines while consuming less energy, the EEVS approach has some limitations. The authors did not take into account the performance and power penalties associated with processor status transitions and VM migrations. These assumptions may not hold in real-world cloud environments, which underscores the need for practical testing using actual workloads.

In [16] The authors introduced a novel method called MOABCQ that applies optimization as multi-objective for scheduling issue. This approach employs artificial bee-colony (ABC) algorithm, which has been enhanced by incorporating Q-learning, a reinforcement learning mechanism, that enables the ABC to achieve faster performance. The primary goal of MOABCQ is to decrease both the makespan, and cost associated with the scheduling problem. This approach combined with the FCFS named "MOABCQ_FCFS" and largest-job first (LJF) named "MOABCQ_LJF". However, the authors cannot provide a guarantee that the MOABCQ_LJF algorithm achieves optimal results and they don't consider the user deadline and budget constraints.

This paper proposed task scheduling with an ABC named (HABC) which is a combination of ABC with heuristic algorithms (FCFS, SJF, and LJF) to improve VMs scheduling in cloud by minimizing makespan and balancing the loads. The result shows that the HABC with LJF (HABC_LJF) offers the best performance in scheduling and load balancing compared to ACO, PSO, and IPSO. So, we compared our work with HABC_LJF. The authors did not take into account the use of real-world datasets or the ability of the HABC algorithm to handle the scheduling of multiple jobs with varying priorities [17].

This paper proposed an approach based on a two-stage strategy called dynamic cloud task scheduling (DTS). First, the task classification process involves using a bayes classifier that leverages historical scheduling data to categorize tasks. Various types of VMs are consequently formed. Then, tasks are matched with specific VMs dynamically. Tasks are assigned to compatible VMs dynamically. The results show that achieving effectively improves the load balancing of the cloud. The authors don't consider energy consumption and do not test on real dataset [18].

This article introduced a new approach called hybrid electro search with a genetic algorithm (HESGA) to enhance the performance of task scheduling by considering various parameters such as makespan, load balancing, resource utilization, and multi-cloud costs. By combining the benefits of a genetic algorithm and an electro-search algorithm. GA to find the top local optimal

solution, whereas ESA provides the best global optimal solutions. But the authors do not test by using a real dataset and did not consider energy as a parameter [19].

The authors of this paper suggested a scheduling technique CBTS by using k-means algorithm and taking only the length of task and VM capacity. In this case, tasks are clustered only according to their length, whereas VMs are grouped depending on their processing capabilities. The proposed enhanced the performance of cloud data centers by reducing execution time and makespan. But, a task may be restricted by deadline. As a result, the deadline may be a significant additional property of the tasks to include when creating task clusters [7].

An in-depth examination of associated works reveals that the most of the current task scheduling algorithms are tested using tiny datasets, that are insufficient to demonstrate their performance and work in static rather than dynamic environment, scheduling the tasks based on user preferences without considering the system capabilities. In addition, cluster tasks are based on length only. So, these studies still there is a need for new solutions. To overcome these limitations a new approach has been proposed, we have set a relationship between the budget and the deadline required by the user, as well as the time of implementation and costs by the system to create a balance between what the user requests and what can be provided by the cloud by formulating a multi-objective function that was overlooked by previous research. Also, the length and deadline of the task it has a close relationship in the scheduling, however, most studies did not consider them. Therefore, we worked on setting a relationship between them by using a clustering algorithm, relying on these two parameters rather than a single perimeter.

3. Resource management and tasks scheduling

In processing large-scale data, applications and workloads are initially split up into smaller units known as tasks. Scheduler allocate a collection of tasks with their characteristics to a group of VMs nodes. The primary objective of scheduling is decision-making, and it is one of the problems that are classified under the NP-hard problem. Due to its dynamic nature, determining the best computing node for task scheduling is a big issue. Along with assigning physical resources to the tasks. Scheduler also has to fulfil the demands of the service-providers, and the customers. Clearly, its primary objectives are reducing waiting time for tasks, time for execution tasks, makespan, and improve

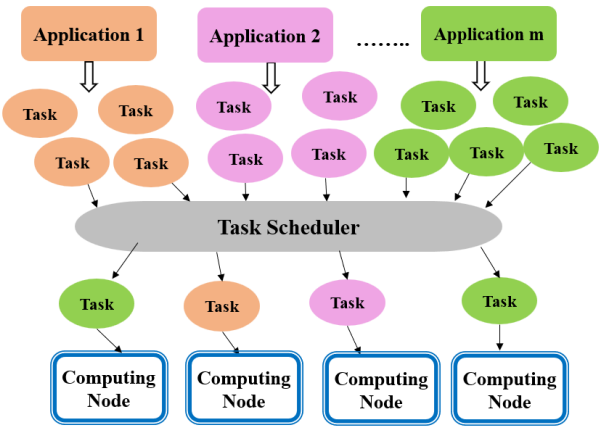


Figure. 1 Scheme of task scheduling

utilization of resources. Some of them concerned with QoS and satisfying the needs of consumers, while others are concerned with maximizing supplier profits [20].

Task scheduling methods place a strong emphasis on how to select resources that minimize the time of execution and waiting. Because virtualization is essential to cloud, scheduling approaches are typically developed in accordance with two tiers of the cloud system. first, at host level, where VMs are distributed according to a group of policies. The second is at the VM level, where allocation tasks to VMs are controlled and necessitate a particular set of policies. The benefits of task scheduling algorithms include improving QoS, increasing resource usage memory, CPU, network, etc., and improving performance, high system throughput, enhanced load balancing, a higher number of finished tasks, etc. [21].

Task scheduling is basically a key process that is managed by datacenter brokers. The broker is crucial in mapping the task-to-VMs It maintains track of the list of available VMs while preserving their QoS. When a user submits a request to a broker, it is the broker's duty to assign a suitable VM to the requested task in a way that doesn't violate the SLA or degrade the QoS. High QoS is ensured in a cloud computing environment by the high performance of VMs. When a low-performing VM is assigned to a high-performance task, the available resources are not fully utilized, which leads to poor performance and throughput, ultimately, a breach of SLA. Therefore, implementing extremely effective scheduling techniques at the broker level has become necessary, in addition to opening up new opportunities for research into task scheduling algorithms for cloud computing [22].

Two methods of the task scheduling process, are static or dynamic. In static method, whole tasks are initially statically assigned to VMs before they begin to execution. As a consequence, tasks given to any resource cannot be changed during run time. Because static task

scheduling techniques have less scheduling overhead, their performance for a single parameter may be superior to dynamic task scheduling algorithms. However, static algorithms still are incapable of dynamically adjusting to changing situations [23]. Fig. 1 shows the task scheduling system.

4. Problem modeling

Tasks scheduling in the could be efficient and achieve high performance if tasks are assigned to cloud VMs properly. This definition assumes that the cloud system is housed in a datacenter with a heterogeneous of servers, which hosts a number of VMs. Assume that there are n tasks, $T = \{T_1, T_2, T_3, \dots, T_n\}$, the tasks are access to the cloud from the IoT devices and MCS applications. The cloud system consists of cloud nodes, each of which has many characteristics, including various processor capacities, memory sizes, and communication links with varying bandwidths and storage capacities. Assume that a collection of N cloud nodes can be represented as follows:

$$N = \{N_1, N_2, N_3, \dots, N_m\}$$

where N_j is represents the jth processing of physical node. Every task T_i would be allocated to a single computing node N_j, and is expressed as T_i^j. Each computing node may be assigned a number of tasks, $N_j (j = 1,2,3, \dots, m)$ as in the expression:

$$N_j^T = \{T_x^j, T_y^j, \dots, T_z^j\}$$

In this proposed approach, the model contains a finite group of m heterogeneous VMs or multi-core computing nodes, with a variable capacity to run a particular task.

The time of execution task i on vm_j is ET_{ij} and can be computed as in the following equation:

$$ET_{ij} = \frac{L(task_i)}{N_{PE\,j} \times VM_{MIPS\,j}} \quad (1)$$

where $L(\text{task}_i)$ is a tasks length expressed in million -instructions (MI), $N_{PE\,j}$ is the number of processing elements of VM, and $VM_{MIPS\,j}$ refers to the speeds of the VM measured in million-instructions-per-second (MIPS).

