AUT Journal of Modeling and Simulation

AUT J. Model. Simul., 57(1) (2025) 89-112
DOI: 10.22060/miscj.2025.23520.5381

COTSA: A Load-Balanced Task Scheduling Algorithm using Coati Optimization in Cloud Computing Environment

Zahra Jalali Khalil Abadi, Najme Mansouri*, Mohammad Masoud Javidi, Behnam Mohammad Hasani Zade

Department of Computer Science, Shahid Bahonar University of Kerman, Kerman, Iran.

ABSTRACT: During the scheduling process, it is important to respect the constraints given by the jobs and the cloud providers. In addition to maintaining a balance between Quality of Service (QoS), fairness, and efficiency of jobs, scheduling is challenging. This paper aims to propose an efficient algorithm for load-balanced task scheduling in the cloud. Our algorithm uses a new meta-heuristic algorithm called COA (Coati Optimization Algorithm) to solve the task scheduling problem. This method is called COTSA (Coati Optimization-based Task Scheduling Algorithm). Its main goal is to reduce execution costs, load balancing, resource consumption, and makepan. Additionally, experimental results indicate that COTSA contributes to reduced energy consumption and enhanced system scalability and fault tolerance under simulated conditions. These improvements suggest potential suitability for dynamic and large-scale cloud infrastructures, though performance may vary depending on workload characteristics and system configurations. It is compared with Walrus Optimizer (WO), Slap Swarm Algorithm (SSA), Whale Optimization Algorithm (WOA), Zebra Optimization Algorithms (ZOA), Grasshopper Optimization Algorithm (GOA), Sooty Tern Optimization Algorithm (STOA), Golden Eagle Optimizer (GEO), Grey Wolf Optimizer (GWO), Subtraction-Average-Based Optimizer (SABO), and Sand Cat Swarm Optimization (SCSO), which are popular meta-heuristics. Experimental results demonstrate that COTSA reduces makespan by approximately 9%, lowers execution cost by up to 40%, improves resource utilization by around 3%, and enhances load balance by up to 30%, energy consumption about 36%, scalability near 17%, and fault tolerance about 16%, making it a robust and scalable solution for efficient cloud task scheduling.

Review History:

Received: Sep. 10, 2024
Revised: Sep. 02, 2025
Accepted: Sep. 07, 2025
Available Online: Sep. 08, 2025

Keywords:

Cloud Computing
Task Scheduling
Load Balancing
Meta-heuristic
Coati Optimization Algorithm
COTSA

1- Introduction

IoT allows the exchange of data over a network between devices, things, and any digital assets without human interaction [1]. Cloud processing of data generated by end-user devices in a short period of time is the main characteristic of IoT. In recent years, communication, interaction, and work have all undergone a revolution. Smartphones and cloud computing have led to this revolution. The smartphone has established itself as the preferred device for interacting with the Internet, with a 97% penetration rate [2]. The success of these devices has been largely attributed to the use of cloud environments [3].

Task scheduling in cloud computing is a critical yet complex problem due to its NP-hard nature and the diverse constraints imposed by both users and cloud service providers [4]. The dynamic and distributed nature of cloud environments makes it challenging to efficiently allocate resources while maintaining high performance. Among the most pressing challenges are minimizing makespan, reducing execution costs, optimizing resource utilization, and achieving balanced task distribution across virtual machines. These objectives often conflict with one another, and designing an algorithm that can simultaneously optimize them remains a significant difficulty. Moreover, scalability and fault tolerance have become essential due to the growing complexity and heterogeneity of modern cloud systems. Energy consumption is another concern, particularly in large-scale and resource-constrained environments, where inefficiencies can lead to high operational costs and environmental impact. This paper aims to address the limitations of existing task scheduling algorithms by proposing a novel solution called COTSA (Coati Optimization-based Task Scheduling Algorithm). The goal is to develop a meta-heuristic algorithm that effectively allocates tasks to virtual machines in a way that minimizes execution cost and makespan, maximizes resource utilization, and ensures a high level of load balancing. Beyond these core objectives, COTSA is also designed to enhance energy efficiency, scalability, and fault tolerance—qualities increasingly vital for real-time and large-scale cloud applications. The algorithm is inspired by the natural

*Corresponding author's email: najme.mansouri@gmail.com

behavior of coatis, whose group hunting and predator-avoidance strategies are modeled to strike a balance between exploration and exploitation in the search space.

