Iraqi Journal of Science, 2025, Vol. 66, No. 12, pp: 5688- 5702
DOI: 10.24996/ijs.2025.66.12.34

ISSN: 0067-2904

Task Scheduling Model in Cloud Computing with the Artificial Gorilla Troops Optimization Algorithm

Juliet Kadum*, Ismael Salih Aref , Muntadher khamees
Department of Computer Science, College of Science, University of Diyala, Diyala, Iraq

Received: 22/5/2024 Accepted: 21/11/2024 Published: 30/12/2025

Abstract

Task scheduling is one of the main problems in cloud computing. Proper utilization of resources is promoted through cloud computing, and a well-planned task schedule can result in efficient use of resources. To improve the overall performance of the cloud system, task scheduling and resource allocation have become basic needs to efficiently and effectively balance workloads among cloud resources. This paper proposed the Gorilla Troops Optimization-Task Scheduling (GTO-TS) algorithm depending on the artificial Gorilla Troops optimizer (GTO). The experimental evaluation used Cloudsim Simulator and comparison of its results with four methods: Minimum Execution Time (MET), Minimum Completion Time (MCT), Minimum- Minimum (Min-Min), and Maximum -Minimum(Max-Min). The proposed algorithm (GTO-TS) distributes the workload evenly among the virtual machines (VMs) to reduce Makespan and optimize resource usage. The suggested algorithm (GTO-TS) outperformed the four traditional algorithms in terms of effectiveness, as demonstrated by the simulation results.

Keywords: Task scheduling, Makespan, Resource Utilization, Load balance, GTO algorithm, cloud computing.

نموذج جدولة المهام في الحوسبة السحابية باستخدام خوارزمية تحسين قوات الغوريلا الاصطناعية

جوليت كاظم*, اسماعيل صالح عارف, منتظر خميس
قسم علوم الحاسوب، كلية العلوم، جامعة ديالى، العراق

الخلاصة

يعد جدولة المهام أحد المشاكل الرئيسية في الحوسبة السحابية. يتم تعزيز الاستخدام السليم للموارد من خلال الحوسبة السحابية، ويمكن أن يؤدي جدول المهام المخطط جيدًا إلى استخدام فعال للموارد. لتحسين الأداء العام لنظام السحابة، أصبحت جدولة المهام وتخصيص الموارد من الاحتياجات الأساسية لتحقيق التوازن الفعال والكفء بين أحمال العمل بين موارد السحابة. اقترحت هذه الورقة خوارزمية Gorilla Troops optimization–Task Scheduling (GTO–TS) اعتمادًا على مُحسّن Gorilla Troops الاصطناعي (GTO). استخدم التقييم التجريبي Cloudsim Simulator ومقارنة نتائجه بأربع طرق: الحد الأدنى لوقت التنفيذ (MET)، والحد الأدنى لوقت الإكمال (MCT)، والحد الأدنى– الحد الأدنى (Min–Min)، والحد الأقصى– الحد الأدنى (Max–Min). توزع الخوارزمية المقترحة (GTO–TS) عبء العمل بالتساوي بين

*Email: julietkadum@uodiyala.edu.iq

الآلات الافتراضية (VMs) لتقليل Makespan وتحسين استخدام الموارد. تفوقت الخوارزمية المقترحة (GTO–TS) على الخوارزميات التقليدية الأربعة من حيث الفعالية، كما يتضح من نتائج المحاكاة.

1. Introduction

Cloud computing is a key part of distributed computing and provides a range of services directly to consumers as needed. These services encompass servers, storage, software, and databases delivered over the internet[1]. The resource management unit receives user requests that are provided to access the services. After that, the task manager informs the scheduler. The scheduler evaluates the requests using the source information server and distributes them to available virtual machines based on task scheduler algorithms[2]. In cloud computing, task scheduling involves linking user tasks with available cloud resources, such as virtual machines[3]. It employs techniques to determine the order, speed, load balancing, and system throughput of tasks to reduce total completion time (Makespan) and ensure fault-tolerance[1], [4]. Users submit their tasks to the cloud for completion, and the cloud assigns them to a processor for execution. The Data Center Broker (DB) is tasked with identifying and gathering information on both currently available resources (VMs) and potential future resources. Tasks are received and scheduled within the VM in accordance with the scheduling algorithm specified in the DB using the task waiting queue. The scheduled task in the cloud system is illustrated in Figure 1 [5].

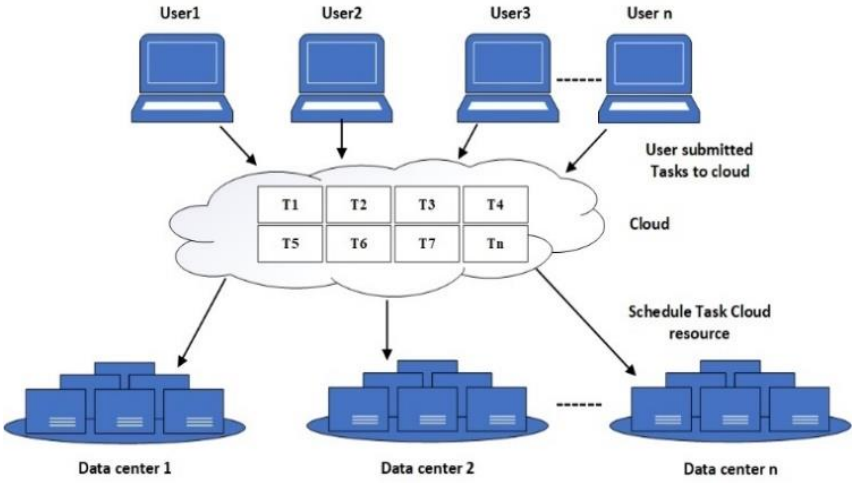

Figure 1: Schedule Task in cloud environment [5].

