Received: February 21, 2023. Revised: March 9, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Deadline and Cost Aware Dynamic Task Scheduling in Cloud Computing Based on Stackelberg Game

Ahmed R. Kadhim[1]* **Furkan Rabee[1]**

[1]*Computer Science Department, Faculty of Computer Science and Mathematics, University of Kufa, Najaf, Iraq*
* Corresponding author's Email: ahmedr.alkhafajee@uokufa.edu.iq

Abstract: Cloud computing has become an essential technology in many industries due to its scalability and cost-effectiveness. The dynamic nature of cloud computing, including elasticity, on-demand provisioning, diverse resource types, and varied pricing model, presents a significant challenge in scheduling tasks for cloud-based systems, especially when considering user quality of service (QoS) constraints such as deadlines and budgets. Therefore, in order to optimize the performance of the cloud systems and end-user satisfaction, an efficient budget and deadline-aware scheduling model is necessary. Game theory provides a framework for modeling and analyzing the strategic interactions between self-interested entities, which makes it an ideal tool for task scheduling in cloud computing. Additionally, the versatility of game models enables the analysis of various cloud computing architectures. This paper proposes a dynamic Stackelberg (leader-follower) game model for modeling the interactions between tasks, scheduler, and cloud resources to find an equilibrium for the game under both budget and deadline constraints. The proposed dynamic task scheduling based on Stackelberg game (DTSSG) model is assisted by the pricing model and satisfaction factors to select the optimal virtual machine for processing the user task. To achieve high average resource utilization, the utilization factor of the cloud resources is considered in the proposed work. Experimental results show that the Stackelberg model equilibrium has been very effective in scheduling the user tasks across the data center resources by selecting the optimal virtual machines. The results demonstrate improved execution efficiency in terms of decreased makespan by 30%, reduced number of deadline violations by 52%, decreased total gain cost by 27.13% and increased provider profit by 19.15 % on average as compared to existing deadline budget scheduling (DBS), genetic algorithm, and MAX–MIN methods. Also, the results show the effectiveness of the proposed work in terms of increased throughput by 59.4 % and decreased makespan by 27.95% using Google cloud jobs dataset (GoCJ) as compared to existing gradient-based optimization (GBO), multi-verse optimizer (MVO), enhanced multi-verse optimizer (EMVO).

Keywords: Cloud computing, Game theory, Dynamic task scheduling, Stackelberg game, Makespan.

1. Introduction

A cloud computing system is composed of a number of servers that are located in remote places and can be accessed through the internet. These servers offer a variety of cloud services such as software, platform, and infrastructure services [1]. Cloud data centers offer computing and storage capabilities on a pay-per-use subscription model by utilizing a huge number of physical computers [2]. Cloud computing infrastructures can process a broad range of applications data, especially the data related to real-time applications like the internet of things (IoT), and mobile crowd sensing (MCS) applications which present a compelling case for executing their data in the cloud environment [3, 4]. Scheduling cloud data requires a lot of computation, storage, and communication costs. task scheduling is the process of assigning the user tasks to the data center resources in such a way the execution will be completed within the user's specified constraints such as budget and deadline [5]. In cloud computing, the cloud service provider (CSP) offers resources with different capabilities and prices. Typically, faster resources cost more than slower resources. Therefore, different plans for scheduling the same data load that utilize different resources may result in different completion times and costs. So, introducing an efficient task scheduling solution for cloud data centers requires satisfying both cost and time constraints according to the user's specifications [6]. Time constraint ensures that the user task is executed within the deadline given by the user and the cost constraint ensures that the user budget is not exceeded. A good scheduling algorithm attempts to achieve a solution that is nearly optimal by balancing these two values [7]. Game theory is a mathematical framework for modeling and analyzing strategic interactions between intelligent and rational decision-makers [8, 9]. In the context of task scheduling, game theory can be used to model the interactions between different tasks and resources, and to analyze the optimal strategies for task scheduling in different scenarios. Some common applications of game theory in task scheduling include the design of efficient scheduling algorithms, the analysis of the performance of different scheduling policies, and the study of the impact of different system parameters on task scheduling [10]. Equilibrium is a concept of game theory in which all participant reaches their optimal outcome. [8, 11]. In task scheduling for a cloud data center, a Nash equilibrium is a state in which no cloud computing resource (such as a virtual machine) has an incentive to change its task allocation, given the task allocations of the other resources. This means that each resource is operating at its optimal level of efficiency, given the actions of the other resources.

In this paper, a Stackelberg game is used to model the interactions between the task scheduler and resources. A Stackelberg game is a type of game in which one player, called the leader, makes the first move before the other players, called the followers [12-14]. In the proposed model, the task scheduler represents the leader while the virtual machines are the followers. In this setting, the leader chooses their best strategy first, and then the followers (virtual machines) respond by choosing their own strategies. The objective of the followers is to maximize their utilization, while the objective of the leader is to maximize profit and minimize the makespan of the system under the deadline and budget constraints imposed by the user tasks. The proposed dynamic task scheduling based on Stackelberg game (DTSSG) measures makespan, throughput, number of tasks violated their deadlines, total gain cost, and provider profit. DTSSG simulated using cloudsim plus environment. In summary, the main contributions of this paper are as follows.

- We model the scenario of task scheduling to various cloud resources as a Stackelberg game. The model consists of one leader (scheduler) and many followers (virtual machines). The optimization of the leader and follower are formulated mathematically.
- A new pricing model is introduced with encouraging and discouraging price functions to effectively utilize the system resources.
- We introduce the satisfaction factors for both deadline and budget constraints related to the user task, then aggregate the virtual machines that maintain these constraints for processing the task.
- We propose a dynamic task scheduling model that depends on the current utilization of the virtual machines at the decisive moment to maximize the average resource utilization.

The rest of the paper is structured as follows. Section 2 presents the related work in this field, section 3 introduces the system model, mathematical modeling, the proposed scheduling algorithm, and evaluation metrics modeling. Section 4 describes the experimentation results, performance evaluation, and discussion. Finally, the paper is concluded in section 5.

2. Related work

Scheduling tasks across the resources of a cloud data center becomes a very challenging problem due to the fact that there are many metrics, including makespan and resource utilization, that affect the scheduling. Also, user task constraints like budget and deadline must be taken into account. many researchers studied task scheduling and tried to improve the system's performance under these constraints. Some of them adopt game theoretical approaches to model the problem of task scheduling and resource allocation under the cloud environment.

Arabnejad et al. [15] proposed deadline-budget constrained scheduling (DBCS). The authors tried to optimize the scheduling algorithm in terms of time. they take into account the user quality of service (QoS) parameters like cost and time which represent the main issues in their work. DBCS looked for a radical schedule mapping that met the user's deadline and budget constraints.

Jeny Varghese et al. [16] proposed an entropy-based monotonic task scheduling with dynamic resource allocation. They aimed to complete the task within the deadline and improve response time, execution time, and use of resources. The work includes the processes of DC clustering, virtual machine (VM) clustering, resource mapping, and task scheduling. The clustering processes are managed in a federated cloud, the brokers are responsible for allocating the resources, and the task scheduler maps the tasks.

Xingwang Huang et al. [17] presented a novel approach to task scheduling in cloud computing that is based on gradient-based optimization (GBO). This technique is chosen for its faster convergence rate and ability to prevent getting stuck in local optima. The author's goal is to enhance the performance of a system with specific computing resources by reducing the makespan.