The total execution time TET_{VM_j} of running collocation of tasks in VM_j node is computed as in Eq. (2):

$$TET_{VM_j} = \sum_{i=0}^{n} ET_{ij} \quad (2)$$

Assume that makespan represents the overall amount of time needed to perform all tasks in T. The makespan may be computed by the following equation:

$$Makespan = Max_{1 \le j \le m}[TET_{VM_j}\ (N_j)] \quad (3)$$

Let's define $\text{Cost}\left(T_i^j\right)$ as the quantity of money required to finish the task T_i^j in cloud node N_j, including the processing cost $C_p\left(T_i^j\right)$, cost of memory usage $C_m\left(T_i^j\right)$, and bandwidth usage cost $C_B\left(T_i^j\right) \cdot$ Calculate the $\text{Cost}\left(T_i^j\right)$ as in the following:

$$Cost\left(T_i^j\right) = C_p\left(T_i^j\right) + C_m\left(T_i^j\right) + C_B\left(T_i^j\right). \quad (4)$$

The above three costs can be defined as:

$$C_p\left(T_i^j\right) = Cost_{CPU-j} \times ET_{ij}\left(T_i^j\right), \quad (5)$$

$$C_m\left(T_i^j\right) = Cost_{M-j} \times Memory\left(T_i^j\right), \quad (6)$$

$$C_B\left(T_i^j\right) = Cost_{B-j} \times Bandwidth\left(T_i^j\right), \quad (7)$$

where $Cost_{CPU-j}$ is the cost of using CPU to execute task in node N_j within time, $Cost_{M-j}$ is the cost of use memory in N_j node , $\text{Memory}\left(T_i^j\right)$ is the amount of memory consumed by a task T_i in node j, $Cost_{B-j}$ is the bandwidth cost, and $\text{bandwidth}\left(T_i^j\right)$ is the value required to transfer task T_i to be executed in node N_j. The cost of running whole tasks in a cloud can be represented as follows:

$$Total_{Cost} = \sum_{T_i^j \in T^{node}} Cost\left(T_i^j\right). \quad (8)$$

We assume that there are N tasks, i.e., $T = \{T_1, T_2, T_3, \dots, T_n\}$ are received from IoT/MCS users to schedule into m of VM , i.e., $\{\text{N}_1, \text{N}_2, \text{N}_3, \dots, \text{N}_\text{m}\}$.

4.1 Defining parameters

Table 1 shows the fundamental notations, terminologies, and concepts that are used in mathematical formulations of the proposed scheduling approach.

5. Proposed work

This section presents the proposed approach DB-

Table 1. DB-MODS schedulers notations

Notations	Description
Cloudlet	Refers to task in CloudSim plus
VMs	Virtual Machines in cloud
MI	Million-Instructions data of cloudlet
MIPs	Million-Instructions Per Second
ET_{ij}	Time of running task T_i on VM_j
TET_{VM_j}	Total execution time of tasks on jth VM
$N_{PE\ j}$	Number of processing elements of VM.
$VM_{MIPS\ j}$	Refers to the speeds of the VM.
deadline (T_i)	is the user-specified time for execution task T_i
budget (T_i)	is the user-specified cost for execution task T_i
VM_{id}	Is the identification number of VM.
$VM_{j\text{-class}}$	Class which VM belongs to.
T_{id}	Identification number of cloudlet.
$T_{i-class}$	Class which task belongs to (i.e. SLHD class).
C_K	K represents cluster number.
δ	is the trade-off coefficient between deadline and budget.
T_F	The number of cloudlets finished with a specific deadline and budget.
T_U	The number of user tasks
μ_{a_j}	Energy that consumes by VM v_j in active state.
η_{i_j}	Energy that consumes by VM v_j in idle state.
$\pounds_j$	Total energy consumption.

MODS. It is a task management and scheduling approach implemented for IoT/MCS in a cloud-based environment. The main goal of the DB-MODS approach is to meet time of response, balance the load, reduce delay, reduce the number of missed critical tasks, and increase the throughput. To achieve this, we emphasize two constraints deadline , and budget in scheduling issue, to enhance the QoS in relation to the two primary parameters (i.e., reduce the cost and time of execution task), and finally increase service gain while also making better use of the hosts and VMs resources.

The proposed approach used the machine learning K-means clustering algorithm to categorize the tasks based on length and deadline, and grouping VMs based on capacity of MIPS, Bandwidth (BW), RAM using thresholds. Our scheduling model represents the users task parameters consist the task length, budget, and deadline.

In DB-MODS, two measures, ready time and load of VM are dynamically updated. The key objective of the DB-MODS scheduler is to maximize resource use and satisfy deadlines for recently coming tasks, cost,

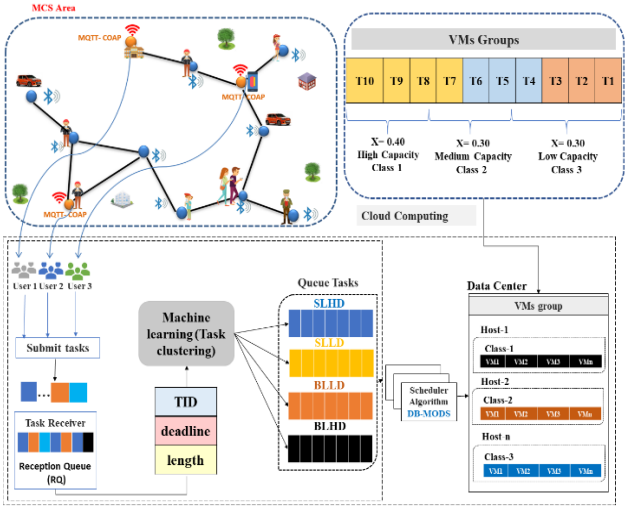


Figure. 2 Architecture of DB-MODS proposed model

and reduce makespan. In our work, the objective function presented aims to minimize makespan and cost for IoT applications (MCS users tasks).

The proposed is comprised of two stages, namely pre-processing and scheduling algorithm.

5.1 DB-MODS system architecture

We propose DB-MODS as a dynamic multi objective task scheduling approach. The primary aim is to execute user requested tasks immediately. In proposed, several algorithms and mechanisms are used to reduce cost and makespan, and these techniques are employed to minimize the average cluster run time as well as the task waiting time.

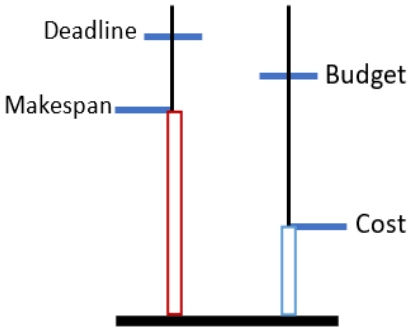
In DB-MODS, K-means was employed to grouping the tasks depending on length and deadline, which there is a clear relationship between them (between the length of the task and the time required by the user to finish it). In addition, using the proposed algorithm DB-MODS scheduler, where an attempt was made to improve the overall time that tasks take to complete. Finally, the clusters of tasks are then sent to hosts whose computing capacity is appropriate for the task cluster constraints. Fig. 2 illustrates the architecture of the proposed DB-MODS model.

5.2 Tasks constraints types

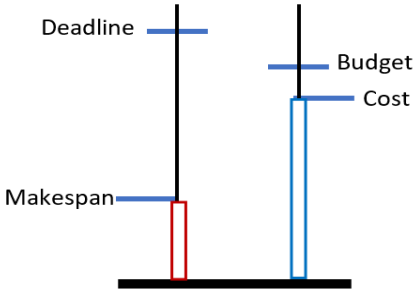
In the DB-MODS approach Each task has one or more constraints depending on the parameters specified by the user, there are two types of constraint are deadline and budget.

Fig. 3 shows the relation between Makespan and Cost which is an inverse relationship as in Eq. (9):

$$Makespan = \frac{1}{Cost} \quad (9)$$



(a)



(b)

Figure. 3 several strategies for achieving deadlines and budget. (a) An example of scheduling that decreases costs while fulfilling deadlines and (b) An example of scheduling that decreases a makespan but satisfies the budget restraint

Were if user deadline is low this means that the task must be executed within the specified time, and this requires a VM with a fast CPU, so the cost here is important as in Fig. 3 where if the user has enough budget he can provide him with the service as he wants, if user deadline is high this means the task is not urgent (need slow CPU) and can execute with low budget but high makespan as shown in Fig. 3 (a), an increase in cost versus a decrease in makespan (Fig. 3 (b)).

Every task needs to be scheduled with a deadline that reflects the fastest possible response time from cloud service providers to requests from users. Cloud providers must react to user requests in a fair amount of time; otherwise, an SLA violation occurs, and the cloud provider must pay the penalty to the users. The task execution time should not be longer than the deadline, as displayed in the following equation:

$$ET(i,j) < Deadline\ (T_i) \tag{10}$$

In cloud systems, cloud providers must pay for using the host/VMs. The consumer is required to pay the price that is determined by cloud providers based on the resources consumed by the consumer. The amount that pays by a consumer per hour/second for using a resource provider's virtual machine must be determined, and the consumer specifies the maximum cost that can be paid as in following:

$$Cost\ (T_i^j) <\ budget\ (T_i) \tag{11}$$

5.3 Tasks classification

Initially, each task has a set of characteristics, Task = {ID, deadline, length, budget}. To carefully select the suitable computing node, need to use task information, status of computing node, and resource availability [20]. Tasks are clustered depending on their length and deadline by K-means method. The prime aim of this proposal is to minimize the time of processing huge data size, which is achieved by less makespan and minimizing overall cost. Therefore, in DB-MODS, for the purpose of load balancing among clusters, task migration between clusters is implemented for purpose of establishing load balancing between groups.