Cloud service providers and users encounter various obstacles in achieving optimal resource allocation, such as inefficient utilization of resources like CPU, memory, and bandwidth, resulting in uneven distribution and negatively impacting performance. Imbalanced resource allocation leads to decreased efficiency and degraded performance[6].

Scheduling algorithms distribute the workload to optimize resource allocation and minimize implementation time when tasks are sent to the cloud. Despite this, conventional scheduling methods like MET, MCT, Min-Min, and Max-Min frequently exhibit suboptimal performance in load balancing, resulting in underutilized machines and inefficient allocation of resources. Addressing this challenge is crucial for improving task scheduling in the cloud, highlighting the importance of artificial intelligence algorithms[7], [8]. Also, these algorithms may face scalability challenges when operating within extensive and intricate cloud computing systems [9].

Numerous studies have recently focused on the scheduling problem using optimization algorithms [10]. While optimization algorithms are effective for specific and small-scale problems, they may produce subpar solutions for complex cloud scheduling scenarios, like heuristic algorithms[11]. Real-time scheduling of tasks involves dealing with NP-hard problems; it is logical to opt for meta-heuristic optimization, which is generally considered the most suitable approach for complex cloud scheduling problems due to its ability to deliver superior and more flexible solutions[12].

Understanding the significance of load balance in task scheduling, we seek to tackle this issue by employing the GTO meta-heuristic algorithm. This algorithm is designed to accommodate NP-hard problem constraints, explore different solutions, and steer clear of local optima. Also, it facilitates a more even mix of exploration (searching for new areas) and exploitation (focusing on promising areas), thereby increasing the likelihood of finding near-optimal solutions.
The main justifications for implementing this work are as follows:
1- Optimize task distribution on virtual machines and compare the performance of the suggested algorithm (GTO-TS) with the MET, MCT, Min-Min, and Max-Min task scheduling algorithms using Cloudsim Simulator.
2- The proposed algorithm focuses on identifying optimal solutions for task scheduling by balancing exploration and exploitation, factors that make GTO-TS distinct from all other known algorithms in framing optimal task scheduling solutions.
3- Minimizing makespan, maximizing resource utilization, and overall system efficiency in the context of cloud computing.

2. Related work

The following is an analysis of a recent survey conducted by multiple researchers who have implemented task scheduling in cloud computing systems.

Zhifeng Zhong et al., 2016 [13] presented the application of the Greedy Particle Swarm Optimizer (G&PSO) in cloud environments. This approach utilizes the system load distribution and the V.M. resource usage to improve the program's productivity. The system does not respond to unexpected changes in the resource demand correctly; thus, insufficient workload assignments occur. The main goals of the research are to improve cloud performance and to reduce the time of operations.

S. H. H. Madni et al., 2017 [14] presented a study including various experimental approaches to task scheduling in the IaaS environment. The aim of this research was to determine whether the GA or PSO is more effective in organizing and optimizing task scheduling for the best utilization of the resources. The outcomes demonstrated that, for example, PSO and GA were more efficient in certain situations by delivering faster responses and saving resources significantly. However, there were considerable disparities in performance in line with the task type and system necessities.

Ruba Abu Khurma et al., 2018 [15] designed a work scheduling method that applies a slightly modified form of the Round Robin algorithm. The technique is to share the work among available resources so that scheduling will be more feasible and effective. It is noteworthy that said algorithm was more efficient in execution and waiting tasks according to the experiments that were conducted to compare them. Resource usage will become unduly biased towards some if the algorithm does not prioritize the activities well while trying to keep the load distribution balance. The inability of the algorithm to become, for instance, a

flexible way of workload management if the load variation is not met with appropriate algorithms could lead to some resources being overloaded while others remain unused.

Mohamed Abd Elaziz et al., 2019 [16] presented a metaheuristic strategy that uses Differential Evolution (DE) to improve the Moth Search approach (MSA). The methodology known as MSA draws inspiration from nature and mimics the behavior of moths. It employs phototaxis for exploration and Levy flights for exploitation. Due to the limitations of exploitation capability, DE employed a local search (LS) approach. The results of the experiments showed that the LS approach outperformed other approaches significantly when compared to the most commonly accepted performance criteria.

BJ. Hubert Shanthan et al., 2020 [17] proposed a scheduling strategy based on a priority-based approach; thus, it improved the allocation of tasks in multi-cloud systems and introduced PIMTSA. While this would improve the efficiency of the performance due to better utilization of resources and faster response time, it may not have been fully tested, as vast comparisons were not made with other algorithms.

Ahmed Y. Hamed and Monagi H. Alkinani, 2021 [18] presented a GA-based task scheduling algorithm. The technique aims to minimize the makespan and the execution cost with resource utilization as the means. The algorithm's performance was tremendously demonstrated by slack time, execution cost, and resource utilization minimization for the procedure and processor pair. The offered alternative has better scheduling results in terms of time and cost than the existing ones.

Mohammed Gollapalli et al.,2022 [19] used CloudSim, a cloud simulation tool, to examine several scheduling methods in a cloud computing context. The research considers factors like average waiting time, virtual resource costs, and execution time while evaluating the performance of Round Robin, FCFS, and Max-Min algorithms. In both the space-shared and time-shared scenarios, the Max-Min algorithm outperforms the alternatives with respect to average execution time and waiting time.

Ismael Salih Aref et al.,2022 [7] aimed to advance the performance of Max-Min and Min-Min task scheduling algorithms by applying the GA algorithm in cloud computing environments. The study incorporates genetic optimization algorithms to enhance distribution efficiency and reduce processing time. It shows a great performance improvement compared to the traditional method, which contributes a lot to increasing the cloud resource utilization efficiency.