The primary contribution of this paper is the development of a novel meta-heuristic algorithm named COTSA for effective task scheduling in cloud computing environments. COTSA is inspired by the natural foraging and survival behaviors of coatis, which are modeled to perform a balanced exploration and exploitation of the solution space. By adapting the Coati Optimization Algorithm (COA) to the cloud context, the proposed method provides an innovative mechanism to address the multi-objective nature of task scheduling problems. Another significant contribution lies in the design of a comprehensive objective function that considers not only traditional performance metrics such as makespan, execution cost, and resource utilization, but also incorporates energy consumption, load balancing, scalability, and fault tolerance. This multi-dimensional optimization approach ensures that the algorithm is aligned with both performance and sustainability goals, making it suitable for real-time and large-scale cloud applications. The paper also contributes by demonstrating the scalability and adaptability of COTSA under varying workload and infrastructure configurations. Two experimental scenarios are designed: one with a fixed number of tasks and varying virtual machines, and another with a fixed number of VMs and varying tasks. Across both scenarios, COTSA consistently outperforms existing algorithms in key metrics, indicating its robustness and generalizability in dynamic environments. The empirical findings are statistically validated using ANOVA and confidence interval analysis, underscoring the significance and reliability of the improvements. Finally, the paper offers insight into the computational complexity and runtime efficiency of the proposed method.

Figure 1 shows the organization of this study. In section 2, we discuss cloud computing, task scheduling, and COA. Section 3 discusses existing papers on task scheduling in cloud environments. Section 4 describes COTSA. Section 5 evaluates COTSA's performance. In section 6, the conclusion is discussed.

2- Background

2- 1- Cloud computing

In cloud computing, several types of requests are handled from the cloud, and clients are provided with a quick service. Globally, it is a model for computing and processing. High-speed computations in the cloud can enhance the prediction process rapidly. Several concepts combined with cloud computing make it the most powerful technology and are used in several different business sectors and IT industries. Users get on-demand access to a wide range of computing resources, including CPUs, memory, servers, storage, and applications. Furthermore, these resources are usually assigned to clients at minimal cost. Whenever the number of requests increases at a particular time, then it becomes difficult to manage each request within the shortest possible reaction time. Cloud Service Providers (CSP) are responsible for allocating incoming tasks to appropriate Virtual Machines (VMs) so as not to overload them and keep the load balanced among them [5].

In Fig. 2, four main layers represent the architecture of cloud computing:

- *Hardware Layer:* Data centers, storage, and CPUs are all part of this foundation.
- *Infrastructure Layer:* In this layer, virtualized resources are provided, such as virtual machines from Amazon Web Services (AWS).
- *Platform Layer:* Google App Engine is a platform for developing and deploying applications.
- *Application Layer:* In this layer, Software as a Service (SaaS) applications like Gmail are included.

2- 2- Task scheduling

Scheduling tasks is one of the most prominent problems in many research studies, and its purpose is to map several tasks to the correct processor to optimize one or more objectives at an acceptable time [6]. Since scheduling has a large solution space, it is classified as an NP-hard problem, and finding the optimal solution takes time. It improves the quality of service by prioritizing given tasks during a specific period of time [7]. It also tends to satisfy some constraint conditions in

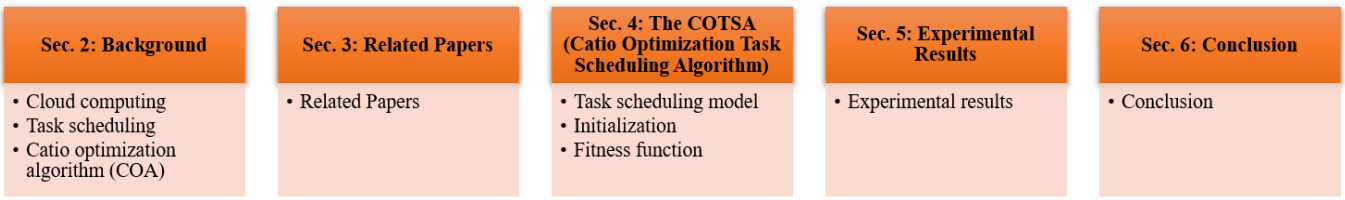

Fig. 1. Article structure.