Liu et al. [18] introduced an implementation of a strategy that combines hardware and software to optimize energy usage and meet time constraints. The system is designed to adjust hardware features based on software requirements, resulting in more efficient execution and lower energy costs. On the software side, the paper explores a task-scheduling algorithm that is both energy-efficient and deadline-aware, using the Q-learning approach.

Natesan et al. [19] proposed a performance-cost grey wolfoptimization (PCGWO) algorithm to optimize the process of scheduling the user tasks to the cloud resources. their main objective is to reduce both processing time and cost of tasks under deadline constraints.

Jing et al. [20] proposed QoS-aware discrete particle swarm optimization (QoS-DPSO) for the optimization of system reliability under budget and deadline constraints. QoS-DPSO is a fault-tolerant scheduling algorithm that meets the service quality requirements of users.

Velliangiri et al. [21] combined electro search with a genetic algorithm and presented HESGA To optimize multi-cloud QoS parameters such as makespan and cost. According to simulation results, HESGA outperforms other methods.

Mokhtar A. Alworafi, and Suresha Mallappa [22] introduced deadline budget scheduling (DBS) as a model capable of scheduling tasks across heterogeneous cloud environments with the user QoS constraints: cost and time while maintaining the satisfaction of user tasks. The most important aspects of the proposed DBS model are minimizing the makespan under the user-specified deadline and minimizing cost without exceeding the user-specified budget.

Sarah E. Shukri et al. [23] introduced an improved version of the multi-verse optimizer (MVO) that is specifically designed to solve task scheduling problems in cloud computing environments. The new algorithm incorporates a novel operation that saves the best solutions at each iteration and reintroduces them as a new solution after a certain number of iterations. The primary focus of the proposed approach is to minimize task execution time while taking into account factors such as task length, cost, and power requirements.

R. Swathy et al. [24] introduced a game-theoretical model based on the Stackelberg (leader-follower) framework, which is enhanced with a satisfaction factor to enable the selection of the best physical host for deploying incoming tasks in a balanced manner at a data center. The model is designed to optimize the deployment process by ensuring assigning of tasks to the appropriate host.

Authors in [25] proposed a non-cooperative and cooperative game as a game-theoretic approach for real-time task scheduling in a cloud computing environment. All user actions have been considered as game players, and virtual machines have been considered as game strategies. They also compared the experimental results from the cooperative and non-cooperative games. The results demonstrate that the cooperative game model for task scheduling outperforms a non-cooperative game model when the payoff is taken as completion time and waiting time.

Previous studies have not considered the utilization status of system resources dynamically during the scheduling process. Also, most of the studies did not address satisfying user quality of service (QoS) requirements in terms of cost and deadline. To address these gaps, we proposed a dynamic scheduling model that guarantees meeting user requirements by considering both cost and deadline satisfaction factors. Furthermore, our model ensures optimal task scheduling by taking into account the status of cloud resources and incorporating an effective pricing model which relies on the current utilization of system resources. By considering these factors, our proposed model guarantees optimal resource utilization, meeting user QoS requirements, maximizing provider profit, and improving the overall performance of the system.

3. Problem formulation

3.1 Proposed scheduling framework

Cloud computing users expect their tasks to be scheduled efficiently and priced optimally within the given deadline. When users submit their tasks, these

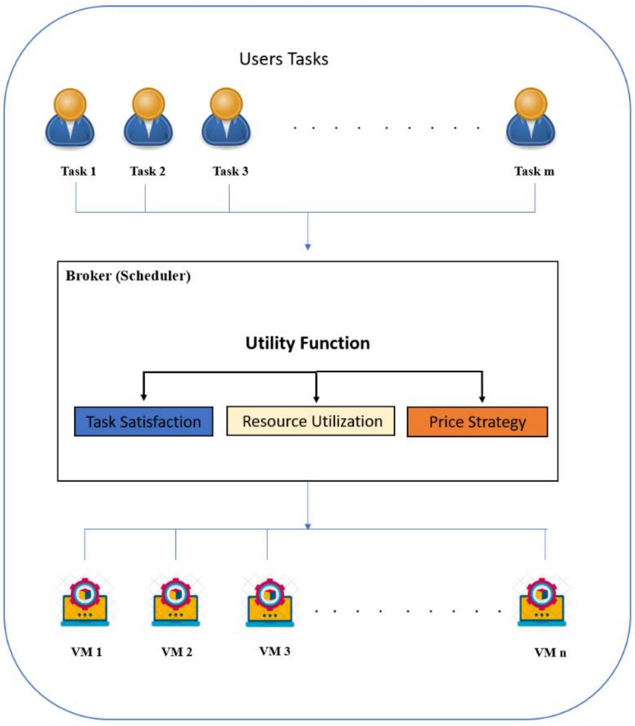


Figure. 1 The proposed scheduling framework

tasks come with several constraints like the deadline and budget. Therefore, in order to achieve good scheduling for user tasks in cloud data centers, it is essential to develop an optimal model for task scheduling. Stackelberg's game theoretical model informs the proposed solution, which maps the user tasks to different VMs in the cloud data center optimally. Fig. 1 highlights the architectural diagram of the Stackelberg game scheduler. The users submit their task requests with the deadline and budget constraints to the leader (scheduler), and the scheduler then deploys these requests among the followers (VMs) that have the necessary computing capabilities to process the user tasks. The utility function is computed using the satisfaction factor, pricing strategy, and current utilization of the VMs. This utility is calculated for every dynamically arrived task then the task is distributed to the optimal VM with the lowest utility value. The satisfaction factor ensures that the VM can satisfy the deadline and budget constraints of the task. While the pricing strategy tries to maximize the average utilization of the data center resources

3.2 Mathematical formulation of the proposed Stackelberg game model

The problem of the proposed model is formulated as follows: given a cloud data center that receives a set (T) of M tasks from the users, $T = t_1, t_2, \ldots, t_M$, each task is represented by three main parameters $t_j = (\ell_j, \text{d}_j, \text{b}_j)$, where ℓ_j indicate the

Table 1. Used notation

Notations	Description
T	Set of Tasks
V	Set of Virtual Machines
N	Number of VMs in V
M	Number of Tasks in T
t_j	The j^{th} task in T
v_i	The i^{th} VM in V
ℓ_j	Length of t_j
b_j	Budget of t_j
d_j	Deadline of t_j
$mips_i$	Million Instructions per Second of v_i
u_i	Current utilization of v_i
s_i	Available Storage of v_i
m_i	The available Memory size of v_i
SF_c	Cost satisfaction factor
SF_d	Deadline satisfaction factor
p_i^s	Subscription Cost on a specific VM
p_i^a	Actual cost (cost per second) using v_i
p_i^{base}	The base price to subscribe on v_i
$d_{i,j}$	Execution Time of t_j on v_i
pes_i	Number of Processing elements in v_i
ETv_i	Total execution time for v_i
Nov	The number of tasks violated their deadline

task length, d_j is the deadline allowed to execute the task, and b_j represents the cost given by the user to execute their task. Also, the data center has set (V) of N virtual machines where $V = \{v_1, v_2, \ldots, v_N\}$, each virtual machine is represented by its available CPU million instructions per second (MIPS), memory, storage, and VM utilization status where $v_i = (mips_i, m_i, s_i, u_i)$. One of the data center nodes is chosen to work as a task scheduler. The task scheduler deploys the tasks to different virtual machines based on the Stackelberg game model. The Stackelberg game model represents a leadership model. The problem applies to situations with one leader and many followers. Here, the selected task scheduler serves as the leader, while all other virtual machines serve as the followers. The leader periodically receives the available resources of the data center virtual machines from the CSP. The amount of remaining resources includes the CPU MIPS, Storage, memory, and the current utilization of the VMs. The leader (scheduler) also receives m number of user tasks with different deadlines and budget constraints. These tasks need to be mapped to optimal virtual machines in the data center. Our work proposes a dynamic scheduling approach to schedule the received tasks to the most suitable VM under user QoS constraints. Table 1 shows the notation used to represent the proposed model.