Seema A. Alsaidy et al.,2022 [20] aimed to enhance PSO algorithm efficiency in cloud-based task scheduling through a heuristic initialization technique. This approach helps reduce processing time and improve the efficiency of resource allocation, especially in environments with dynamic loads. The results showed an improvement in the performance of the algorithm thanks to the initialization, which contributed to a more balanced distribution of cloud tasks.

S. Phani Praveen et al.,2023 [21] proposed a technique that uses a mix of GA and GELS algorithms. It possesses a high-performance resource allocation system as well as the capability of shrinking processing time using strategies based on efficient exploration and exploitation. According to the test results, the proposed GAGELS algorithm worked perfectly with GA and PSO by 10% and 30%, respectively. This indicates that it can improve cloud computing performance and make the work more efficient. On the other hand, the longer running time issue demonstrating its effectiveness is in effect.

In a cloud environment, it is important to coordinate task scheduling and resource allocation to create an optimal schedule. By using the GTO algorithm, we can improve scheduling and achieve a better balance between exploring options and avoiding local optimal solutions. In cloud computing, key task scheduling issues include minimizing makespan, load balancing, resource utilization, and scalability. Metaheuristics for task scheduling aim to optimize performance and help achieve the best schedule.

3. Gorilla Troops Optimization Algorithm (GTO)

The metaheuristic design of the Gorilla Troops Optimization algorithm was inspired by the social behaviors demonstrated by groups of gorillas (swarm-based optimizer). The optimization space of the GTO algorithm includes three kinds of solutions, where GX is a proposed gorilla position vector that might be utilized if the current solution doesn't work out. X denotes the position vector of the gorillas. In the end, each iteration's best solution was the silverback. Silverback is the group leader and performs several main tasks, including decision-making, group management, migration, searching for supplies, and protecting the group from enemies. Therefore, it is considered the ideal solution when applying the algorithm mathematically. The foraging strategies and the group lifestyle of gorilla communities have given GTOs unique properties in several optimization problems. The GTO algorithm applies a variety of optimization mechanisms, which are detailed in the below subsections [22].

3.1 Exploration phase

In the exploration phase, three factors come into play. The initial factor involves moving to an unfamiliar location to expand the search area. The second factor relates to the movement of the gorilla. Finally, the third factor is moving towards a familiar place, thus efficiently exploiting the search space. Equation (1) demonstrates the factors involved.

$$GX(t-1) = \begin{cases} (U_d - L_d).r_1 + L_d & if\ rand < p \\ (r_2 - D).X_r(t) + M.H & if\ rand \geq 0.5 \\ X(i) - M(M(X(t) - GX_r(t)) + r_3(X(t) - GX_r(t))) & if\ rand < 0.5 \end{cases} \quad (1)$$

In Eq.(1), the position vector of the gorilla filter solution is denoted as GX (t + 1) in the iterations. The vector X(t) represents the gorilla's present location. Utilized are random numbers with values ranging from 0 to 1, including r_1, r_2, r_3, and rand. The parameter, p, calculates the probability of choosing a relocation mechanism to move the gorilla to an unspecified location. Ud and Ld, denoting the upper and lower limits of the variables, are represented by Xr and GXr, respectively. In conclusion, the values of D, M, and H are computed using Eq.(2), Eq.(4), and Eq.(5), respectively.

$$D = F.\left(1 - \frac{it}{maxit}\right), \quad (2)$$

$$F = cos(2.r_4) + 1, \quad (3)$$

$$M = D.k, \quad (4)$$

$$H = Z.X(t), \quad (5)$$

$$Z = [-D, D]. \quad (6)$$

The iteration value is in Equation (2). In optimization, Max_it is the total number of iterations that must be performed, while Eq.(3) provides an estimation of F, specifies the cosine function as the variable (cos), and a random variable (r_4) which changes from 0 to 1 with each iteration. The silverback leadership is simulated using Eq.(4), where k is a number

chosen at random from -1 to 1. Equation (5) and Eq.(6) are utilized to determine H and Z, respectively, where Z is a variable with random values ranging between –D and D.

3.2 Exploitation phase

The exploring phase makes use of two mechanisms.: first is obeying the silverback's commands, Eq.(7) is used to model this mechanism, and the second is battling for adult females , and Eq.(10) is used to model this mechanism. The parameters D from Eq.(2) and W determine the best mechanism. If D >= W, the following mechanism is chosen; otherwise, competition is selected. The value W is predetermined before initiating the optimization procedure.

$$Gx(t+1) = M * S * (X(t) - X_{silverback}) + X(t) \quad (7)$$

$$S = \left(\left| \frac{1}{N} \sum_{i=1}^{N} GX_i(t) \right|^{g} \right)^{\frac{1}{g}} \quad (8)$$

$$g = 2^{M} \quad (9)$$

X(t) and Xsilverback are two vectors in Eq.(7); one for the gorilla position and the other for the best solution (silverback gorilla position). Moreover, S and M are calculated using Eq.(4) and Eq.(8). Equation (8) denotes the vector positions of the candidate gorillas in iteration t, denoted as GXi(t), in which N is the overall count of gorillas. The g and M are also calculated using Eq.(9), and Eq.(4), respectively.

$$GX(i) = X_{silverback} - (X_{silverback} * \rho - X(t) * \rho) * A , \quad (10)$$

$$\rho = 2 * R_5 - 1, \quad (11)$$

$$A = \alpha * E , \quad (12)$$

$$E = \begin{cases} N_1 , & rand \geq 0.5 \\ N_2 , & rand < 0.5 \end{cases} \quad (13)$$

X(t) and Xsilverback are two vectors in Eq.(10); one for the gorilla position and the other for the best solution (silverback gorilla position). Equation (11) is used to determine Q, which is observed to simulate the impact force. In Eq.(11), R_5 is a variable with random values between 0 and 1. A is a coefficient vector used in Eq.(12) to quantify the level of violence in disputes. E is evaluated, and the impact of violence on the range of solutions is simulated using Eq.(13), while N_1 and N_2 are two random variables with values in a normal distribution.