In the proposed approach, clustering is made using K-means algorithm. Applying K-means prevents task clustering from falling into the local optimum problem due it converges readily. The elbow curve method is applied to choose the K value. K-means is an unsupervised learning clustering algorithm. To cluster the tasks by K-means needs to train the model based on features of the tasks and workload type used to determine task clusters. Afterward, the newly arriving task is assigned to the cluster having similar features of tasks. Two features of the task, length and deadline are considered for task classification. The Euclidean distance is used to decide the closest centroid assuming four clusters, C_K, where (k $= 1, 2, 3, 4$).

$$d(t_j - c_k) = \sqrt{\left(T_{length\ j} - c_k\right)^2 + \left(T_{Deadline\ j} - c_k\right)^2} \tag{12}$$

The tasks that are allocated to clusters can be illustrated in matrix form as displays in Fig. 4. The column, and row represent a task T_i and a cluster respectively. Digit 1, in the i^{th} row, and j^{th} column of matrix refers to the task i allocated to cluster j. Each column in the matrix only contains single (1). This means that each task belongs only to a single cluster.

Similar tasks are not always having the same characteristics. However, tasks are clustered according to their length and deadline. Consequently, there is n number of small-length tasks with low-deadline in cluster, n of big-length tasks with high-deadline in another cluster, n of big-length tasks with low-deadline in another one, and n of small-length

	t_1	t_2	t_3	t_4
C_1	0	0	1	0
C_2	1	0	0	1
C_3	0	1	0	0

Figure. 4 An example of assigned tasks to clusters in G generation matrix

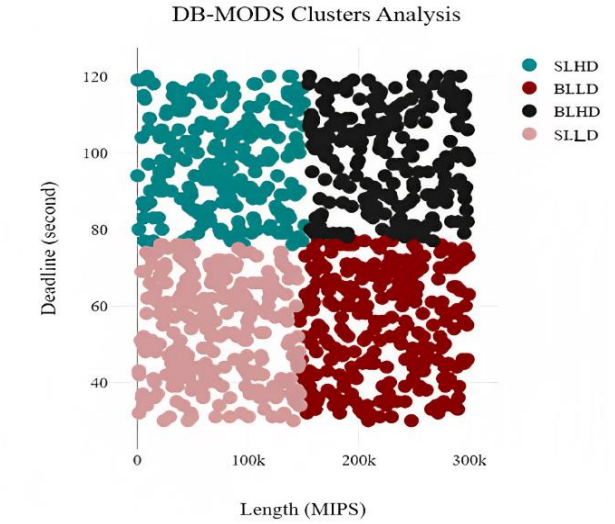


Figure. 5 Cluster analysis of proposed approach

Table 2. Categories of proposed clusters tasks

DB-MODS Queues	Task Class	Description
(LCI)T2	SLHD	Light Compute Intensive
(HCI)T1	SLLD	Heavy Compute Intensive
(HCI)T1	BLLD	Heavy Compute Intensive
(LCI)T2	BLHD	Light Compute Intensive

tasks with high-deadline in cluster as a result of clustering of 1000 tasks with length between $100 \sim 300{,}000$ MI , and with deadline between $30 \sim 120$ second as shown in Fig. 5. Despite cluster with smaller-tasks being mapped to the computing node with lower computing capacity as well as vice versa. However, the two clusters average processing times are substantially dissimilar. As previously mentioned, relevant tasks are clustered in a similar group to prevent the exchange of data among other nodes. Table 2 shows the classes of tasks and power requirements.

Where: T1, T2 are type one and two respectively, where T1 for LD and T2 for HD. HCI is the tasks of cluster requires high computing power. LCI is the tasks of cluster requires low computing power. SLHD: the first letter refers to small and second is length of tasks, third is refer to high and fourth is deadline.

SLLD: Small length and high deadline. BLLD: Big length and low deadline. BLHD: Big length and high deadline.

After classifying the tasks into four categories and viewing the results of the clustering process, we conclude that the SLLD and SLLD categories can be merged because they need the same computing capacity, and thus it becomes three categories.

5.4 Physical nodes classification

After the phase of task classification, tasks are categorized into groups and distributed among distinct clusters. To categorize the VMs, firstly, it is necessary to compute the total capacity of each VM depending on the following formulation:

$$Capacity_{VM_j} = CPU_j + RAM_j + BW_j \quad (13)$$

where $\mathrm{CPU}_j, \mathrm{RAM}_j$ and BW_j are CPU speed in MIPS, the size of memory and bandwidth capacity of VM_j, respectively. Additionally, we compute capacity of each VM class through the sum capacity of each VMs in a specific class as in the following equation:

$$Capacity_{Class_k} = \sum_{VM_j \in Class_k} Capacity_{VM_j} \quad (14)$$

Depending on the capacity calculated by Eq. (14), we arrange the available VMs in a list. Next step in the proposed model, the group of the VMs is determined in relation to the type of task class as shown in Table 3. Therefore, the proposed groups of VMs based on their capacity by using thresholds as shown in Fig. 6. The resulting VMs classes are $\mathrm{VM}_{\mathrm{HC}}$, $\mathrm{VM}_{\mathrm{MC}}$, and $\mathrm{VM}_{\mathrm{LC}}$.

Where: $\mathrm{VM}_{\mathrm{HC}}$, $\mathrm{VM}_{\mathrm{MC}}$ and $\mathrm{VM}_{\mathrm{LC}}$ are VM with high capacity, VMs with medium capacity and VMs with low capacity respectively.

Table 3 shows the allocation of the task groups to suitable VMs groups that meet task cluster requirements.

6. Proposed load-balancing and scheduling algorithm DB-MODS

The proposed DB-MODS scheduling approach is discussed in this section. which has three algorithms the first is for clustering tasks, the second for grouping VMs and the last is for a scheduler. DB-MODS scheduler allocates user tasks to VMs nodes relying on minimum execution time (MET) and cost, then updates status of the task and VMs.

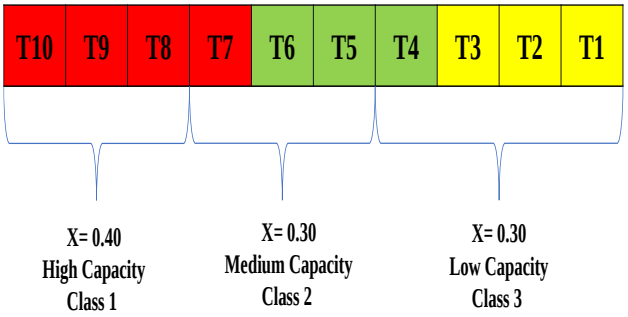


Figure. 6 An example of VMs classification depending on capacity and by using threshold values

Table 3. Allocate the classes of tasks to types of VM groups

Task types in received queue	VM types
BLLD	Type-1 (Large - CPU, BW, RAM intensive).
SLLD+ BLHD	Type-2 (Medium- CPU, BW, RAM intensive).
SLHD	Type-3 (Small - CPU, BW, RAM intensive)

6.1 DB-MODS scheduler algorithm

To perform a process of mapping tasks on virtual machines, DB-MODS scheduler receives task T_i with its characteristic's deadline (T_i) in second as well task length in (MIs) as in algorithm 1. Then by using k-means, the outputs of algorithm 1 include distributing incoming tasks into clusters as mentioned in section 5.3. Algorithm 2 includes a number of VMs and their processing power in MIPS, RAM, and BW as input parameters. The output of algorithm 2 includes grouping VMs into three classes based on the threshold as mentioned in section 5.4. Algorithm 3 is DB-MODS scheduler algorithm to coordinate the process of mapping clusters of tasks on VMs groups using objectives functions.

The objective function (ObjF) of DB-MODS focuses on minimizing the makespan and cost, which is displayed as follows:

$$ObjF = Min\{Makespan, Cost\ (T_i^j)\} \quad (15)$$

The *Quality Measure Objective Function (QOF)* of the proposed work is presented here:

$$QOF = Min\ (\delta \times\ ET_{ij}\ + (1-\delta) \times\ Cost\ (T_i^j)) \quad (16)$$

Algorithm 1: Task Classification Function (T_{CF}) and VMs Classification Function (VM_{CF})

Input: List of unmapped tasks which have constraints (ID, length in MIPS and deadline D in sec.), set of VMs with their characteristics (MIPS, CPU, RAM and BW)
Output: cluster tasks (Ti), and grouping VMs

Algorithm 1: Task Classification Function (T_{CF})

```
Begin {
M←Number of input tasks
K←Number of Cluster
k-Mean (T)    // for tasks classification
for All Task t_j do
    get Metric Variables of task for clustering process
    Task ID, Task length and Deadline
    Normalization for data
    Start the K-Means Clustering and divide them
    into 4 clusters
    Assume Centroid A, B, C, D
    for each Cloudlet k in CL do
        Calculate Euclidean Distance of k with A, B, C
        and D based on Eq. (12)
        if previous distance = new distance then
            Stop Iterations
            else
            Add Ck into minimum clustered distance
            Again Compute the Centroids
        end
    end
end
    get four clusters of tasks LLHD, LLLD, HLLD and
    HLHD
```

And calculate the objective function for user parameter to compare with Eq. (16) as follow:

$$U - ObjF = \delta \times deadline(T_i) + (1-\delta) \times budget(T_i) \quad (17)$$

Where: deadline (T_i) is a largest amount user-assigned time allowed to execute task T_i in the cloud, which if exceeded, will be violated SLA. budget (T_i) is the highest permitted cost determined by the cloud user. which if exceeded, will be violated SLA. δ is the trade-off coefficient between the deadline (T_i) and budget (T_i) and its $\in [0,1]$. If the value of $\delta > 0.5$, the task assignment technique prioritizes reducing execution time over overall operating costs. If the value of $\delta < 0.5$, execution time is less significant than operating costs. The value of δ depends on the value of the budget or the level of the needed response time.