3.2.1. Scheduler—leader

All virtual machines in cloud data center announce their available amount of resources to the leader (scheduler). The remaining amount of resources includes the available CPU, RAM, storage, and VM utilization for the set V of VMs. The price strategy for the resources of the data center is determined by the CSP. Based on the leader's strategy, the followers are assigned tasks for processing. The maximum price will be chosen for the VM with the high utilization ratio, and the minimum price will be chosen for the VM with the lower utilization ratio. Let $P = \{p1, p2, \dots\dots, pN\}$ the price strategies of the followers based on the price functions, the price strategies p is determined by the CSP for each virtual machine as in Eq. (1).

$$p_i = p_i^s + p_i^a \tag{1}$$

Where P_i^s is the subscription cost for a specific VM with index $i = \{1,2,3 \dots\dots N\}$, this cost depends on the current utilization of the VM. The subscription pricing function is described in Eq. (2) which either encourages or discourages utilizing VM based on it is current utilization status. Each virtual machine has a desired utilization range $[u_i^{low}, u_i^{high}]$ in this approach. The intention is to charge a base price (P_i^{base}) for the virtual machine v_i if it is utilization (u_i) falls within this range. An additional price is added to the base price to discourage utilizing the VM if the utilization u_i is above u_i^{high} and less than the threshold value $u_i^{thre.}$ (if the VM utilization reaches the threshold value, it cannot be used). Alternatively, a certain price is discounted from the base price to encourage usage if the VM utilization u_i is below U_i^{low}. In our work, the values of u_i^{low}, u_i^{high}, and $u_i^{thre.}$ are selected to be 30%, 70%, and 95% respectively.

$$p_i^s = \begin{cases} p_i^{base} - p_i^{encourage}, & 0 \le u_i \le u_i^{low} \\ p_i^{base}, & u_i^{low} < u_i < u_i^{high} \\ p_i^{base} + p_i^{discourage}, & u_i^{high} \le u_i \le u_i^{thre.} \end{cases} \tag{2}$$

The encouraging and discouraging prices are linearly varying functions of utilization. If VM utilization is slightly higher or lower than the desired range, the overall price should be close to the base price. Hence, both $p_i^{encourage}$ and $p_i^{discourage}$ prices must be relatively small as compared to the base price (p_i^{base}). As utilization moves far from the desired range, both prices should rapidly increase. the base price is set to be 0.5 in our simulation. The encouraging and discouraging prices formulas described in Eqs. (3) and (4).

$$p_i^{encourage} = p_i^{base} \times (u_i^{low} - u_i) \tag{3}$$

$$p_i^{discourage} = p_i^{base} \times (u_i - u_i^{high}) \tag{4}$$

The actual processing price (p_i^a) represent the cost per second for using VM resources (CPU, Memory, and Storage), it is calculated as in Eq. (5).

$$p_i^a = \left[c_i^{cpu} + c_i^{ram} + c_i^{storage}\right] \tag{5}$$

The execution time needed to complete the task t_j on each v_i is referred by $d_{i,j}$ and is computed by the leader as in Eq. (6).

$$d_{i,j} = \frac{\ell_j}{mips_i \times pes_i} \tag{6}$$

Each task comes with a deadline $\underline{\mathrm{d}}_j$ and budget $\underline{\mathrm{b}}_j$. First of all, the proposed approach aggregates the VMs suitable for processing the task by checking the deadline and cost satisfaction factors (SF_c *and* SF_d). The VMs that have both SF_c *and* SF_d greater than zero will be passed to the mapping stage. This step ensures that the task will be processed within the specified deadline and will not exceed the user budget see Eqs. (7) and (8).

$$SF_c = 1 - e^{\left(1 - \left(\frac{\mathrm{b}}{p}\right)\right)} \tag{7}$$

$$SF_d = 1 - e^{\left(1 - \left(\frac{\mathrm{d}}{d}\right)\right)} \tag{8}$$

Then, the satisfaction factor SF is calculated as in Eq. (9) to be used in the utility function of the followers.

$$SF = (\alpha \times SF_c) + (\beta \times SF_d) \; where \; \alpha + \beta = 1 \tag{9}$$

The values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ coefficients in Eq. (9) are tested under different conditions in the simulation, and the result is shown in Table 2.

The scheduler calculates these values for each

Received: February 21, 2023. Revised: March 9, 2023.

Table 2. Different values of α and β coefficients

SF_c	α	β
$SF_c \leq 0.25$	0.2	0.8
$0.25 < SF_c \leq 0.5$	0.5	0.5
$0.5 < SF_c \leq 0.75$	0.7	0.3
$SF_c > 0.75$	0.9	0.1

incoming task. This behavior aims to maximize the profit of the leader keeping in mind the deadline constraint is already satisfied. In fact, using different values of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ coefficients in different conditions ensures that the task will be scheduled to the VM which gives the system a high profit.

The utility function of the followers will be calculated only for the VMs that give a positive SF_c and SF_d. This function is calculated based on the satisfaction factor SF and the total cost to execute the task t_j on v_i as in Eq. (10).

$$UF_v(p,d) = (p_i \times d_{i,j}) + [SF]_n \qquad (10)$$

The proposed model assumes that the utility function of the followers equally depends on both the SF and the total cost. Therefore, $[SF]_n$ represents the normalized value of the SF. The value of SF is normalized using the MIN-MAX normalization technique to fall within the range $[0 - (p_i * d_{i,j})]$. The optimal virtual machine for processing the user task is described in Eq. (11).