3.3 fitness function

An important factor in the success of the GTO algorithm is the choice of the fitness function, which is the Makespan in the modified GTO scheduling algorithm. Makespan is the longest completion time of all tasks in the system, which is essentially the identification of when the last task is completed. It is, therefore, an excellent standard for problems with a time objective such as scheduling, the main objective of minimizing the time to complete all tasks also often comes. The formula for makespan (MK) is defined as:

$$MK=\max(C_1, C_2 , \ldots, C_N) \quad (14)$$

where C_i is the completion time of task T_i and N is the total number of tasks. This formula ensures that the makespan reflects the completion time of the longest task, which is key to optimizing resource allocation and ensuring efficient scheduling. By using the makespan as

the fitness function, the improved GTO algorithm can effectively focus on minimizing the overall completion time, improving the performance of the scheduling process [23].

4. Task scheduling algorithms

The metaheuristic scheduling approaches address issues by offering approximate optimum solutions in an acceptable time. Because metaheuristics are so good at addressing complicated and large-scale matters, they have become trendy in recent years[24]. At the same time, the heuristics scheduling approach concerns the design and implementation of scheduling algorithms that fit only certain aspects of the problem (e.g., Makespan). Algorithms for mapping and assigning tasks are two of the most used heuristics for scheduling. These algorithms are discussed extensively because of their importance compared to the proposed algorithm (GTO-TS).

The MET task scheduling heuristic, known as minimum Execution Time, allocates all tasks, regardless of sequence, to a machine with the quickest anticipated execution time for that task. The underlying principle of MET is to allocate each assignment to the most suitable machine. This may result in significant burden imbalances among machines [25].

The MCT (Minimum Completion Time) heuristic task scheduling utilized completion times for task assignment, which considers tasks that have already been assigned contrary to the MET heuristic. This avoids assigning most tasks to the dominant device [25],[26].

The Max-Min task scheduling algorithm chooses tasks with the maximum implementation time and appoints them to resources with the minimum implementation time; the Max-Min algorithm prioritizes larger tasks. The machine's total reply (response) time increases if important tasks are implemented early. A significant drawback of this algorithm is that the most salient tasks are implemented first, which can increase the system's response time [27].

The Min-Min algorithm for task scheduling is a heuristic method that Maps all tasks to the relevant devices (machines) based on their minimum completion times. It first finds the Minimum Completion Times (MCT) for every task in an array of unmapped tasks. It then chooses a task and allocates it to the relevant devices (machines). The Min-Min method grants the highest priority to small tasks, thus increasing the waiting time for large tasks [7].

Proposed GTO-based task scheduling model

The Gorilla algorithm is the basis of the proposed system, which has been used to schedule tasks in a way that reduces Makespan and achieves a balance between the machines that execute the tasks by allocating tasks to the machines in a planned and organized manner. Figure 2, the provided flowchart, illustrates the steps that the proposed system will go through and is set in two main phases: exploration and exploitation, the combination of which can lead to the best possible solution. The algorithm's variables are initialized, and the fitness value of each thought option is evaluated. In the exploration phase, different possible solutions are being devised. In this step, multiple solutions are proposed, and all are passed on to the next step (exploitation), where the best solution is selected. On the other hand, the exploitation step is aimed at further working with the synthetically provided solutions so that we can choose the best of them. In this stage, the most efficient scheduling system is ensured by the identification of the most efficient solution to fix the optimal scheduling.