The proposed includes these attributes for each task: {T_{ID}, task length, data file size, number of PES. budget and deadline}. While, each task T_i has the constraint parameter can be defined as $T_i = \{L(task_i), \text{deadline}\ (T_i)\ , \text{budget}\ (T_i), T_{i-class}\}$,

Algorithm 2: VMs Classification Function (VM_{CF})

Create number of PH_i with their characteristics
Create list of (VMs, PH_i) with different CPUj + PEj + BWj *and determine all required parameters*
for *each Virtual Machine VM_j on host PH_i* do
  compute capacity of all VMs in VM List [ ] based: VMC=vm.getRam().getCapacity()+vm.getMips()+ vm.getBw().getCapacity()
  Arrange the list of VMs in decreasing order
  VMs class$_{NO.}$ = number of VMs%k //NO. of VMs class = NO. of cloudlet class
  if *VMs class $NO.$ mode k* = 1 then
    Class 1=class 1+ 1
  end
  if *VMs class $NO.$ mode k* = 2 then
    class1=class1+ 1
    class2=class2+ 1
  end
  Group VMs based on the threshold (ψ, φ and ε) into three Class (high-cap., medium-cap.,low-cap.), ($Class_1$,..$Class_n$)
  Create number list of type VM == VMs class$_{NO.}$, ($VL1, VL2, VL3$)
  if $VM \in \psi$ then
    Set vm.class == VMC1
    Add.vm to vm VL1
  end
  else if $vm \in \varphi$ then
    Set vm.class == VMC2
    Add.vm to vm VL2
  else
    Set vm.class == VMC3
    Add.vm to vm VL3
  end
end
where:
VMC1 capacity > VMC2 capacity and VMC2 capacity > VMC3 capacity

where $L(task_i)$ refers to task length measured in (MI), deadline (T_i) The time limit for completing the user task in second, budget (T_i) is the cost specific by user in dollar and $T_{i-class}$ is a class which task belongs to. Similarly, each VM_i is also characterized as $VM_i = \{VM_{ID}, V_{MIPS}, VM_{RAM}, VM_{BW}, VM_{j\text{-class}}\}$, where $VM_{j\text{-class}}$ is a class which VM belongs to.

The execution-time ET_{ij} ,and cost of task T_i is calculated during run and each task has deadline (T_i) , and budget (T_i) to execute from user. To select the most suitable processor for the current task, the *QOF* for each processor $p_j \in$ list 1 is computed based on Eq. (16), and then calculate the objective function for user *UserObjF* parameter based on Eq. (17), then calculate the minimum value can return from QOF with VM_{ID} , then compare *UserObjF* with *QOF*. And mapped cloudlet to VM has a minimum value, then task and VM status lists are updated. If *UserObjF* is greater than *QOF*, DB-MODS scheduler migrates task to next group.

7. Performance metrics

In the context of testing and evaluation the DB-MODS performance based scheduling of the task in cloud environment includes makespan, cost, success rate, throughput, energy consumption, and performance improvement ratio (PIR) are considered as analytical performance metrics. In the literature, the majority of researchers use the makespan or cost as the single criterion to assess the efficiency of their algorithm. The DB-MODS approach is evaluated using the following metrics:

- **Makespan**

One of the most popular metrics for evaluating the effectiveness of scheduling in cloud environment. It can be characterized as the finish time of execution latest task. If the makespan value is small, this refers that the cloud broker correctly assigning workloads to the relevant VMs. The definition of makespan as in Eq. (3) [24].

- **Success rate**

The success rate is used to determine the fault tolerance approach's effectiveness. It is a crucial performance indicator that is used to evaluate the reliability of the cloud's systems. This is the ratio between the number of tasks finished and the time limit allotted by users (deadline and budget). It is computed using Eq. (18) [25]:

$$Success\ Ratio\ = \frac{T_F}{T_U}\ x\ 100 \tag{18}$$

Where:

T_F Number of tasks finished within user deadline and budget. T_U is a number of users tasks.

- **Cost**

The cost of processing a task in cloud is defined as the demand cost to process the incoming task to cloud and can be computed from the cost of the CPU, memory cost, and cost of bandwidth consumption. To calculate cost when t_i executes at v_j can be computed by Eq. (4), and total cost by Eq. (8) [16].

- **Throughput**

The term throughput denotes the number of tasks a virtual machine completes in a certain period of time [25].

$$Throughput = \frac{NTSe}{Makespan} \tag{19}$$

Algorithm 3: DB-MODS Scheduler Algorithm

Input: List of unmapped tasks which have constraints (ID, length in MIPS and deadline D in sec.), set of VMs with their characteristics (MIPS, CPU, RAM and BW)
Output: Map (Ti, VMj) with Minimizing the Makespan and expenditure cost for these tasks
ETij = 0, Cost(T_k^i)=0, VM_{Id} = 0, *U-ObjF* = 0, minQOF= 0 Timespan = 0, α = 0
while T_i != *null* do
  for $j = 1$ *to* m do
    if *cloudlet class LLHD || LLLD class* then
      maxTries1 = VL1.size();
      For each machine VM (i) [] VL1 [ ] Resource list and$_j$ maxTries1 do
      ET_{ij} = *computeET*ij (T_i, VMj) *based Eq.*(1)
      Cost (T_k^i) =compute Cost (T_i, VM_j) based Eq.(4)
      QOF=compute (α × ETij + (1 −α) × Cost (T_k^i))
      M-QOF = Min (QOF)
      list.add (VMId, M-QOF) //Get minimum value of QOF function from list of vmgroup1with vmid
    end
  end
  U-ObjF = compute (α × deadline (T_i) + (1 −α) × budget (T_i))
  if *U-Obj <= M-QOF* then
    Map.add (Ti,VMj)//return cloudlet id that allocated to VMid
    update VMs status (utilities)
    updateVm list (VL1)
  end
  else
    set cloudlet class = 2
    migrate cloudlet to next group
  end
end
Note: repeat steps from 5-26 for each class of task, but with different values of α Computation performance

Calculate the metrics:

Makespan based on the Equation 3.
Success Rate based on the Equation 18.
Energy consumed based on the Equation 24.
Throughput based on the Equation 19.
Cost based on the Equation 8.
PIR based on the Equation 25.

Where NTSe is a number of tasks successfully executed.

In a cloud context, throughput represents productivity or overall system performance i.e. the higher value of throughput refers to the better the system performs.

- **Energy modeling**

The amount of energy that cloud nodes consume is generally due to the execution process, power conditioning, and cooling of the system. VM is the foundation of our energy model in execution environment. mostly, the energy consumption is depending on VM state. A virtual machine state can either be in active mode, or idle. Here, we suppose the VM will be in an active state when executed it, otherwise, a VM considers idle. Typically, the energy taken up by a VM in idle mode is between 60-70% of the active state. Let, μ_{a_j} and η_{i_j} indicate the energy that consumes by VM v_j in both states active, and idle consecutively. while total consumption of energy $(£_j)$ of VM v_j is calculated by taking into account active and idle states together and calculation is as in Eq. (20) [14, 26].

$$£_j = \left(\mu_{a_j} + \eta_{I_j}\right) \times MIPS_j \tag{20}$$

Calculate the total execution time $(£_j)$ for every task allocated to v_j as in the following expression:

$$TET_{VM_j} = \sum_{i=1}^{n} X_i^j \times ET_{ij} \tag{21}$$

Where binary value $X_i^j = 1$, if t_i is allotted to v_j. If t_i is not allotted to v_j then $X_i^j = 0$. The energy that is consumed by v_j in active, and idle mode is computed as follows:

$$\mu_{a_j} = TET_{VM_j} \times ß_j \tag{22}$$

$$\eta_{i_j} = \left(Makespan - TET_{VM_j}\right) \times 0.6 \times ß_j \tag{23}$$

Where: $ß_j = 10^{-8} \times \left(MIPS_j\right)^2$ in Joules/MI.