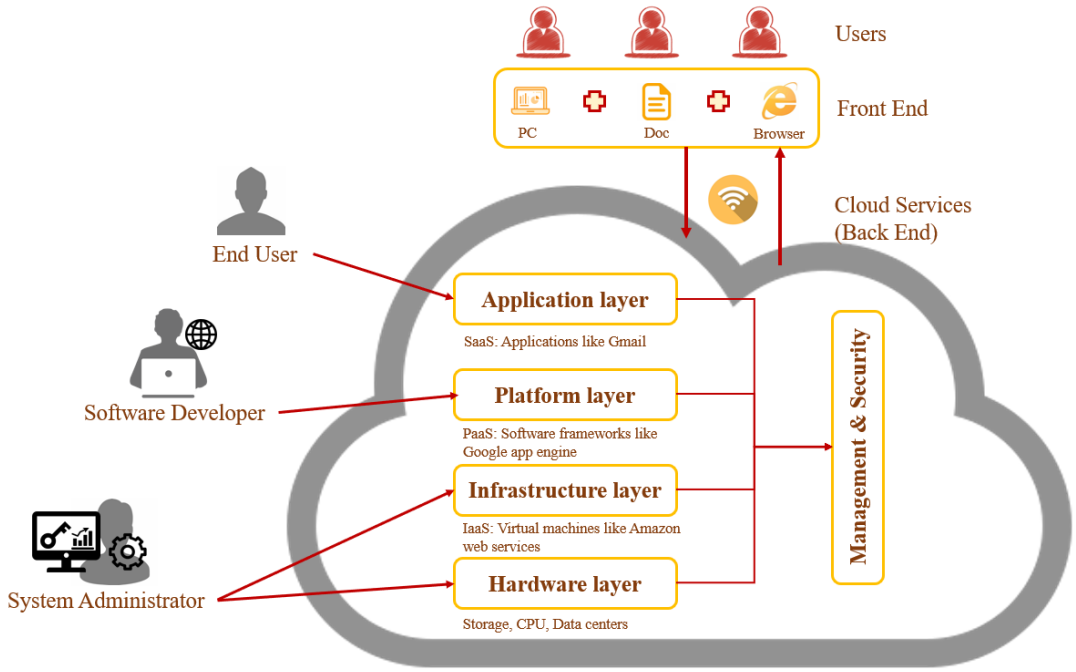

Fig. 2. Cloud computing architecture.

the problem and optimizes one or more objective functions. Creating schedules will allow tasks to be processed and allocated to processors. To meet all requirements of the system, we must schedule tasks so that we can maximize our limited resources. Cloud Computing Systems have recently been researched in detail [9]. They are widely used to process tasks very quickly and to meet the varied computing needs of a wide range of users. Task scheduling systems can divide tasks into smaller subtasks so they will run in parallel. There are almost always constraints and dependencies within these smaller subtasks, such that some subtasks must be run in order before others [10].

Figure 3 illustrates a task scheduling model. Tasks (often called cloudlets) are submitted to the cloud system by users. Workflows can be as simple as computations or as complex as workflows. Virtual machines, storage, and network bandwidth are identified by the cloud system. To determine the best allocation of tasks to resources, various algorithms are used. Task schedulers monitor the system for the specified triggers and execute actions when the conditions are met. With automation, repetitive tasks can be streamlined and performed consistently and efficiently. As soon as tasks are completed, users are notified of the results. In order to improve future scheduling decisions, feedback from the execution process is used. Cloud computing uses resources efficiently, minimizes costs, and provides users with the performance they expect.

2- 3- Coati Optimization Algorithm (COA)

The metaheuristic algorithm begins with a set of randomly feasible solutions. In a repetition-based process, candidate solutions are then updated and improved. The algorithm steps are completed before choosing a suitable candidate solution. The results of metaheuristic algorithms are not guaranteed to be the best global solutions. This optimization approach uses random search. Since these solutions are close to the original solutions, they can be accepted as quasi-optimal solutions [11]. Different metaheuristic algorithms produce different results when solving the same problem. For optimization problems, several algorithms have been developed.