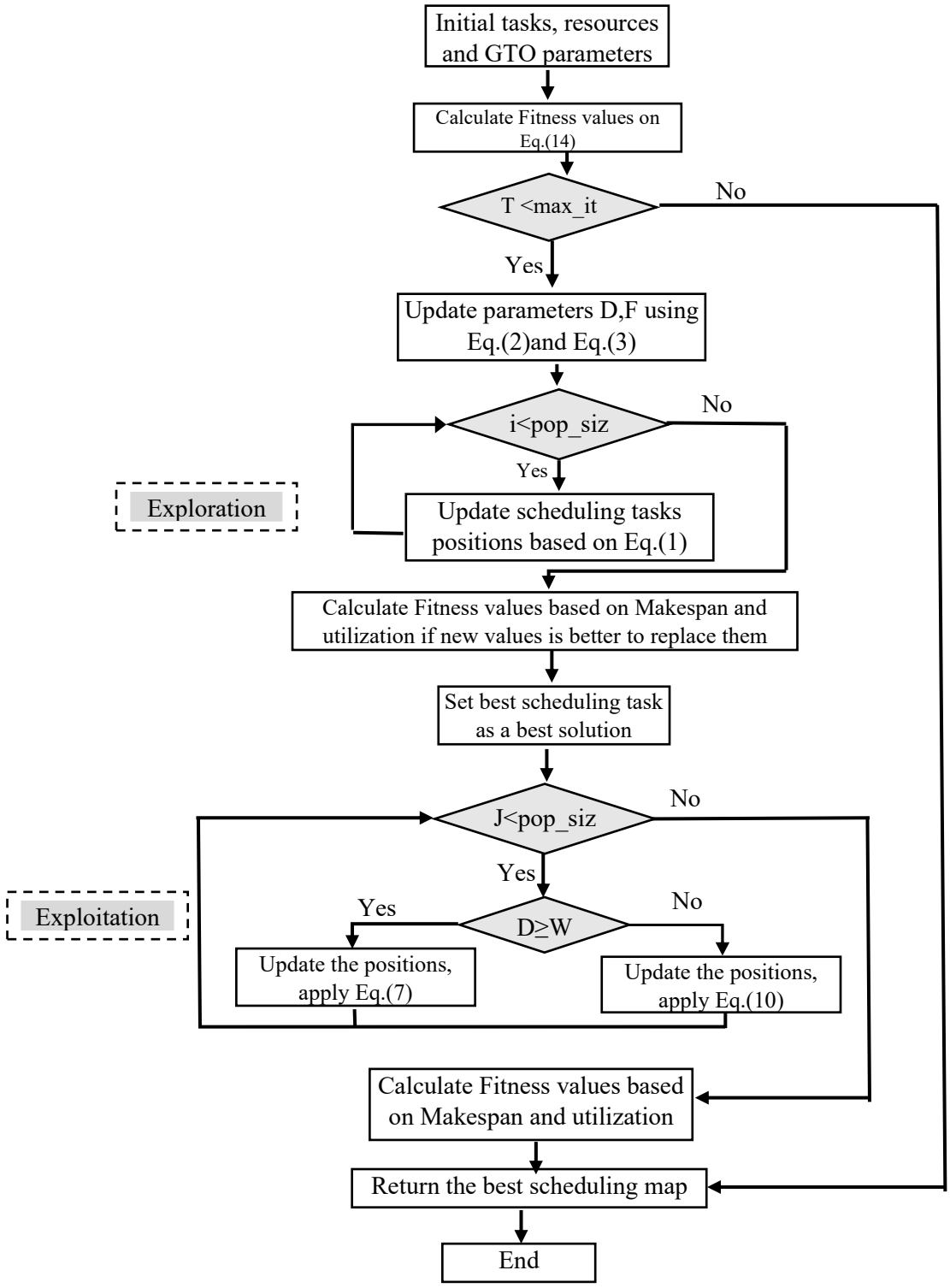

Figure 2: Flowchart for Proposed GTO-TS Algorithm

A. Exploration Phase

The mechanisms are used in this section to identify potential solutions. In the GTO algorithm, every machine is considered a potential solution, and the strongest candidate at each optimization operation phase is chosen to carry out a specific task. Three mechanisms are used to identify three potential machines for work, and the best machine that meets the

target objectives (reduce Makespan or make balance) will be chosen later to complete a task from the tasks waiting in the queue.

First mechanisms.

In this case, the value of p determines which mechanism is selected first, as this value is considered a rule for choosing the mechanism. When the value of p>rand, first mechanical is chosen to enable the algorithm to monitor the full issue space effectively.

Second mechanisms.

When the value of rand< 0.5, the second mechanism is chosen to find the best machine M that greatly improves the GTO's capacity to look for various optimization areas.

Third mechanism.

When the value of rand>= 0.5, the third mechanism is chosen to find a machine M that increases the balance between exploration and exploitation.

B. Exploitation phase

Two behaviors, working as instructed by the silverback and grappling and competing to win adult females, were utilized during the GTO algorithm's exploitation phase. The gorilla group's leader, Silverback, was chosen as the finest site to install a machine. Two mechanisms were implemented, as delineated in the exploitation phase. The first is to follow the silverback, and the second is competition for adult females by using the value of the C parameter in Eq.(2). According to the value of the parameter C, if C>=W, then the decision tends to follow the instructions of the silverback, which means choosing a machine and assigning tasks that improve load balance across machines. Else, if C<W, the balance tilts to the other side, which is choosing the second mechanic, adult females' Competition, which means selecting the machine and assigning the task that reduces Makespan over all machines.

first mechanism

This mechanism chooses the machine that helps maintain balance among the other machines. It is important to pay attention while selecting tasks for scheduling processes to ensure balance, and this means allocating tasks almost equally among machines to reduce the load on specific machines. Therefore, in this technique, importance is placed on creating a balance between the machines. This approach is adopted when the C>=W value is specified.

second mechanism

In this mechanism, the machine that helps to reduce the cost is selected. One machine can execute more than one task after another, so the execution time is computed for each task carried out by the same machine and added to the overall execution time. Therefore, it is necessary at this stage to determine the path that contributes to reducing the total time of implementation on the machines by choosing the tasks that contribute to reducing the cost and distributing them to the tasks; this is done in this mechanism that identifies and chooses the tasks that reduce the final cost of the system. The second mechanism is chosen for the exploitation stage when C < W.

Algorithm 1 : Proposed GTO-TS Algorithm

Input:

Set Parameters p, β, N: pop_size,
T: iterations maximum number. M: number of resources.
Tasks (X_i) and Recourses (R_j) where i ∈{1.. pop_size }, j ∈ {1..M}

Output:

Optimal Task Scheduling

The initialization step:
Initialize the random task (population) X_i **(i= 1, 2, , pop_size)**
Initialize the random Resources R_j **(j= 1, 2, ,N)**
Calculate each gorilla's fitness values Eq. (14) .
****Main Loop**
while (No satisfied condition stop) **do**
use Eq. (2) to modify *D*
use Eq. (4) to modify *M*
***** The Exploration stage**
for (every Gorilla (***Xi***) in population) do
Modify Gorilla (***Xi***) position by using Eq. (1)
End for
**** Create group**
Compute the value of Gorilla's fitness using Eq. (14) ;
if GX is more valuable than X **then** exchange them
Set the position of silverback by *Xsilverback*. (best position)
***** The Exploitation stage**
for (every Gorilla (***Xi***) in population) **do**
if (|C| is equal and greater than **1) then**
Modify Gorilla (***Xi***) position by Eq. (7)
Else
Modify Gorilla (***Xi***) position by Eq. (10)
End if
End for
**** Create group**
Compute the value of Gorilla's fitness
Replace the old solutions if the new ones are more valuable.
Assign the position of silverback by *Xsilverback*. (best position)
According to best position, assign T to R .
End while
Return scheduling.