$$Follower\ best\ strategy = Min\{UF_v\} \qquad (11)$$

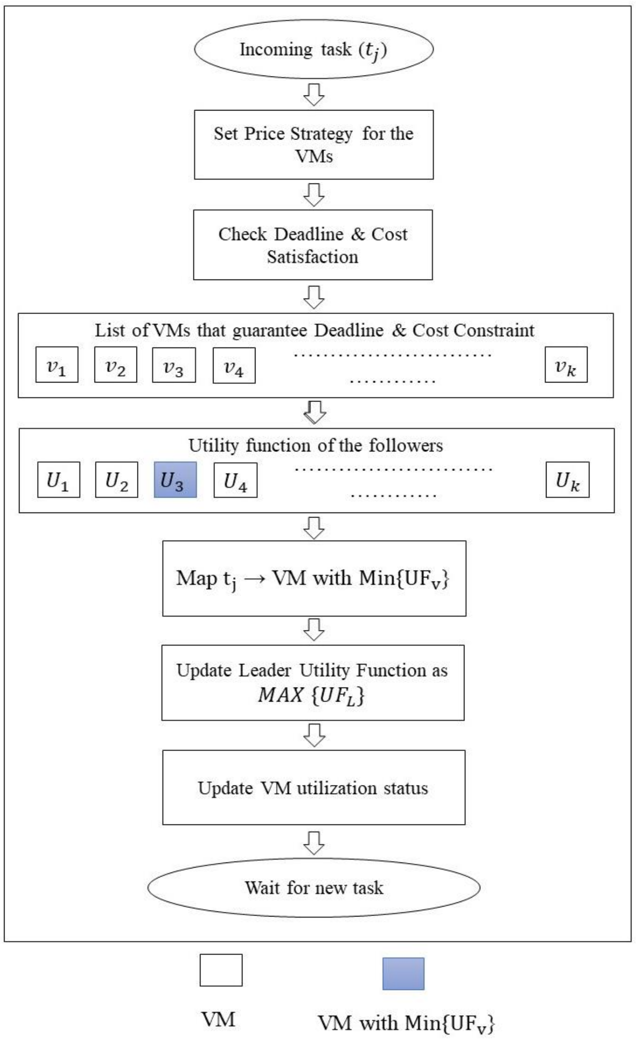
The utility function of the leader (scheduler) is formulated as the total gain acquired through processing the successful tasks in the data center. Thus, the utility function is given in Eq. (12).

$$UF_L(p,d) = \sum_{i=1}^{n}\sum_{j=1}^{m}(p_{i,j} - p_{i,j}^{s}) \times d_{i,j} \times M_{i,j} \qquad (12)$$

Where $M_{i,j}$ is a Boolean value activated only when a task t_j mapped to VM v_i, the best strategy for the leader that ensures the task is mapped to the most suitable VM is shown in Eq. (13).

$$Leader\ best\ strategy = MAX\ \{UF_L\} \qquad (13)$$

The total gain cost of the leader is computed based on the follower's best strategy. The leader utility value is calculated as the summation of the cost awarded for processing all the tasks. This utility



Incoming task (t_j)
Set Price Strategy for the VMs
Check Deadline & Cost Satisfaction
List of VMs that guarantee Deadline & Cost Constraint
v_1 v_2 v_3 v_4 ... v_k
Utility function of the followers
U_1 U_2 U_3 U_4 ... U_k
Map $t_j \rightarrow$ VM with Min{UF_v}
Update Leader Utility Function as $MAX\ \{UF_L\}$
Update VM utilization status
Wait for new task
VM
VM with Min{UF_v}

Figure. 2 Flow chart of the proposed model

is maximized after scheduling a task indicating that the user tasks are assigned to VMs optimally.

3.2.2. Followers (virtual machines)

The followers choose their best response strategies by choosing the strategy that offers the lowest utility function value. The follower accepts the task request and executes it based on its best strategy. The flow chart of the proposed system is given in Fig. 2.

3.3 Stackelberg game scheduling model

Following is the sequence of procedures for dispatching tasks according to the Stackelberg game model:

1. The leader determines the price strategy of every individual VM according to the price strategy function. Then, it calculates the UF_v for the VMS and finally, the utility strategy values are announced to the followers.

Received: February 21, 2023. Revised: March 9, 2023.

ALGORITHM 1: DTSSG ALGORITHM

```
   Inputs: an available set of VMs V with their status ($mips_i, m_i, s_i, u_i$),
   task $t_j$ with its length, Deadline, and budget ( $\ell_j$ , $\underline{b}_j, and\ \underline{d}_j$);.
   Output: scheduler (Leader) optimal strategy, VMs (Followers) optimal strategy.
1  V= List of available VMs; $t_j$ = user task;
2  Set price strategy for the follower (VMs) based on their utilization ($u_i$) $P = \{p_1, p_2, p_s, \ldots., p_N\}$ as:
3  $p_i = p_i^s + p_i^a$ , where
4  $p_i^a = [c_i^{cpu} + c_i^{ram} + c_i^{storage}]$
5  $p_i^s = \begin{cases} p_i^{base} - p_i^{encourage}, & 0 \le u_i \le u_i^{low} \\ p_i^{base}, & u_i^{low} < u_i < u_i^{high} \\ p_i^{base} + p_i^{discourage}, & u_i^{high} \le u_i \le u_i^{thre.} \end{cases}$
6  for each VM $vi \in V$ do
7      | Calculate execution time for the task $t_j$ as:
8      | $d_{i,j} = \frac{\ell_j}{mips_i \times pes_i}$
9      | Check Deadline and cost satisfaction for $t_j$ as:
10     | $SF_c = 1 - e^{\left(1-\left(\frac{\underline{b}}{p}\right)\right)}$
11     | $SF_d = 1 - e^{\left(1-\left(\frac{\underline{d}}{d}\right)\right)}$
12     | if ($SF_d > 0\ AND\ SF_c > 0$)
13     |     $V_{sf}.add\ (vi)$
1      | End if
15 End for
16 for each $vi \in V_{sf}$ do
17     | Choose appropriate values of α and β according to table 2.
18     | Calculate SF as:
19     | $SF = (\alpha \times SF_c) + (\beta \times SF_d)$
20     | Calculate the Utility function of the VM as:
21     | $UF_v\ i,j = P_i \times d_{i,j} + [SF]_n$
22 End for
23 Find $Min\{UF_v\}$ as the optimal strategy for the follower
24 Map $t_j \rightarrow$ VM with $Min\{UF_v\}$
25 Compute $UF_L(p,d) = \sum_{i=1}^{n} \sum_{j=1}^{m} (p_{i,j} - p_{i,j}^s) \times d_{i,j} \times M_{i,j}$ for the scheduler
26 The optimal strategy of the broker is given by $MAX\ \{UF_L\}$
27 Update VM utilization status.
28 Repeat the above steps for each incoming task $t_j$.
```

Figure. 3 Pseudo-code of DTSSG algorithm

2. The follower (VM) selects its best response strategy as $Min\{UF_v\}$. This strategy keeps the VM utilization (u_i) to stay within the desired utilization range [u_i^{low},u_i^{high}].
3. After the follower selects its best response strategy, the leader selects its optimal strategy as $MAX\ \{UF_L\}$. The leader then updates the pricing strategies for the followers based on their updated utilization status.

Steps 1 to 3 are repeated for each incoming task.

3.4 Evaluation metrics modeling:

Five evaluation metrics are used to evaluate the proposed work. They are makespan, throughput, number of deadline violations, total gain cost, and total provider profit.

Makespan refers to the maximum amount of time needed to complete the execution of a set of tasks [26]. It is influenced by various factors including the size and complexity of the tasks, the number of resources available, the load on the system, and the algorithms used for task scheduling. Minimizing the makespan is important for optimizing the use of resources and improving the performance of the cloud data center. Makespan is represented mathematically as shown in Eq. (14) [26, 27].

$$makespan = max(ETv_i) \quad \forall\ i \in 1,2,3, \ldots m \qquad (14)$$

Where ETv_i is the execution time of all assigned tasks to the ith VM. As described in Eq. (15).

$$ETv_i = \sum_{j=1}^{m} d_{i,j} \times M_{i,j} \qquad (15)$$

The average makespan is computed as described in Eq. (16) [28].

$$Avg.\ Makespan = \frac{\sum_{i=1}^{N} ETv_i}{N} \qquad (16)$$

The second evaluation metric used to evaluate our work is throughput. It refers to the total number of tasks accomplished within a given makespan, throughput can be calculated as in Eq. (17).

$$Throughput = \frac{\sum M_{i,j}}{makesapn} \qquad (17)$$

The next evaluation metric is the number of deadline violations (NoV) which represents the total number of tasks that violates their deadline. The NoV is defined in Eq. (18) [29].