In nature, coati behavior is mimicked by the Coati Optimization Algorithm (COA) [12]. The COA simulates two of the most important natural behaviors of coatis: attacking and hunting iguanas and escaping from predators. Exploration and exploitation are two phases of COA implementation. There are several advantages to the proposed COA approach for global optimization problems. Since COA has no control parameter, no parameters need to be controlled. Furthermore, COA is highly effective in solving a wide variety of optimization problems across various sciences and fields. The proposed method is highly convergent in providing applicable decision variables to optimization tasks, especially complex ones, by balancing research and research in the search process. The proposed COA is also very powerful when dealing with real-life optimization tasks.

They are members of the Procyonidae family, also known as coatis. These mammals are diurnal in the southwest United States, Mexico, Central America, and South America [13]. Moreover, coatis have a long non-prehensile tail used for balance and signaling, and a slim head with a large nose and black paws. From head to tail tip, an adult coati can measure 69 cm in length. In general, it's about the size of a big house cat, weighing between 2 and 8 kilograms and standing about

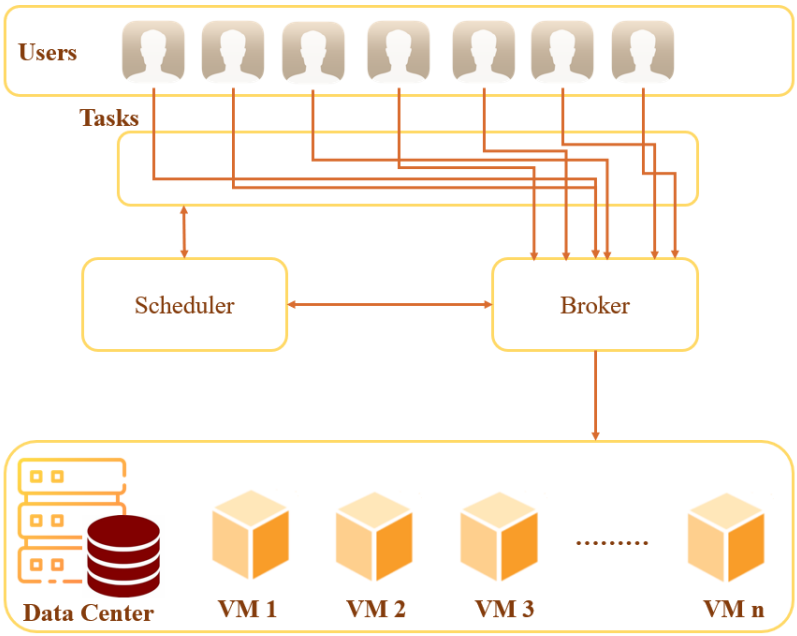

Fig. 3. Task scheduling model.

30 cm tall at its shoulder. As well as having large, sharp canines, males are almost twice as large as females. This is a measurement of the South American coati and white-nosed coati. There is a difference in size between mountain coatis and other coatis. Coatis enjoy eating iguanas. The coatis hunt iguanas in groups because they live in trees. Others attack iguanas quickly by climbing trees and scaring them into falling. However, predators may attack coatis. In addition to ocelots and tayras, dogs, foxes, boa constrictors, maned wolves, anacondas, and jaguarundis, there are many predators for coatis. Eagles such as the harpy, black-and-chestnut, and ornate hawk hunt them. The COA consists of the following steps:

- Initialization: In the search space, randomly generate coatis (solutions).
- Fitness Evaluation: Assess the fitness of each coati based on the objective function.
- Exploration Phase: Simulate how coatis attack and hunt iguanas. The following steps are involved:
  - Random Walk: Coatis search randomly for potential prey (solutions).
  - Group Hunting: Coatis collaborates with others to explore different areas of the search space.
- Exploitation Phase: Create a simulation of the coatis' escape from predators. The process consists of:
  - Local Search: solutions are refined by Coatis to make them better.
  - Leader Selection: Assess the performance of the best coati (leader) and adjust other coatis accordingly.
- Update Positions: Assess coatis' strategic exploration and exploitation.
- Termination: Evaluate fitness, explore, and exploit until a termination criterion is met (e.g., satisfactory fitness level).
- Output: Provide the best solution found by the coatis.