Results And Discussion

The findings outlined in this section are derived from computational experiments conducted on the proposed methodology, which shows its effectiveness. The recommended algorithm was designed using the Cloudsim platform developed in the Java programming language for simulating cloud computing environments. Experiments were performed and validated on a computer with Intel(R) Core (TM) i7-3632QM, 8 CPU @ 2.5 GHz, RAM of 8GB, and 64-bit Windows OS as its configuration. The following section outlines the findings from simulations that prioritized minimal processing time and improved load distribution. The

results are compared to the proposed algorithm with (MET, MCT, Min-Min, and Max-Min) algorithms.

The experiment involves five virtual machines (VM), each with two processing elements (pe). Two data centers and 100-200 tasks were used in the simulation platform to experiment on Cloudsim. The tasks have durations ranging from 10,000 MI (million instructions) to 50,000 MI. The parameter settings for the Cloud simulator can be found in Table 1. The parameters of the GTO-TS setting using the MATLAB R2021b laptop running Windows 11 Enterprise 64-bit with an Intel Core i7-8510U 2.6 GHz processor and 8.00 GB RAM have been used to test and run the GTO-TS. Three parameters in the GTO-TS algorithm shown in Table 2 are used as presented in Reference [28].

Table 1: The Cloudsim parameters

Entity Type	Parameters	Values
Virtual machine (VM)	MIPS of pe	512-1024
	Numbers of vm	5
	Numbers of pe per vm	1-3
	The bandwidth	500-1200
	The memory	512-2048
	The storage	100000-800000
	Cost per unit of vm	1-10
Tasks (cloudlet)	Task length	10000-50000
	Total number of tasks	100-500
	Size of file	500-2000

Table 2: GTO-TS Parameter settings value

Algorithm	Parameters	Value
GTO	β	3
	W	0.3
	P	0.08

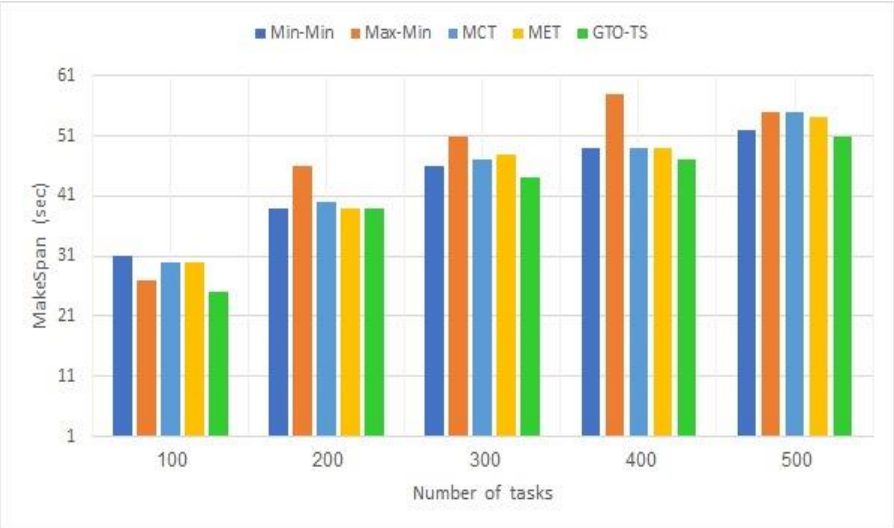

Figure 3: Comparison of the performance of the proposed GOT-TS algorithm with other state-of-the-art

algorithms with respect to Makespan

The proposed algorithm operates within a static environment, meaning that no additional resources are introduced during runtime. Performance evaluations were conducted using five virtual machines (VMs) to execute 100-500 tasks. In Figure 3, a comparison of algorithm performance is presented through a bar chart that depicts the correlation between the number of completed tasks (x-axis) and the maximum completion time (makespan, y-axis). This visual representation also highlights variations in the outcomes of different algorithms and their effectiveness in achieving optimal results. The algorithms undergo multiple tests under diverse conditions, with random values used for each test to showcase their efficiency and effectiveness. The first experiment involved testing 100 tasks to assess how well the algorithms performed with a limited task load. The findings showed that the recommended algorithm outperformed the others, achieving an average completion time of 25 and securing the top spot. The Max-Min algorithm came in second with an average time of 27. Following that, we ranked the remaining algorithms: Min-Min, MCT, and MET.

In the second experiment, these algorithms were tested with 200 tasks. It is widely known that increasing the number of tasks leads to changes in execution time and consequently affects the results. The efficiency of the proposed algorithm becomes more apparent as the number and variety of tasks increase. The experimental findings revealed that the proposed algorithm delivered comparable results to Min-Min and Max-Min algorithms, with an average completion time of 39, due to all algorithms' ability to reach optimal solutions. The remaining algorithms were then ranked (MCT, Max-Min).

The third experiment took place to further assess the algorithms' efficiency by scheduling 300 tasks over the available resources. The results showed that the GTO-TS substantially surpassed the other algorithms, attaining the maximum task completion efficiency with a value of 44, which is the shortest recorded completion time in this experiment. The remaining algorithms yielded varying results, with Min-Min, MCT, MET, and ultimately Max-Min ranking in terms of performance. Of all the algorithms, Max-Min performed the poorest, suggesting that tasks scheduled with this algorithm had shorter execution times due to its frequent association with tasks requiring minimal processing time. The third test result further emphasizes the efficiency of the algorithm in scheduling tasks with short execution times.

The previous three experiments confirmed that the proposed algorithm is effective. To further validate the results and ensure their accuracy, two more experiments were carried out: one with 400 tasks and another with 500 tasks. The outcomes of these experiments continued to show that the proposed algorithm outperforms other algorithms, even when dealing with a larger number of tasks and varying execution times. Overall, these experiments demonstrate that the proposed algorithm efficiently schedules diverse tasks, whether they are numerous or of varying durations. It consistently achieves efficient scheduling in minimal time, confirming its effectiveness in handling diverse conditions. Table 3 illustrates the superiority of the proposed algorithm over its alternatives.