The overall energy-consumption (δ) for cloud VMs is computed as:

$$ov_{£_j} = \sum_{j=1}^{m} £_j \tag{24}$$

- **Performance improvement ratio (PIR%)**

PIR metric employed to indicate the effectiveness of a DB-MODS algorithm depending on the decrease in time of execution. Therefore, it is considered one of the crucial measures that benefit to determine the

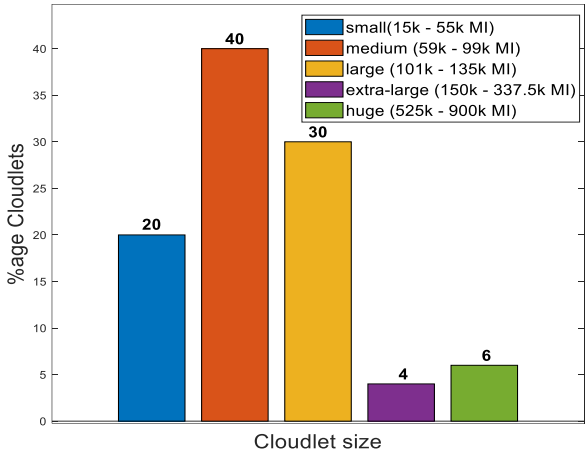


Figure. 7 Composition of GOCJ realistic workload

Table 4. Parameters of simulation environment

version of simulator	CloudSimPlus v7.3.0
Experimental environment	Intel(R) Core(TM) i7-10750H CPU @ 2.6 GHz 2.59 GHz. Memory 16.00 GB HD 1 TB

effectiveness of the work. It is calculated as follows [27].

$$PIR\% = \frac{Makespan_r - Makespan\text{-}_{proposed}}{Makespan_r} \times 100 \qquad (25)$$

where $Makespan_r$ and $Makespan_{proposed}$ is refer to makespan acquired from the rth algorithm also from proposed algorithm.

8. Implementation and experimentation results

8.1 Experimental setup

To simulate DB-MODS we used the CloudSimPlus platform for simulating, modulization, and testing the performance of our proposed DB-MODS scheduling approach. which include several models, and algorithms, such as heuristic-algorithms, VM migration approach, scalability, more precision, and ease to operate [28]. Additionally provides fundamental classes for characterizing cloud system ingredients such as datacenters, brokering policy, computational resources, VMs and cloudlet [29].

8.2 Benchmark datasets

To assess the effectiveness of our scheduling method, two distinct datasets are used: first) random dataset, second) Google cloud jobs (GOCJ) dataset. workload datasets are be described in the following:

8.2.1. Random dataset

We generated tasks with length varying from 1k-100k MIs. There is a total of 1k tasks in randomly generated dataset. The dataset contains task ID, task length, deadline and budget.

8.2.2. GOCJ dataset

GOCJ is regarded as Google such a real dataset produced as a result from workload that reflects behaviors of Google cluster evaluation employing bootstrapped (i.e. Monte-Carlo). simulation, a widely used simulation technique. In the GOCJ dataset, tasks length lies between range from $15x10^3$ - $900 x10^3$ MIs, then, dataset is categorized as: jobs with small size ($15 x10^3$-$55 x10^3$ MIs), medium size jobs ($59 x10^3$-$99 x10^3$ MIs), large size jobs ($101 x10^3$ - $135 x10^3$ MIs), extra size jobs ($150 x10^3$ - $337.5 x10^3$ MIs), and huge-size jobs($525x10^3$-$900x10^3$MIs) as shown in Fig. 7 [16,30], and available on: https://data.mendeley.com/datasets/b7bp6xhrcd/1

8.3 Implementation environment

The experimental environment includes CPU Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz, memory (16.0 GB) HD 1 TB, and uses Eclipse IDE 2022-09, jdk-17.0.1, and CloudSim Plus v7.3.0 as shown in Table 4.

Simulated the dynamic arrival of cloudlets randomly during simulation runtime. At any time, the simulation clock updates, a new cloudlet will be created from dataset based normal distribution equation.

To cluster cloudlets, cloudlets values need to be normalized. The normalization procedure reduces a variety of values to a narrow range, such as 0 and 1, and is used as a pre-processing, mapping, and scaling method to turn severely skewed results into a new values range [31].

In order to evaluate efficiency of proposed DB-MODS algorithm, we compared with other methods in the literature such as LA-based scheduling (LAS), Greedy-R and Greedy-P [14], EEVS [15], and random in scenario 1. And compared it with well-known methods based on FCFS scheduling algorithm, (MOABCQ-FCFS), (MOABCQ_LJF), Q-learning method, multi-objective PSO (MO-PSO), Max-Mi, and multi-objective-cuckoo search (MO-CS) [16], artificial-bee-colony (ABC) algorithm, and the largest-job-first (HABC-LJF) [17] in scenario 2, and HESGA, GA, and ACO [19] in scenario 3.

Various numbers of cases that are acquired by the generation of a different count of tasks are between 200 and 1000 whose length is ranging from 1000 MI

to 900,000 MI and varies number of VMs from (40) to (100). Therefore, for simulation the proposed, we have carried out three scenarios as in the following section.

8.4 Experimentation results and discussion

This section shows the experiments of the proposed DB-MODS scheduling approach and discusses the results.

Scenario 1:

For simulation scenario 1, the following specific settings and parameters are provided:

- The VMs are modeled to have processing capacity with value lying in range [3k-6k] MIPS.
- A tasks creation is modeled to simulate the tasks with the behavior of arrival in real-time to the cloud system.
- The poisson distribution is used to simulate task arrival, and the time of arrival is distributed in an exponential manner.
- Deadline of tasks is defined as: $\text{deadline}\,(T_i) = ar_i + \text{base}D$, where ar_i is the task arrival time, $\text{base}D$ is in uniform distribution form in range between $U(5,10)$.
- Task length is specified to be between [1000 to10000] MI.
- To prevent the impact of uncertainty factors on the experimental outcomes, each experiment is run 20 times, and the average is calculated.

In this scenario, we have taken 40 VMs which are constant in number and varying in processing capacity needed to complete the user task allocated to its. The collection of tasks generated in a random way whose numbers are different from 200 to 1000 in intervals of 200 based on a random dataset is considered for the evaluation of the proposed approach.

To evaluate the effectiveness of the proposed DB-MODS in this scenario several metrics are considered. These parameters include makespan, success ratio, and total energy consumed.

A comparison result of proposed DB-MODS method performance in terms of makespan is shown in Fig. 8. There were 40 VMs used in this experiment, and the system was given 200, 400, 600, 800, and 1000 tasks. After testing the random dataset, the experimental findings in Fig. 8 show that the DB-MODS approach reduced the average makespan more effectively than the LAS, EEVS, Greedy-R, Greedy-P, and random.

In our tests, we used a value of $\delta = 0.5$, indicating that time and cost are equally important in the objective. The DB-MODS model achieves the shortest makespan by minimizing the time required to complete tasks within the given budget and deadline constraints. This results in a higher performance level for the DB-MODS model, achieved through clustering the tasks, and assigning large tasks to high-capacity VMs and small tasks to low-capacity VMs. By reducing the average execution time, this approach minimizes the makespan. We conclude, the DB-MODS model's performance is optimized by prioritizing task completion within the specified time and cost constraints, accomplished through intelligent task clustering.

Fig. 9 illustrates that for all the compared methods. There is little difference in success rate, regardless of the number of tasks. While it is clear that when increasing the number of tasks leads to an increase in the failure rate for all tested methods. This is because of the inefficient distribution of tasks on the VMs, which leads to an increase in execution time, and therefore it is not possible to meet the negotiated deadline between the service provider and the user. whereas DB-MODS maintained the success rate of tasks even when increasing them, and the reason is the efficient distribution of tasks on the VMs, this is due to use the K-means algorithm, which groups tasks depending on their length and deadline, where we conclude from our work that the large tasks and with short deadline, they were executed on VMs with high resources to ensure their execution is within the specified time which.

Fig. 10 illustrates the performance analysis of DB-MODS algorithm in term of consumption of energy for the various task combinations. Fig. 10 shows that DB-MODS conserves energy better than other approaches, and this tendency is evident as the number of tasks increases. This experimental finding indicates that the DB-MODS scheduler aids the efficient use of VM nodes, which contributes to energy conservation low as possible compared with others methods. The results indicate that our work optimizes the processing time through distributed tasks on VMs in an efficient way and this means decreasing the idle time (η_{U_j}). The main reason is due to the objective function presented in Eq. (16), which tried to reduce the makespan. Minimizing the makespan of a system is equivalent to conserving energy, since the makespan is directly related to the energy consumption of the system. By reducing the makespan, the idle time of virtual machines is reduced as well. As a consequence, the energy consumed by the system is lower when virtual machines have less idle time.