The various stages of the COA implementation are presented as pseudocode in Figure 4 (Algorithm 1). The pseudo-code outlines the COA, a metaheuristic approach inspired by the foraging behavior of coatis. The algorithm begins by initializing the population of coatis and evaluating their positions. Over a set number of iterations, COA operates in two main phases: exploration (Phase 1) and exploitation (Phase 2). In Phase 1, the coatis simulate hunting and attacking an iguana, where half the population updates positions based on targeted calculations (Eq. 4, 7 in main article), while the other half follows randomized iguana movements (Eq. 5–7 in main article). Phase 2 mimics escaping predators, refining solutions by adjusting positions within local bounds (Eq. 8–10 in main article). The best solution is retained in each iteration, and the algorithm ultimately outputs the optimal solution. COA balances exploration and exploitation to efficiently navigate complex optimization problems.

3- Related works

More than ten years have passed since cloud computing was first proposed. Thousands of researchers and companies today are attracted to cloud computing as a result of its scalability, high reliability, low cost, and on-demand capabilities. Cloud computing has a major scheduling and resource allocation problem. Resource allocation and task assignment are highly critical challenges. Resource allocation and task scheduling cannot be improved using any methods or techniques. Previously, virtual machine instances were used for scheduling. A major disadvantage of using virtual machines is that they take a long time to start and consume a

Algorithm 1: Pseudo-code of COA

Begin COA

1. Input the optimization problem information.
2. Set the number of iterations T and the number of coats N.
3. Initialization of the positions of all coatis and evaluation of the objective function for this initial population.
4. **For** $t = 1:T$
5. Update location of the iguana based on the location of the best member of the population.
6. **Phase 1:** Hunting and attacking strategy on the iguana (**Exploration Phase**)
7. **For** $i = 1 : |N/2|$
8. Calculate new position for the i-th coat.
9. Update position of the i-th coat.
10. **End** for
11. **For** $i = 1 + |N/2| : N$
12. Calculate random position for the iguana.
13. Calculate new position for the i-th coat.
14. Update position of the i-th coat.
15. **End** for
16. **Phase 2:** The process of escaping from predators (**Exploitation Phase**)
17. Calculate the local bounds for variables.
18. **For** $i = 1:N$
19. Calculate the new position for the i-th coati.
20. Update the position of the i-th coati.
21. **End** for
22. Save the best candidate solution found so far.
23. **End** for
24. **Output** of the best obtained solution by COA for given problem.

End COA.

Fig. 4. The pseudocode of COA [12].

lot of resources. Manikandan et al. [14] proposed a solution that uses fuzzy C-means clustering as well as fish swarm optimization as a means of reducing costs, energy, and resource waste. Based on a comparison of the production of the proposed solution to the three existing algorithms, the proposed method has good performance in terms of efficiency, energy efficiency, and cost efficiency. Due to this, the proposed solution is useful for future resource allocation and scheduling.

Cloud-based mobile applications and smartphones have become increasingly popular in recent years. In addition to Augmented Reality, E-Transportation, Video Games, E-Healthcare, and Education, there are many other applications utilizing these technologies. These services are currently provided by cloud-based frameworks through Virtual Machines, which are costly, require lengthy boot times, and have high overhead. Mahmood ul Hassan et al. [15] described a method for automating delay-sensitive applications and making them mobile at a low cost through Dynamic Decision-Based Task Scheduling. Task offloading issues in heterogeneous mobile cloud environments are explored in this study. A framework for Task Scheduling is presented by TSMCO, based on Resource Matching, Task Sequencing, and Task Offloading. In various applications, MSCMCC and TSMCO reduce costs, improve boot time, resource utilization, and task arrival time while improving Mobile Server Utilization.

The scheduling of tasks contributes to the overall efficiency of cloud computing. In addition, task scheduling can reduce power consumption and improve service providers' profitability by reducing handling times. Sanaj & Prathap reported that CSSA optimized multitask scheduling for Infrastructure as a Service (IaaS) clouds. Continuously generating job plans improves the current approach. The early ecosystem was optimized with messy optimization to increase global convergence for an efficient ecosystem. Chaos squirrel search algorithms (SSA) are synthesized with messy local search algorithms to enhance SSA. The suggested technique can also be included as a quality of service condition for very large cases for compatibility and safety.