Table 3. simulation experimental environment

Parameter	value
Operating system	Windows 10
processor	Intel Core i5 2.4 GHz
RAM	8.00 GB Memory
Simulation environment	Cloudsim Plus
IDE Tool	Eclipse IDE
Tool Ver.	1.1

$$NoV = \sum_{j=1}^{m} TvD \qquad (18)$$

where TvD is a binary variable activated only when the specific task violates its deadline.

Another evaluation metric used in the experiment is the total gain cost, it refers to the cost gained by the data center for executing a set of tasks under specified budget and deadline constraints. the total gain cost is represented by the utility function of the leader (UF_L) and it is calculated as in Eq. (12).

The last evaluation metric is the provider profit, it represents the total profit that is summed from the fees charged to execute the successful tasks. Provider profit is calculated by subtracting the actual implementation costs for the successful tasks from their budgets. The provider profit is described in Eq. (19) [22].

$$Provider\ Profit = \sum_{j=1}^{m} (\underline{b}_j - p_j^a) \qquad (19)$$

4. Performance evaluation

The experimental setup, workload formation, as well as results, and discussions are presented in this section.

4.1 Experimental setup

The cloudsim plus simulator has been used to simulate and find out how well the proposed method works in practice. CloudSim plus is a popular open-source tool developed in Java that can simulate both the cloud environment and cloud-based services [30]. The experimental setup consists of an Intel Core i5 2.4 GHz CPU, 8.00 GB of memory, and a 512 GB hard drive, running on the Eclipse IDE 2021 R and CloudSim plus. To implement our model, we extended several classes of the CloudSim plus simulator, including Cloudlet.java, VM.java, Datacenter.java, and DatacenterBrokerAbstract.java. Also, we created some classes for the proposed work as needed. Table 3 shows the simulation experimental environment Configuration properties.

Two simulation scenarios were studied in our implementation with different simulation parameters to analyze the performance of the proposed DTSSG model.

Table 4. VM, tasks, hosts, and data center configurations for scenario 1

Entity type	Parameters	Value
Data centers	Number of data centers	1
Hosts	Number of hosts	2
	PE per Host	$2, 4$
	PE speed	$10000\ MIPS$
	RAM	$16\ GB$
	Host storage	$10\ GB$
	Host bandwidth	$100\ GB/s$
Virtual machines	Number of VMs	$5, 7, 9, 11$
	No. of PE	1 Per each VM
	PE speed	(500,1000,2000,3000) MIPS
	VM RAM	512MB
	Bandwidth	$1\ GB/s$
	Type of policy	Time-shared
	Costs for using CPU, RAM, and Storage	(0.1, 0.1, 0.1)
Tasks	Number of tasks	$(250, 500, 750, 1000)$
	Length	$[300 - 3000]$ MIPS
	File size	$[100 - 300]$MB
	Output size	$[20 - 40]$MB

Scenario 1: In this scenario, the experimental environment included one data center with two physical machines (host). Each host has (16) GB of RAM, (1) TB of storage, (100) GB/s of bandwidth, and a time–shared scheduling policy. One of the host machines is a quad-core with the other being a dual-core machine, both built on the X86 architecture, operating on a Linux system, and equipped with a Xen virtual machine monitor (VMM) that can process at a rate of 10,000 MIPS. To conduct the experiments for this scenario, a number of virtual machines are created (5, 7, 9, 11). Each VM has an image size of (10) GB, (0.5) GB of memory, (1) GB/s of bandwidth, one processing element, and a MIPS value of (500, 1000, 2000, and 3000). The configuration includes Xen VMM architecture and Time-Shared scheduling policy. Tasks are created in different lengths, different input and output file sizes, and with different task numbers for each experiment (250, 500, 750, and

Table 5. GOCJ tasks distribution

No.	Job Type	Job Size (MI)	Distribution
1	Small	15,000 - 55,000	20%
2	Medium	59,000 - 99,000	40%
3	Large	101,000 - 135,000	30%
4	Extra large	150,000 - 335,000	6%
5	Huge	525,000-900,000	4%

Table 6. GOCJ dataset groups

Number of Tasks	Task Group
100	Regular-Size
200	
300	
400	
500	
600	
700	Big-Size
800	
900	
1000	

1000). The deadline and budget values associated with each task are produced according to the procedure followed in [5]. Table 4 shows the configuration for Cloudsim plus VM, tasks, hosts, and data centers.

Scenario 2: for conducting this experiment, the Google cloud jobs dataset (GoCJ) is used. The key advantage of selecting this particular dataset was its ability to provide real-time information about user requests, thus enabling the researchers to gain insight into actual user behavior. The GoCJ dataset was created by aggregating data from MapReduce logs and Google cluster traces acquired from the M45 supercomputer cluster [31]. The repository of the GoCJ dataset comprises multiple collections of text files, each containing a specific number of tasks.

The GOCJ dataset files are categorized into two groups: the regular-size dataset and the large-size dataset as mentioned in [32] (refer to Table 6). The regular dataset contains between 100 and 600 tasks, and 50 VMs are used for this dataset with MIPS ranging from 100 to 4000. The large-size dataset, on the other hand, contains between 700 and 1000 tasks, and 100 VMs are used with MIPS ranging from 100 to 4000.

4.2 Simulation results

This section introduces the experimental results of our work. The proposed scheduling algorithm (DTSSG) has been evaluated and compared using a number of inertial parameters. Some of these evaluation criteria are Makespan, throughput, number of deadline violations, total gain cost, and Provider profit. The performance of the DTSSG task scheduling approach is measured using Eqs. (16), (17), (18), (12), and (19), respectively as described in section 3.4.

4.2.1. Experimental results for Scenario 1

To evaluate the performance in this experimental scenario, we consider 250, 500, 750, and 1000 tasks. with various numbers of virtual machines such as 5, 7, 9, and 11 for all the evaluation metrics. The proposed method results are compared with DBS [22], GA [22], and MAX–MIN [22] for this scenario.

i. Makespan evaluation:

In this section, the performance of the task's execution time of the proposed work is evaluated. The proposed method makespan results are compared with DBS, GA, and MAX–MIN. Fig. 4. clearly shows a reduction in makespan as compared with the other algorithms. It can be noticed from the figure that the DTSSG approach results in a makespan reduction of 25.1%, 51.7%, and 26.03% over DBS, GA, and MAX-MIN methods respectively for scheduling 250 tasks. It can also be observed that a reduction of 17.1%, 51.8%, and 21.9% in makespan has been achieved using the DTSSG approach over DBS, GA, and MAX-MIN methods respectively for scheduling 500 tasks. When the number of tasks is 750 the DTSSG approach exhibits 13.8%, 47.6%, and 17.8% reduction in makespan over DBS, GA, and MAX–MIN respectively. finally, when we simulate our algorithm on 1000 tasks, the reduction in makespan is 16.3%, 51.4%, and 23% over DBS, GA, and MAX–MIN respectively.

ii. Number of deadline violations

The number of deadline violations is evaluated for verifying the quality of service (QoS) of the scheduling algorithm. In this evaluation experiment, we also considered 250, 500, 750, and 1000 tasks.