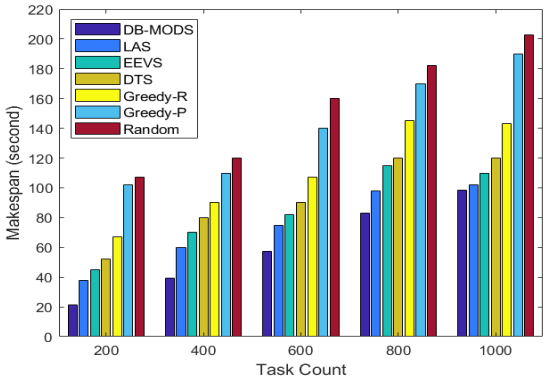


Figure. 8 Makespan under scenario 1 using random dataset

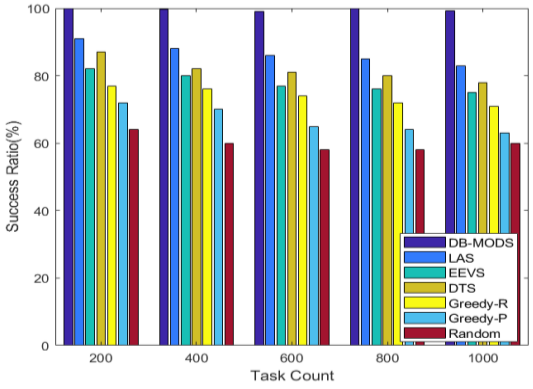


Figure. 9 Success rate comparison under scenario 1 using random dataset

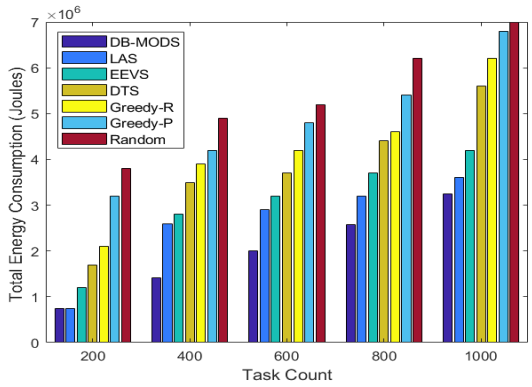


Figure. 10 Total energy consumption under scenario 1 using random dataset

In this paper, we used the statistical analysis t-test to conduct according to the makespan values on the same random dataset with 200 tasks, but one on 40 VMs and another on 60 VMs as shown in Fig. 8 and Fig. 11 respectively.

The null hypothesis assumes that the success rate for all implemented algorithms is 75%. we applied a (t-test) using 200 tasks, and the number of VMs is 40 for (Test-1) whereas, 60 for (Test-2). For Test-1 the mean calculated value = 61.76, Std. deviation = 32.32, and N equal to 7. Whereas Test-2, the mean value = 61.6, Std. deviation = 35.0, and N equal to 7.

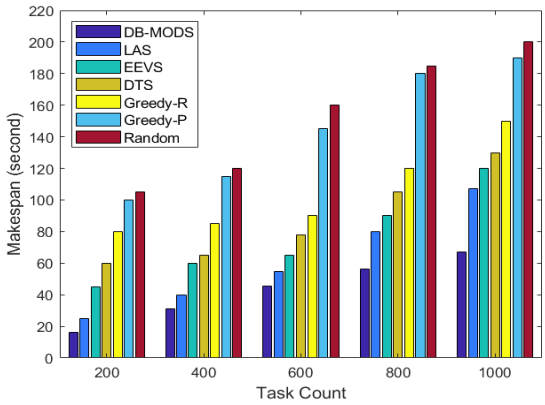


Figure. 11 Makespan under scenario 1 using random dataset (60 VMs)

Table 5. Simulation environment (scenario 2)

Type	Parameter	Value
Host	Host Number	20
	MIPS	$177.73^{'}x10^3$
	Storage capacity	2TB
	Bandwidth	10GB/s
	RAM	16GB
	VM-Monitor	Xen
Data Center	DC Number	1
	VM-Scheduler	Time Shared
	Memory Cost	$0.1 - 1.0$
	Storage Cost	$0.1 - 1.0$
VM	VMs number	$5 - 100$
	Speed of processor	$3.5x10^3$ – $100x10^3$ 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
	PEs Number	1
	VM Monitor	Xen
Cloudlet/Task	Length of Tasks	$1x10^3$-$900x10^3$
	Tasks number	$200 - 1000$

The value of t is (0.01) and (degree-of-freedom) (df)=12. The value of t(p) at 12 df is 0.993 when significance is 5% (value is 0.05) for 12 df in (two-tailed). At a 5% level of significance, the null-hypothesis may be to accepted because the estimated t-value is smaller than 0.993.

Scenario 2:

In this scenario, simulated a virtual environment to evaluate efficiency of DB-MODS in terms of cloud-based load balancing and scheduling. The parameters used in this scenario for the simulation are defined as shown in Table 5. We have considered four objectives in scenario 2 are makespan, throughput, cost and PIR as follows:

Fig. 12 when comparing DB-MODS against MOABCQ-FCFS, MOABCQ-LJF, Max_Min method, HABC_LJF, FCFS, Q-learning, MO-PSO, and MOCS algorithms, the results show clear superiority in reducing makespan. The reason for this is due to the mechanism of dividing tasks and resources into clusters and groups and then distributing them in several levels. Fig. 13 shows the number of virtual machines in each group with the total capacity. Thus, the clustered tasks will be submitted according to their cluster to the appropriate groups of VMs. As the number of tasks increases, the makespan of DB-MODS becomes significantly smaller compared to the other algorithms because our algorithm selects the resource based on the task class. This approach has the capability to improve the convergence speed and the effectiveness of optimization in the DB-MODS algorithm. The proposed demonstrates a high capability to reduce the makespan and determine the optimal resources for processing incoming tasks, and arrive at the best decision even when the task count increases.

Fig. 14 compares the effectiveness of the DB-MODS method using important throughput metric, can be defined is the number of tasks finished in a specific time which is calculated using Eq. (19).

Proposed algorithm processes more tasks in a given time due to better balancing. The GOCJ dataset used in this experiment. The result of simulation indicates that DB-MODS outperformed all other algorithms in terms of throughput. And higher throughput value achieved at 800 tasks, DB-MODS gave higher values than the others at 195%, 163%, 88%, 86%, 86%, 77%, 26% and 22% when compared to Max-Min method, FCFS algorithm, Q-Learning method, HABC-LJF approach, MOPSO technique, MOCS, MOABCQ-FCFS and MOABCQ-LJF respectively. The result indicate that proposed is stable at each number of tasks 200, 400, 600, 800 and 1000. From the result, based on our examination, it can be inferred that the proposed method can perform load balancing effect to ensure that no single server is overloaded. Because resources and tasks are properly grouped and divided in the pre-processing process which leads to increased overall cloud throughput.

In Fig. 15, the performance from a cost perspective is calculated for scheduled tasks, which are assigned to 100 VMs under GOCJ dataset. The maximum costs for 200, 400, 600, 800, and 1000 number of the task are 117.733, 229.773, 368.108, 446.081, and 581.574 which are less compared to existing algorithms such as MOABCQ-FCFS, MOABCQ-LJF, Max_Min method, FCFS algorithm, HABC-LJF, Q-learning, MO-PSO, and MOCS