The evaluation of resource utilization, also known as load balance, is the second aspect considered in performance assessment [29]. Figure 4 illustrates an evaluation of the effectiveness of the algorithms employed. The horizontal axis represents the number of tasks to be completed, ranging from 100 to 500. This augmentation of tasks includes a significant task number that could impact algorithm performance. On the vertical axis, the scheduling rate is measured to gauge the efficiency of resource allocation in the system. A higher

percentage indicates a more efficient solution, indicating that fewer tasks are utilizing the same resources.

Table 3: Comparison of Makespan performance

No of Tasks	Algorithm				
	Min-Min	Max-Min	MCT	MET	GTO-TS
100	31	27	30	30	25
200	39	46	41	39	39
300	46	51	47	48	44
400	49	58	48	49	47
500	52	55	55	54	51

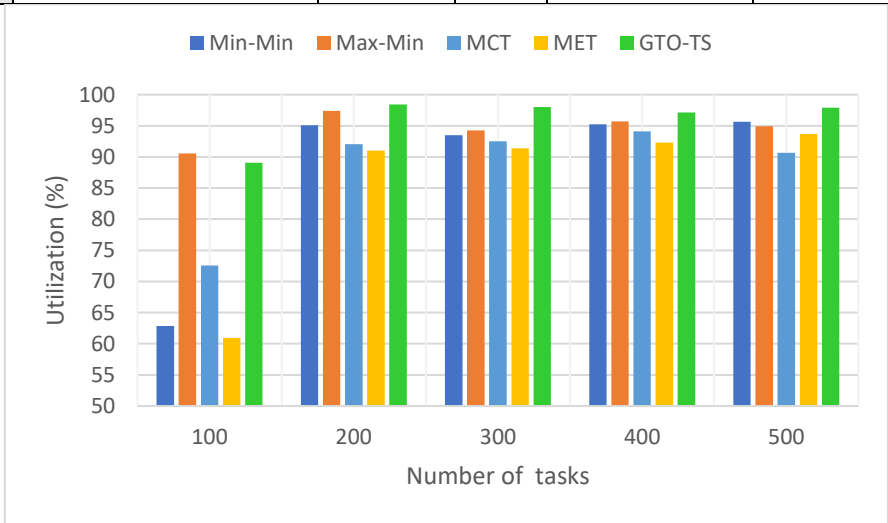

Figure 4: Resource Utilization vs. number of tasks

The initial test assessed the effectiveness of different algorithms in solving a problem with 100 tasks, and it revealed notable differences among the potential solutions. The Max-min algorithm stood out as the top performer, achieving nearly 90 percent success and claiming victory. The GTO-TS algorithm came in second place with a success rate close to that of Max-Min, while the others ranked in descending order: MCT, Min-Min, MET. GTO-TS's second-place position was influenced by the prevalence of tasks with longer execution times, giving Max-Min a slight edge. Despite this slight advantage, GTO-TS demonstrated reduced execution time, underscoring its efficiency in task completion as evidenced in Table 1 during testing with 100 tasks. In the second and third tests, the number of tasks rose to 200 and 300, respectively. Most algorithms showed increased utilization levels. However, the GTO-TS algorithm performed even better, proving its ability to efficiently handle large numbers of tasks and consistently achieve good results.

In the fourth and fifth experiments, the algorithms are tested on many task sets, with the fourth experiment including 400 tasks and the fifth 500 tasks. Results showed that the proposed algorithm outperformed the other algorithms in both experiments, reflecting its ability to maintain high efficiency as the number of tasks increased, as well as its effective scalability with the growth of a number of tasks. To get a clearer idea of the test results across the five experiments, Table 1 shows each experiment's expenditure point by the rate of its utilization.

Table 4: Comparison of the Usage of Resources

No of Tasks	Algorithm				
	Min-Min	**Max-Min**	**MCT**	**MET**	**GTO-TS**
100	62.829582	90.5393258	72.5819936	60.9581994	89.06122449
200	95.0895141	97.4025974	94.2437338	91.0293166	97.4025974
300	93.5064935	94.2437338	92.0343156	91.3749351	98.4258065
400	95.2254098	95.7059315	92.5064935	92.3236066	97.1607516
500	95.6563707	94.924182	94.0983607	93.7065637	97.9269497

Conclusion

Scheduling plays a great role in achieving efficiency in task scheduling and resource utilization, which remains one of the challenging problems in cloud computing. The system proposed here employed the GTO optimization algorithm for task scheduling. Experimental results and comparisons with various algorithms resulted in GTO-TS performing well in task scheduling and resource utilization in diverse operating environments, hence proving its effectiveness and adaptability. In the future, it would be useful to investigate how GTO can be combined with some other algorithms, like the evolutionary algorithm, to enhance scheduling accuracy and flexibility for various operating conditions.

References

[1] M. B. Gawali and S. K. Shinde, "Task scheduling and resource allocation in cloud computing using a heuristic approach," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 7, pp. 1–16, 2018.

[2] D. Saxena and A. K. Singh, "A Comprehensive Survey on Sustainable Resource Management in Cloud Computing Environments," Authorea Preprints, 2024.

[3] S. Gurusamy and R. Selvaraj, "Resource allocation with efficient task scheduling in cloud computing using hierarchical auto-associative polynomial convolutional neural network," *Expert Systems with Applications*, vol. 249, p. 123554, 2024. doi:10.1016/j.eswa.2024.123554.

[4] D. Sharmah and K. C. Bora, "A Survey on Dynamic Load Balancing Techniques in Cloud Computing," in *International Conference on Emerging Electronics and Automation*, Springer, 2022, pp. 273–282.doi: 10.1007/978-981-99-6855-8_21.

[5] K. Dubey, M. Kumar, and S. C. Sharma, "Modified HEFT algorithm for task scheduling in cloud environment," *Procedia Computer Science*, vol. 125, pp. 725–732, Jan.2018.