Fig. 5 shows the results for the number of tasks that violated their deadlines for our approach as compared with DBS, GA, and MAX-MIN. it can be observed that the proposed DTSSG exhibits better performance by reducing the number of tasks that violated their deadlines by using the deadline satisfaction factor. This help to identify and select the optimal VM that gives a lower execution time for processing the task considering the deadline constraint. the number of deadlines violations decreased by 30%, 53.3%, and 61.1 over DBS, GA, and MAX-MIN methods respectively for 250 tasks.

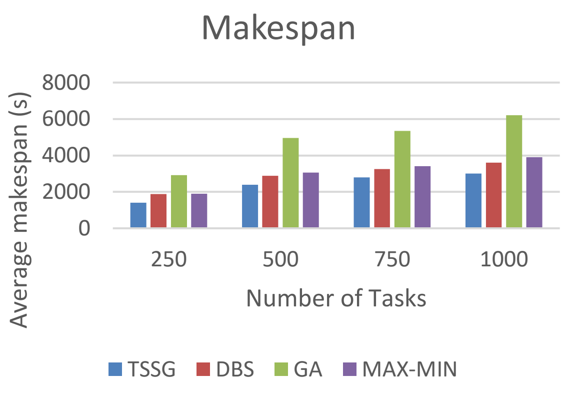


Figure. 4 Comparison of Makespan

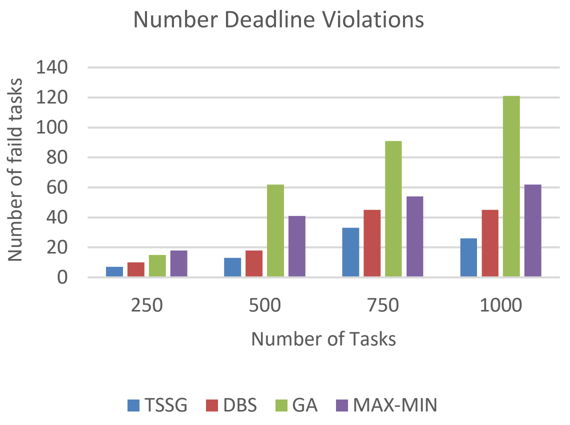


Figure. 5 Comparison of the number of deadline violations

For scheduling 500 tasks, the reduction is 27.78%, 79%, and 68.29% over DBS, GA, and MAX-MIN methods respectively. When the number of tasks is 750, the number of violations is decreased by 26.6%, 63.74%, and 38.89% over DBS, GA, and MAX-MIN methods respectively. Finally, the number of deadline violations decreased by 42.2%, 78.5%, and 58% as compared to DBS, GA, and MAX-MIN methods respectively.

iii. Total gain cost

Total gain cost is the actual cost for executing the successful tasks on the data center VMs under specified budget constraints. From Fig. 6, it can be shown that the lowest gain cost is achieved for executing the tasks with the DTSSG approach while the highest cost is for tasks executed with the MAX-MIN approach in all cases. DTSSG approach uses the cost satisfaction factor in the utility function to ensure that the task will be executed on the optimal VM. Using this factor guarantee that the execution cost of the task will not exceed the task budget and reduce the execution cost by utilizing the VM that gives the lowest cost.

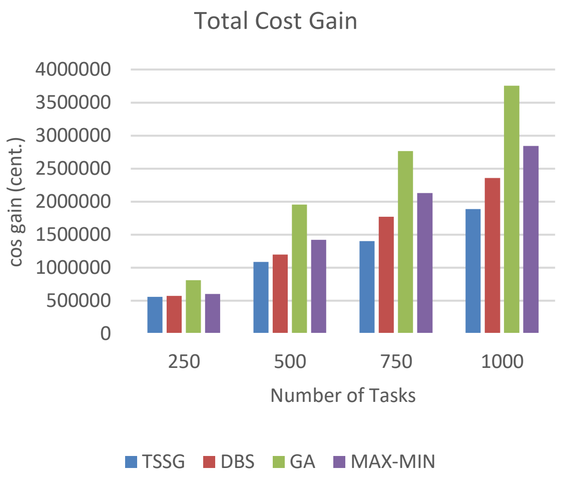


Figure. 6 Comparison of total gain cost

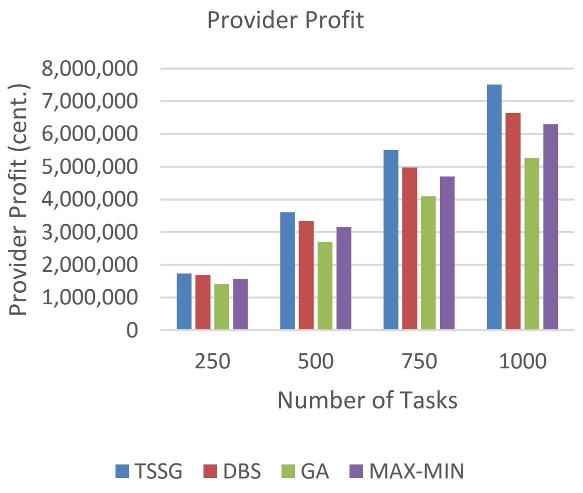


Figure. 7 Comparison of provider profit

iv. Provider profit:

The pivotal parameter for the cloud service provider is the profit, which is determined by the total revenue earned from completing the user tasks. As depicted in Fig. 7, the profits of the provider are compared to those of DBS, GA, and MAX-MIN. The results indicate that the proposed DTSSG approach outperforms all other methods in terms of provider profit.

The provider profit increased by 2.9%, 22%, and 10.9% over DBS, GA, and MAX-MIN respectively for scheduling 250 tasks. For 500 tasks, DTSSG outperforms in increasing the provider profit by 7.9%, 33%, and 14% over DBS, GA, and MAX-MIN methods respectively.

Scheduling 750 tasks with the proposed method results in increasing provider profit by 10.6%, 34,4%, and 17.2% over DBS, GA, and MAX-MIN methods respectively. Finally, for 1000 tasks the provider profit increased by 13%, 42.7%, and 19.2%

Table 7. Makespan results for regular-size datasets

	Makespan values			
No. of Tasks	DTSSG	EMVO	MVO	GBO
100	139	187.2	187.7	283.2
200	322	387.6	453.5	601.4
300	489	542.9	661	870.5
400	698	768.5	782.1	1105.5
500	803	875.4	1085.1	1426.9
600	974	1099	1286.7	2042.9

Table 8. Throughput results for regular-size datasets

	Throughput values			
No. of Tasks	DTSSG	EMVO	MVO	GBO
100	71.9%	53.4%	53.2%	35.3%
200	62.1%	51.5%	44.1%	33.2%
300	61.3%	55.2%	45.3%	34.4%
400	57.3%	52%	51.1%	36.1%
500	62.2%	57.1%	46%	35%
600	61.6%	54.5%	46.6%	29.3%