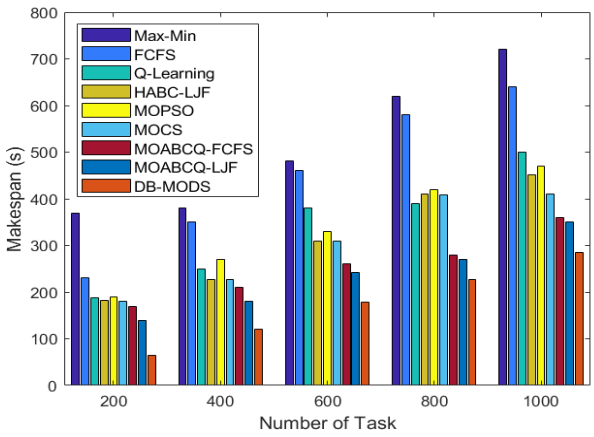


Figure. 12 Comparison of makespan under scenario 2 on GOCJ dataset

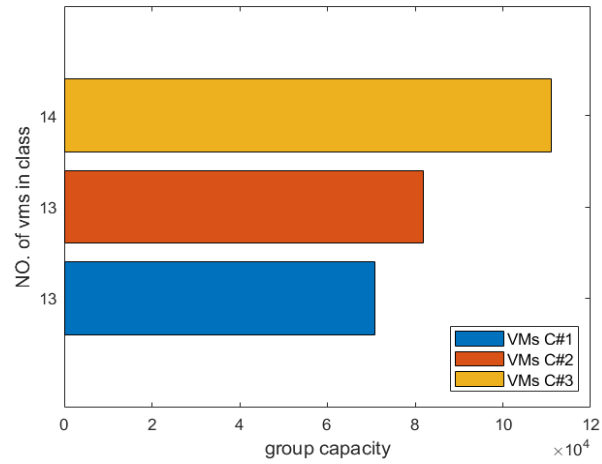


Figure. 13 VMs number comparison of the different classes

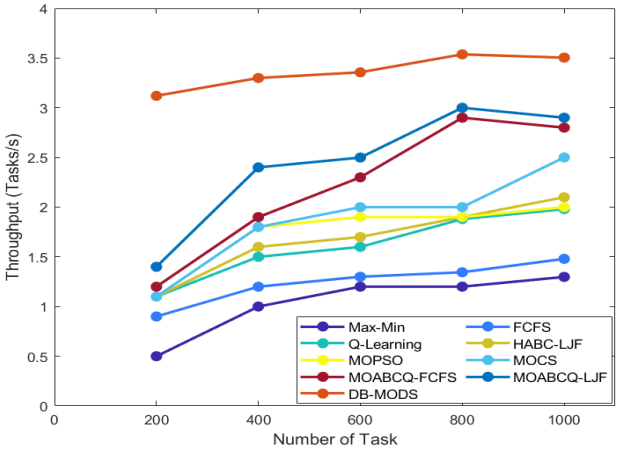


Figure. 14 Comparison of throughput under scenario 2 on GOCJ dataset

techniques. Because the DB-MODS reduce tasks execution time due to better distribution of tasks on VMs by running the task with large length on high VM capacity and this led to reducing in execution time, since the total cost depends upon the CPU, BW,

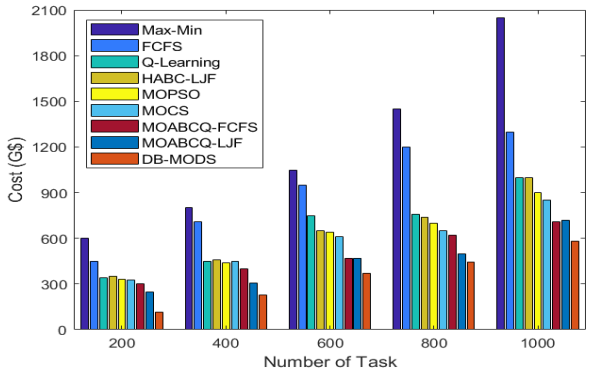


Figure. 15 Cost comparison under scenario 2 on GOCJ dataset

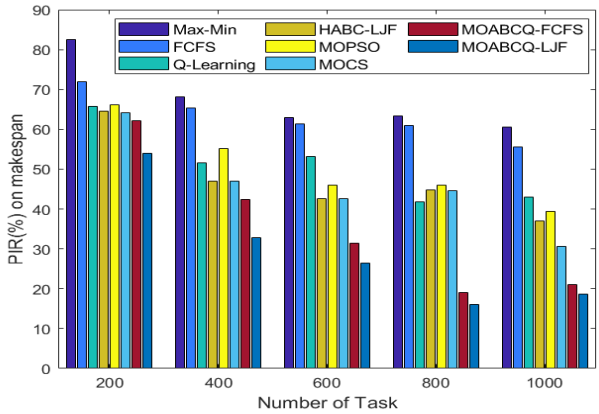


Figure. 16 PIR (%) on makespan comparison under scenario 2 on GOCJ dataset

and RAM usage cost multiply by execution time, thus, the total cost decreases. Also, we concentrate in our objective function on minimizing user cost by set δ = 0.5 through selecting a VM with a small cost, and this leads to executing a task on a VM that returns a minimal cost.

The PIR (%) of the DB-MODS approach based on makespan as it relates to the MOABCQ-FCFS, MOABCQ-LJF, Max-Min, FCFS, HABCLJF, Q_learning, MO_PSO, and finally MOCS algorithm is presented in Fig. 1. For the GOCJ workload, the results show that the DB-MODS algorithm produces 82.60%–60.48%, 72.02%–55.54%, 65.76%–43.09%, 64.64%–37.05%, 66.12%-39.46, 64.24%-30.60, 62.14%-20.96% and 54.03%–18.71% makespan time improvements over the (Max-Min), FCFS, Q_Learning, HABC_LJF, MO-PSO, MOCS, MO-ABCQFCFS and MO-ABC-QLJF algorithms, respectively.

Scenario 3:

In this scenario, simulating is to evaluate the effectiveness of DB-MODS in makespan and throughput to show a performance of our proposed when a huge number of tasks. In this scenario, the parameters used for the simulation are: length of the task 1000-20000 MI, number of tasks from 1000 to 5000, number of VMs is 1000, MIPS is 500-2500, BW is 500-1500, RAM from 256 to 2048 and PEs from 1 to 4.

In Fig. 17, a comparison of makespan is shown for various algorithms (DB-MODS), HESGA, GA, and ACO with different numbers of tasks. The findings indicate that the DB-MODS algorithm has minimize the makespan by 46% in average. This proves that our proposal is better when compared even with a large number of tasks, and it becomes clear that the time decreases with the growth in tasks number significantly, and this is due to the balanced distribution of tasks on VMs and the speed of adaptation, in addition to finding the appropriate VMs better as a result of the clustering and dynamism provided by the algorithm.

Fig. 18 the algorithm we developed has improved task scheduling throughput in cloud environments when compared to HESGA, GA, and ACO by maximize throughput by 39% in average. The algorithm may be better at allocating resources in a way that maximizes utilization and reduces wastage. The algorithm may be better at placing tasks on the most appropriate servers or VMs, based on factors such as execution time, and the ability to faster decision-making based on real-time data about resource availability and task requirements. This can reduce the time required for scheduling and improve overall throughput.

9. Conclusion and future works

Resource management and task scheduling are crucial challenging of the cloud-based IoT/MCS ecosystem**.** the challenging represented in select the optimal processing node. because users want to complete their application in given time and within specified budget, whereas the service provider wants to utilizing the resource to gain maximum profit. In this paper, we proposed deadline-budget multi-objective dynamic scheduling scheme (DB-MODS). The proposed approach employs the K-means to cluster tasks based length and deadline to assigning to appropriate computing nodes, and grouping VMs based on capacity using thresholds. DB-MODS depend on objective function which consider user task constraint i.e. deadline and budget to minimizing the rejection rate of tasks. Two scenarios were conducted on random and real GOCJ dataset using CloudSim plus. The proposed approach facilitates balancing load workload across the system's existing resources. in addition, is clear effective in improving user QoS and save SLA through reduction makespan, cost, energy consumption, failure task ratio, and

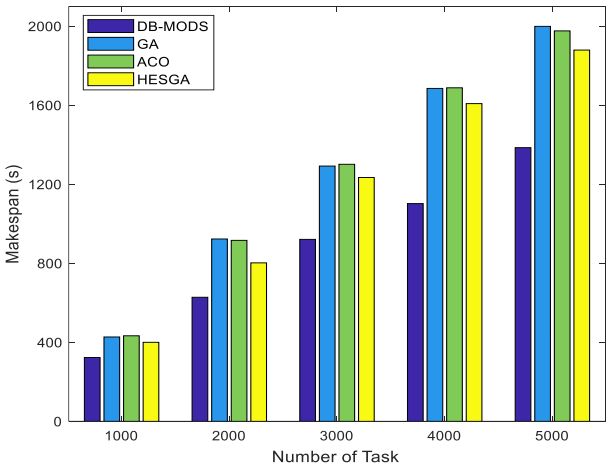


Figure. 17 Makespan under scenario 3

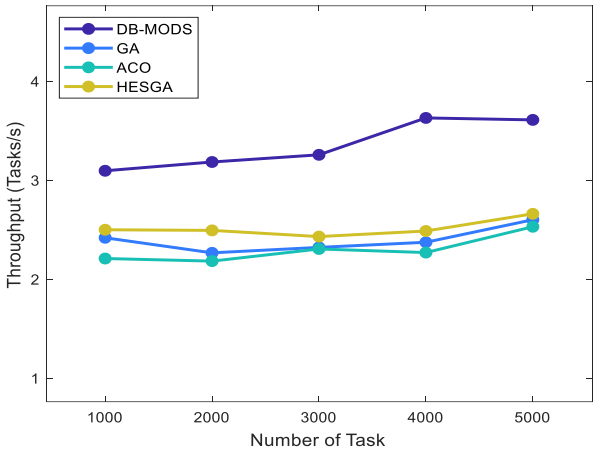


Figure. 18 Comparison of throughput under scenario 3

maximize throughput when compared to the EEVS, Random, Greedy-R, Greedy-P, LAS, and DTS in first scenario. HABC-LJF, Max-Min, Q-Learning, MOPSO, MOCS, FCFS, MOABCQ-FCFS and MOABCQ-LJF in second scenario, and HESGA, GA, and ACO in third scenario.

Task scheduling in Cloud/Fog for IoT and MCS environments might be an issue and it could be our future study. We will propose a scheduling method in multi-environments and planning to apply meat-heuristic algorithms and other machine learning techniques. Furthermore, the proposed approach will be evaluated in a real-world scenario.