[6] K. Saidi and D. Bardou, "Task scheduling and VM placement to resource allocation in Cloud computing: challenges and opportunities," *Cluster Computing*, vol. 26, no. 5, pp. 3069–3087, 2023.

[7] I. S. Aref, J. Kadum, and A. Kadum, "Optimization of max-min and min-min task scheduling algorithms using ga in cloud computing," in 2022 *5th International conference on engineering technology and its applications (IICETA)*, IEEE, 2022, pp. 238–242.

[8] U. Bhoi and P. N. Ramanuj, "Enhanced max-min task scheduling algorithm in cloud computing, " *International Journal of Application or Innovation in Engineering and Management (IJAIEM)*, vol. 2, no. 4, pp. 259–264, 2013.

[9] G. Natesan and A. Chokkalingam, "Task scheduling in heterogeneous cloud environment using mean grey wolf optimization algorithm," *Information & Communications Technology Express*, vol. 5, no. 2, pp. 110–114, June. 2019.

[10] F. S. Prity, M. H. Gazi, and K. M. A. Uddin, "A review of task scheduling in cloud computing based on nature-inspired optimization algorithm," *Cluster Computing*, vol. 26, no. 5, pp. 3037–3067, June. 2023.

[11] 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, Apr .2022.

[12] Y. Zhang, L. Wu, M. Li, T. Zhao, and X. Cai, "Dynamic multi-objective workflow scheduling for combined resources in cloud," **Simulation Modelling Practice and Theory**, vol. 129, p. 102835, 2023.

[13] Z. Zhong, K. Chen, X. Zhai, and S. Zhou, "Virtual machine-based task scheduling algorithm in a cloud computing environment," *Tsinghua Science and Technology*, vol. 21, no. 6, pp. 660–667, December 2016.

[14] S. H. H. Madni, M. S. Abd Latiff, M. Abdullahi, S. M. Abdulhamid, and M. J. Usman, "Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment," *PLoSOne*, vol.12,no.5,p.e0176321, May.2017.

[15] R. A. Khurma, H. Al Harahsheh, and A. Shariah, "task scheduling algorithm in cloud computing based on modified round robin algorithm.," *Journal of Theoretical & Applied Information Technology*, vol. 96, no. 17, Sep. 2018.

[16] M. Abd Elaziz, S. Xiong, K. P. N. Jayasena, and L. Li, "Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution," *Knowledge-Based Systems*, vol. 169, pp. 39–52, 2019.

[17] B. J. H. Shanthan, L. Arockiam, A. C. Donald, A. D. V. Kumar, and R. Stephen, "Priority intensed meta task scheduling algorithm for multi cloud environment (PIMTSA)," in *Journal of Physics: Conference Series*, IOP Publishing, Vol. 1427, No. 1, p. 012007, 2020.

[18] A. Y. Hamed and M. H. Alkinani, "Task scheduling optimization in cloud computing based on genetic algorithms," *Computers, Materials & Continua*, vol. 69, no. 03, pp. 3289–3301, Jan.2021. DOI:10.32604/cmc.2021.018658.

[19] Gollapalli, M., Alamoudi, A., Aldossary, A., Alqarni, A., Alwarthan, S., AlMunsour, Y.Z., Abdulqader, M.M., Mohammad, R.M., Chabani, S., "Modeling Algorithms for Task Scheduling in Cloud Computing Using CloudSim.," *Mathematical Modelling of Engineering Problems*, vol. 9, no. 5, pp. 1201-1209, 2022.

[20] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 2370–2382, Jun.2022.

[21] S. P. Praveen, H. Ghasempoor, N. Shahabi, and F. Izanloo, "A hybrid gravitational emulation local search-based algorithm for task scheduling in cloud computing," *Mathematical Problems in Engineering*, vol. 2023, Feb.2023.

[22] B. Abdollahzadeh, F. Soleimanian Gharehchopogh, and S. Mirjalili, "Artificial gorilla troops optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems," *International Journal of Intelligent Systems*, vol. 36, no. 10, pp. 5887–5958, July.2021.

[23] M. Agarwal and G. M. S. Srivastava, "A cuckoo search algorithm-based task scheduling in cloud computing," in *Advances in Computer and Computational Sciences: Proceedings of ICCCCS 2016*, Volume 2, Springer, 2018, pp. 293–299.

[24] E.-G. Talbi, "Metaheuristics: from design to implementation". John Wiley & Sons, 2009, p.23.

[25] A. S. e Santos and A. M. Madureira, "Ordered minimum completion time heuristic for unrelated parallel-machines problems," in 2014 *9th Iberian Conference on Information Systems and Technologies (CISTI)*, IEEE, 2014, pp. 1–6.

[26] L. D. Briceño, H. J. Siegel, A. A. Maciejewski, and M. Oltikar, "Characterization of the iterative application of makespan heuristics on non-makespan machines in a heterogeneous parallel and distributed environment," *The Journal of Supercomputing*, vol. 62, pp. 461–485, March.2012. doi:10.1007/s11227-011-0729-7.

[27] J. Zhou, T. Wang, P. Cong, P. Lu, T. Wei, and M. Chen, "Cost and makespan-aware workflow scheduling in hybrid clouds," *Journal of Systems Architecture*, vol. 100, p. 101631, Nov.2019.

[28] M. Balachandran, S. Devanathan, R. Muraleekrishnan, and S. S. Bhagawan, "Optimizing properties of nanoclay–nitrile rubber (NBR) composites using face centred central composite design," *Materials and Design*, vol. 35, pp. 854–862, 2012.

[29] S. Dhahbi, M. Berrima, and F. A. M. Al-Yarimi, "Load balancing in cloud computing using worst-fit bin-stretching," *Cluster Computing*, vol. 24, no. 4, pp. 2867–2881, May.2021.