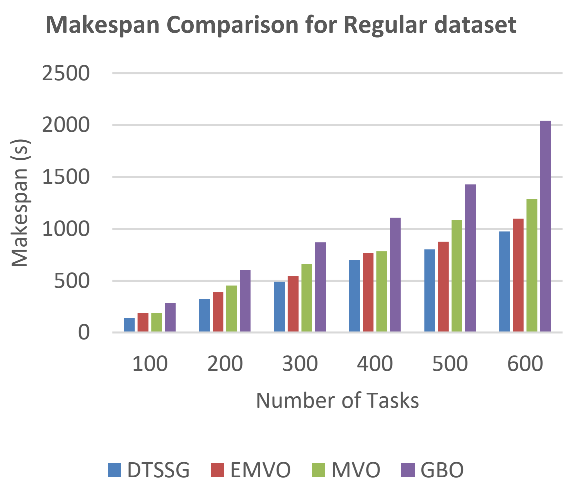


Figure. 8 Makespan comparison for regular-size datasets

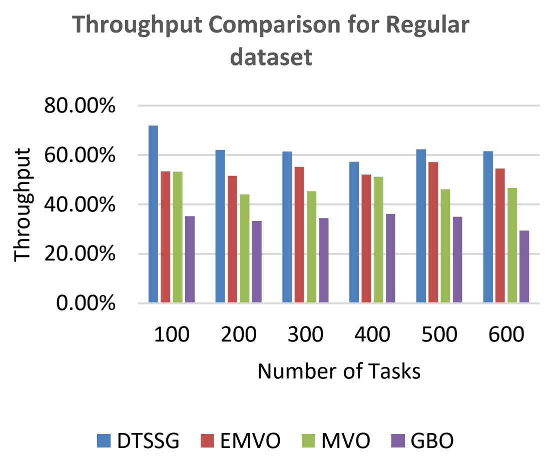


Figure. 9 Throughput comparison for regular-size dataset

over DBS, GA, and MAX-MIN methods respectively.

4.2.2. Experimental results for Scenario 2

This section presents the experimental results of the proposed method for both regular-size and large-size groups of the GOCJ dataset. We evaluate the performance of our method in terms of makespan, and throughput. The results are compared with EMVO [23], MVO [32], and GBO [17] methods in this scenario.

i. Regular-size dataset:

In this subsection, we introduce the makespan and throughput results of the regular dataset, the number of tasks used are 100, 200, 300, 400, 500, and 600. For these groups of tasks, 50 VM are used. Table 7 highlights the average makespan results for the regular datasets. The results indicate that the DTSSG method performed better than EMVO, MVO, and GBO methods across all datasets.

Fig. 8 illustrates the average makespan results of the DTSSG method as compared to other methods for regular-size datasets. The average makespan results obtained for this experiment are 570.8, 643.46, 742.72, and 1055.12 for the DTSSG method, to EMVO, MVO, and GBO respectively.

Also, throughput performance analysis for regular-size datasets is presented here. The obtained throughput results shown in table 8 clearly explain that the proposed method outperforms DTSSG, EMVO, MVO, and GBO in all experiments.

Fig. 9 illustrates the throughput performance for the regular dataset. The average throughput was obtained as 62.76%, 54%, 47.77%, and 33.93% for DTSSG, EMVO, MVO, and GBO methods respectively.

ii. Big-size dataset:

The experimental analysis for the large-size group of the GOCJ dataset is presented in this subsection. Similar to the previous experiment, the performance is measured using the makespan and throughput. We have considered the number of tasks to be 700, 800, 900, and 1000 for large-size datasets. These tasks are scheduled on 100 VMs.

Table 9 shows the comparative analysis for large-size datasets in terms of average makespan

Fig. 10 illustrates the graphical representation for makespan comparison. According to this

Table 9. Makespan results for large-size datasets

	Makespan values			
No. of tasks	DTSSG	EMVO	MVO	GBO
700	803	823.7	908.2	2280
800	896	933.1	1038.6	2489
900	923	969.7	1158.4	3071
1000	1089	1158.7	1275.7	3772

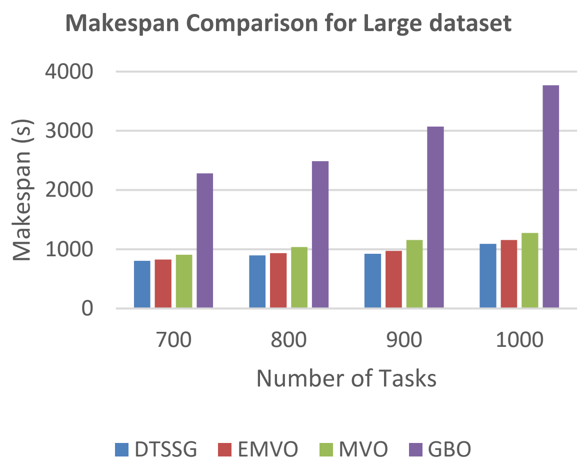


Figure. 10 Makespan omparison for large-size dataset

Table 10. Throughput results for regular-size datasets

	Throughput values			
No. of tasks	DTSSG	EMVO	MVO	GBO
700	87.1%	85%	77%	30.7%
800	89.2%	86%	77%	32.1%
900	97.5%	93%	78%	29.3%
1000	91.8%	86%	78%	26.5%

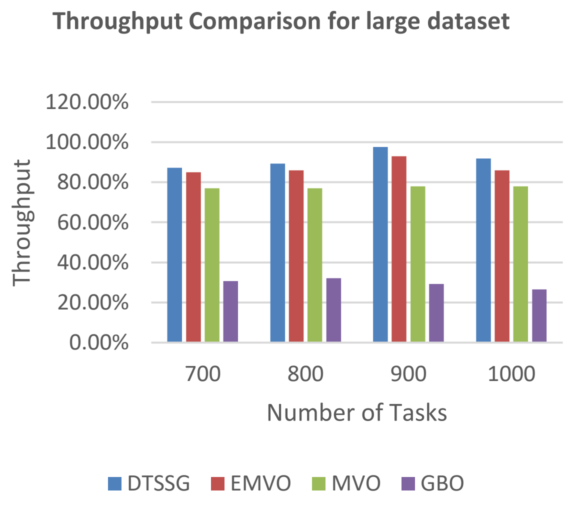


Figure. 11 Throughput comparison for large-size dataset

experiment, the average makespan values were obtained as 927.75, 971.36, 1095.26, and 2903.2, using the proposed approach, EMVO, MVO, and GBO respectively.

The obtained throughput results for the large dataset are compared with other existing techniques as shown in Table 10.

Fig. 11 illustrates the comparative analysis of throughput for the large-size dataset. The obtained average throughput results are 91.45%, 87.50%, 77.50%, and 29.6% for the proposed approach, EMVO, MVO, and GBO respectively.

5. Conclusion

This paper proposed a game theoretical-based scheduling model for the cloud data center. the proposed model uses Stackelberg (leader-flower) game model to effectively schedule user tasks under budget and deadline constraints. First, the favorable virtual machines that maintain the task cost and time constraints are filtered using the satisfaction factor and then the leader-follower strategy is used to identify the optimal virtual machine for processing the task. The proposed DTSSG method assigns tasks to suitable virtual machines based on the virtual machine's current utilization and pricing strategy. The proposed model benefits the virtual machines, and the user tasks, and maximizes the overall performance of the system. The simulation results have shown that (i) tasks are deployed to virtual machines in the data center at a faster rate. (ii) reduced the number of tasks that violates their deadlines by checking the deadline satisfaction before the scheduling process(iii) Increased throughput and decrease makespan values. Finally, the proposed model is observed to have better performance and increased system profit as compared to the state of art algorithms.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

Conceptualization, Furkan Rabee, and Ahmed R. Kadim; methodology, Ahmed R. Kadhim; software, Ahmed R. Kadhim; validation, Ahmed R. Kadim, and Furkan Rabee; formal analysis, Ahmed R. Kadim, and Furkan Rabee; investigation, Furkan Rabee, and Ahmed R. Kadim; resources, Furkan Rabee and Ahmed R. Kadim; writing—original draft preparation, Ahmed R. Kadhim; writing—review and editing, Ahmed R. Kadim; supervision, Furkan Rabee.