Data availability

The authors can provide the data that was used to support the findings of experiments upon request.

Conflicts of interest

The authors Abbas M. Ali Al-muqarm and Dr. Naseer Ali Hussien declare there are no conflicts of interest.

Author contributions

Conceptualization, Abbas M. Ali Al-muqarm, Naseer Ali Hussien; methodology, Abbas M. Ali Al-muqarm; software, Abbas M. Ali Al-muqarm; validation, Abbas M. Ali Al-muqarm and Naseer Ali Hussien; formal analysis, Abbas M. Ali Al-muqarm; investigation, Abbas M. Ali Al-muqarm, Naseer Ali Hussien; resources, Abbas M. Ali Al-muqarm, Naseer Ali Hussien; data curation, Abbas M. Ali Al-muqarm; writing original draft preparation, Abbas M. Ali Al-muqarm, Naseer Ali Hussien; writing review and editing, Abbas M. Ali Al-muqarm, Naseer Ali Hussien; visualization, Abbas M. Ali Al-muqarm; supervision, Naseer Ali Hussien.

Acknowledgments

The authors would like to thank Islamic University for partially supporting this project. Thanks, are extended to the (anonymous) reviewers and Editor in Chief for their great efforts and valuable comments.

References

[1] A. M. A. A. Muqarm and F. Rabee, “Prediction Communication Time and Data Size Based-Bluetooth in Mobile Crowdsensing for IoT”, In: *Next Generation of Internet of Things: Proceedings of ICNGIoT 2021*, 2021, pp. 445–466.

[2] N. Manikandan, N. Gobalakrishnan, and K. Pradeep, “Bee optimization based random double adaptive whale optimization model for task scheduling in cloud computing environment”, *Comput. Commun.*, Vol. 187, pp. 35–44, 2022.

[3] P. Krishnadoss, C. Chandrashekar, and V. K. Poornachary, “RCOA Scheduler: Rider Cuckoo Optimization Algorithm for Task Scheduling in Cloud Computing”, *Int. J. International Journal of Intelligent Engineering and Systems*, Vol. 15, No. 5, pp. 505–514, 2022, doi: 10.22266/ijies2022.1031.44.

[4] H. B. Alla, S. B. Alla, A. Ezzati, and A. Touhafi, “A novel multiclass priority algorithm for task scheduling in cloud computing”, *J. Supercomput.*, Vol. 77, No. 10, pp. 11514–11555, 2021.

[5] M. A. Alworafi and S. Mallappa, “A collaboration of deadline and budget constraints for task scheduling in cloud computing”, *Cluster Comput.*, Vol. 23, No. 2, pp. 1073–1083, 2020.

[6] N. R. Rajalakshmi, A. Dumka, M. Kumar, R. Singh, A. Gehlot, S. V. Akram, D. Anand, D. H.

Elkamchouchi, and I. D. Noya, "A Cost-Optimized Data Parallel Task Scheduling with Deadline Constraints in Cloud", *Electronics*, Vol. 11, No. 13, p. 2022, 2022.

[7] G. Muthusamy and S. R. Chandran, "Cluster-based task scheduling using K-means clustering for load balancing in cloud datacenters", *J. Internet Technol.*, Vol. 22, No. 1, pp. 121–130, 2021.

[8] T. Khan, W. Tian, G. Zhou, S. Ilager, M. Gong, and R. Buyya, "Machine learning (ML)–Centric resource management in cloud computing: A review and future directions", *J. Netw. Comput. Appl.*, p. 103405, 2022.

[9] J. Varghese and J. Sreenivasaiah, "Entropy Based Monotonic Task Scheduling and Dynamic Resource Mapping in Federated Cloud Environment", *International Journal of Intelligent Engineering and Systems*, Vol. 15, No. 1, pp. 235–250, 2022, doi: 10.22266/ijies2022.0228.22.

[10] M. Kumar, S. C. Sharma, A. Goel, and S. P. Singh, "A comprehensive survey for scheduling techniques in cloud computing", *J. Netw. Comput. Appl.*, Vol. 143, pp. 1–33, 2019.

[11] M. Adhikari and T. Amgoth, "Heuristic-based load-balancing algorithm for IaaS cloud", *Futur. Gener. Comput. Syst.*, Vol. 81, pp. 156–165, 2018.

[12] N. Alaei and F. S. Esfahani, "RePro-Active: a reactive–proactive scheduling method based on simulation in cloud computing", *J. Supercomput.*, Vol. 74, No. 2, pp. 801–829, 2018.

[13] F. Alhaidari and T. Z. Balharith, "Enhanced round-robin algorithm in the cloud computing environment for optimal task scheduling", *Computers*, Vol. 10, No. 5, p. 63, 2021.

[14] S. Sahoo, B. Sahoo, and A. K. Turuk, "A Learning Automata-Based Scheduling for Deadline Sensitive Task in The Cloud", *IEEE Transactions on Services Computing*, Vol. 14, No. 6, pp. 1662-1674, 1 Nov.-Dec. 2021, doi: 10.1109/TSC.2019.2906870.

[15] Y. Ding, X. Qin, L. Liu, and T. Wang, "Energy efficient scheduling of virtual machines in cloud with deadline constraint", *Futur. Gener. Comput. Syst.*, Vol. 50, pp. 62–74, 2015.

[16] B. Kruekaew and W. Kimpan, "Multi-Objective Task Scheduling Optimization for Load Balancing in Cloud Computing Environment Using Hybrid Artificial Bee Colony Algorithm With Reinforcement Learning", *IEEE Access*, Vol. 10, pp. 17803-17818, 2022, doi: 10.1109/ACCESS.2022.3149955.

[17] 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.

[18] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy", *IEEE Trans. Autom. Sci. Eng.*, Vol. 15, No. 2, pp. 772–783, 2017.

[19] S. Velliangiri, P. Karthikeyan, V. M. A. Xavier, and D. Baswaraj, "Hybrid electro search with genetic algorithm for task scheduling in cloud computing", *Ain Shams Eng. J.*, Vol. 12, No. 1, pp. 631–639, 2021.

[20] Z. Jalalian and M. Sharifi, "A hierarchical multi-objective task scheduling approach for fast big data processing", *J. Supercomput.*, Vol. 78, No. 2, pp. 2307–2336, 2022.

[21] D. A. Dewi, T. Mantoro, U. Aditiawarman, and J. Asian, "Toward Task Scheduling Approaches to Reduce Energy Consumption in Cloud Computing Environment", *Multimed. Technol. Internet Things Environ.* Vol. 3, pp. 41–58, 2022.

[22] V. Sharma and M. Bala, "An improved task allocation strategy in cloud using modified K-means clustering technique", *Egypt. Informatics J.*, Vol. 21, No. 4, pp. 201–208, 2020.

[23] S. Nabi and M. Ahmed, "OG-RADL: Overall performance-based resource-aware dynamic load-balancer for deadline constrained cloud tasks", *J. Supercomput.*, Vol. 77, pp. 7476–7508, 2021.

[24]M. K. Patra, S. Misra, B. Sahoo, and A. K. Turuk, "GWO-Based Simulated Annealing Approach for Load Balancing in Cloud for Hosting Container as a Service", *Appl. Sci.*, Vol. 12, No. 21, p. 11115, 2022.

[25] V. Sathiyamoorthi, P. Keerthika, P. Suresh, Z. J. Zhang, A. P. Rao, and K. Logeswaran, "Adaptive fault tolerant resource allocation scheme for cloud computing environments", *J. Organ. End User Comput.*, Vol. 33, No. 5, pp. 135–152, 2021.

[26] S. K. Mishra, D. Puthal, J. J. P. C. Rodrigues, B. Sahoo, and E. Dutkiewicz, "Sustainable service allocation using a metaheuristic technique in a fog server for industrial applications", *IEEE Trans. Ind. Informatics*, Vol. 14, No. 10, pp. 4497–4506, 2018.

[27] M. Agarwal and G. M. S. Srivastava, "Opposition-based learning inspired particle swarm optimization (OPSO) scheme for task scheduling problem in cloud computing", *J. Ambient Intell. Humaniz. Comput.*, Vol. 12, No. 10, pp. 9855–9875, 2021.

[28] J. Yang and G. Zhang, "Dynamic Dual-Threshold Virtual Machine Merging Method Based on Three-Way Decision", *Symmetry (Basel).*, Vol. 14, No. 9, p. 1865, 2022.
[29] H. Singh, S. Tyagi, and P. Kumar, "Comparative analysis of various simulation tools used in a cloud environment for task-resource mapping", In: *Proc of the International Conference on Paradigms of Computing, Communication and Data Sciences*, pp. 419–430, 2021.
[30] A. Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures", *Data*, Vol. 3, No. 4, p. 38, 2018.
[31] S. K. Panda and P. K. Jana, "Normalization-based task scheduling algorithms for heterogeneous multi-cloud environment", *Inf. Syst. Front.*, Vol. 20, pp. 373–399, 2018.