References

[1] J. Yang, L. Zhang, and X. A. Wang, "On Cloud Computing Middleware Architecture", In: *Proc. of 2015 10th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput. 3PGCIC 2015*, pp. 832–835, 2015.

[2] R. M. Singh, L. K. Awasthi, and G. Sikka, "Towards Metaheuristic Scheduling Techniques in Cloud and Fog: An Extensive Taxonomic Review", *ACM Comput. Surv*, Vol. 55, No. 3, pp. 1–43, 2023.

[3] S. Akter, T. N. Dao, and S. Yoon, "Time-Constrained Task Allocation and Worker Routing in Mobile Crowd-Sensing Using a Decomposition Technique and Deep Q-Learning", *IEEE Access*, Vol. 9, pp. 95808–95822, 2021.

[4] X. Ma, H. Gao, H. Xu, and M. Bian, "An IoT-based task scheduling optimization scheme considering the deadline and costaware scientific workflow for cloud computing", *EURASIP Journal on Wireless Communications and Networking*, No. 2019:249, 2019.

[5] Verma and S. Kaushal, "A hybrid multi-objective Particle Swarm Optimization for scientific workflow scheduling", *Parallel Comput*, Vol. 62, pp. 1–19, 2017.

[6] S. Pandey, "Scheduling and Management of Data Intensive Application Workflows in Grid and Cloud Computing Environments", *PhD Thesis*, Univ. Melbourne, Aust., No. December, p. 203, 2010.

[7] K. Chakravarthi and L. Shyamala, "TOPSIS inspired Budget and Deadline Aware Multi-Workflow Scheduling for Cloud computing", *Journal of Systems Architecture*, Vol. 114, p. 101916, 2021.

[8] B. Heydenreich, R. Müller, and M. Uetz, "Games and mechanism design in machine scheduling-an introduction", *Prod. Oper. Manag*, Vol. 16, No. 4, pp. 437–454, 2007.

[9] M. O. Agbaje, O. B. Ohwo, T. G. Ayanwola, and O. Olufunmilola, "A Survey of Game-Theoretic Approach for Resource Management in Cloud Computing", *Journal of Computer Networks and Communications*, Vol. 2022, 2022.

[10] S. H. H. Madni, M. S. A. Latiff, Y. Coulibaly, and S. M. Abdulhamid, "Recent advancements in resource allocation techniques for cloud computing environment: a systematic review", *Cluster Comput.*, Vol. 20, No. 3, pp. 2489–2533, 2017.

[11] X. Xu and H. Yu, "A game theory approach to fair and efficient resource allocation in cloud computing", *Math. Probl. Eng*, Vol. 2014, No. 1, 2014.

[12] Y. Liu, L. L. Njilla, J. Wang, and H. Song, "An LSTM Enabled Dynamic Stackelberg Game Theoretic Method for Resource Allocation in the Cloud", In: *Proc. of 2019 Int. Conf. Comput. Netw. Commun. ICNC 2019*, pp. 797–801, 2019.

[13] T. Roughgarden, "Stackelberg scheduling strategies", *SIAM J. Comput*, Vol. 33, No. 2, pp. 332–350, 2004.

[14] P. Y. Nie and P. A. Zhang, "A note on Stackelberg games", In: *Proc. of Chinese Control Decis. Conf. 2008, CCDC 2008*, pp. 1201–1203, 2008.

[15] H. Arabnejad, J. G. Barbosa, and R. Prodan, "Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources", *Futur. Gener. Comput. Syst.*, Vol. 55, pp. 29–40, 2016.

[16] 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.

[17] X. Huang, Y. Lin, Z. Zhang, X. Guo, and S. Su, "A gradient-based optimization approach for task scheduling problem in cloud computing", *Cluster Comput*, Vol. 25, No. 5, pp. 3481–3497, 2022.

[18] X. Liu, P. Liu, L. Hu, C. Zou, and Z. Cheng, "Energy-aware task scheduling with time constraint for heterogeneous cloud datacenters", *Concurrency and Computation: Practice and Experience*, Vol. 32, No. 18, pp. 260–267, 2020.

[19] G. Natesan and A. Chokkalingam, "An improved grey wolf optimization algorithm based task scheduling in cloud computing environment", *Int. Arab J. Inf. Technol.*, Vol. 17, No. 1, pp. 73–81, 2020.

[20] W. Jing, C. Zhao, Q. Miao, H. Song, and G. Chen, "QoS-DPSO: QoS-aware Task Scheduling for Cloud Computing System", *Journal of Network and Systems Management*, Vol. 29, No. 1, pp. 1–29, 2021.

[21] S. Velliangiri, P. Karthikeyan, V. M. Arul 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.

[22] 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.

[23] S. E. Shukri, R. A. Sayyed, A. Hudaib, and S. Mirjalili, "Enhanced multi-verse optimizer for task scheduling in cloud computing environments", *Expert Systems with Applications*, Vol. 168, No. February 2020, 2021.

[24] R. Swathy, B. Vinayagasundaram, G. Rajesh, A. Nayyar, M. Abouhawwash, and M. A. Elsoud, "Game theoretical approach for load balancing using SGMLB model in cloud environment", *PLoS One*, Vol. 15, No. 4, pp. 1–22, 2020.

[25] M. K. Patra, S. Sahoo, B. Sahoo, and A. K. Turuk, "Game theoretic approach for real-time task scheduling in cloud computing environment", In: *Proc. of 2019 Int. Conf. Inf. Technol. ICIT 2019*, No. December, pp. 454–459, 2019.

[26] M. Y. Mulge, "Optimization of task scheduling algorithm using modified mean Grey-Wolf", *International Journal of Intelligent Engineering and Systems*, Vol. 12, No. 4, pp. 192–200, 2019, doi: 10.22266/ijies2019.0831.18.

[27] D. Alsadie, "TSMGWo: Optimizing task schedule using multi-objectives grey Wolf optimizer for cloud data centers", *IEEE Access*, Vol. 9, pp. 37707–37725, 2021.

[28] Dhari and K. I. Arif, "An Efficient Load Balancing Scheme for Cloud Computing", *Indian Journal of Science and Technology*, Vol. 10, No. 11, pp. 1–8, 2017.

[29] M. A. Alworafi and S. Mallappa, "An enhanced task scheduling in cloud computing based on deadline-Aware model", *Int. J. Grid High Perform. Comput*, Vol. 10, No. 1, pp. 31–53, 2018.

[30] M. C. S. Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, "CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness", In: *Proc. of IM 2017 - 2017 IFIP/IEEE Int. Symp. Integr. Netw. Serv. Manag.*, No. i, pp. 400–406, 2017.

[31] Hussain and M. Aleem, "GoCJ: Google cloud jobs dataset for distributed and cloud computing infrastructures", *Data*, Vol. 3, No. 4, pp. 1–12, 2018.

[32] S. Mirjalili, S. M. Mirjalili, and A. Hatamlou, "Multi-Verse Optimizer: a nature-inspired algorithm for global optimization", *Neural Computing and Applications*, Vol. 27, No. 2, pp. 495–513, 2016.