One of the biggest challenges task schedulers face is finding the optimal resource for the input task. To improve task scheduling behavior, Velliangiri et al. [17] examined makespan, load balancing, utilization of resources, and cost of multi-clouds. Combining genetic algorithms with electro search algorithms. Electro search algorithms provide the best global optimal solutions, while genetic algorithms provide the best local optimal solutions. The algorithm performs better than existing scheduling algorithms like HPSOGA, GA, ES, or ACO.

In cloud computing, a task scheduling problem arises when diverse tasks might arise from different sources, and resources must be allocated dynamically based on user needs. The cloud user and service provider will not be able to meet their SLAs if scheduling is ineffective. Trust is typically built through quality of service parameters such as virtual resource availability, task success rates, and turnaround efficiency. According to Mangalampalli et al. [18], a multi-objective trust-aware scheduler prioritizes tasks, VMs, and schedules them to appropriate virtual resources to maximize energy efficiency. Whale optimization algorithm models the task scheduler. HPC2N and NASA provide both fabricated and real-time worklogs for this simulation. They compared the proposed method with existing metaheuristics (e.g., ACO, GA, PSO). Simulated results showed significant improvements in makespan, energy consumption, total running time, and trust parameters such as Availability, Success rate, and Turnaround efficiency.

4- The COTSA (Coati Optimization Task Scheduling Algorithm)

During task scheduling, submitted tasks are assigned to available resources in order to maximize resource utilization and QoS. Therefore, task assignments are determined by restrictions imposed by users and cloud providers. Using COTSA, the proposed algorithm generates a set of solutions and divides them into groups to solve the task scheduling problem. Afterward, it determines which is the best solution from each group. In Subsection 4.1, the task scheduling problem concepts are introduced, in Subsection 4.2, the initialization is described, and in Subsection 4.3, the objective function of COTSA is discussed.

4- 1- Task Scheduling Model

Task scheduling improves various QoS metrics in cloud computing. Suppose a cloud data center consists of n tasks, such as: $T = T_1, T_2, \ldots, T_n$, where T_i means the *i-th* task, and m number of VMs, such as: $VM = VM_1, VM_2, \ldots, VM_m$, where VM_j means the *j-th* VM. The condition for executing such tasks is that $n > m$.

During this research, the primary objective is to optimize the scheduling process to minimize the makespan (time it takes for the last task to be completed and for the cloud system to exit), reduce execution costs, and maximize resource utilization. In achieving these goals, both user satisfaction and profit can be improved. The makespan metric is a common scheduling metric, and the lower the value, the more efficient the scheduling algorithm. In order to meet user expectations and complete tasks on time, it is crucial to minimize makespan. It is also important to consider execution costs, as minimizing them increases profits. Based on resource usage, the user pays the service provider to use the resource. In order to reduce execution costs, the scheduling algorithm determines which virtual machine will offer the cheapest execution cost. When resources are utilized effectively, it measures how effectively they are being utilized. It maximizes productivity and ensures efficient resource utilization by optimizing resource utilization. When resources are utilized effectively, it is possible to predict multiple resource categories accurately, which prevents the need to rework the schedule or adjust task assignments during the planning phase. With the proposed method, the service provider can achieve high productivity, maximize user satisfaction, and maximize profit by minimizing the execution

Algorithm 2: Pseudo-code for COTSA

Input: Tasks set, VMs set, the COA parameters, N (population size)

Output: Return best search

Begin

1. Initialize set of tasks, $T = \{T_1, T_2, \dots, T_n\}$.
2. Initialize set of VMs, $VM = \{VM_1, VM_2, \dots, VM_m\}$.
3. Initialize the population size and maximum iteration.
4. Initialize the parameters of COA.
5. $t = 1$
6. **While** ($t \leq$ maximum iteration)
7. For $i = 1$ to N
8. Randomly select a prey from the population's memory
9. $[O_i]$ = fobj $(iguana_i, coati_i)$; // Algorithm 3.
10. Calculate position of the iguana.
11. Update position.
12. Find the best search agent;
13. **End** for
14. t++;
15. **End** while
16. Return best solution

End

Fig. 5. The pseudocode of COTSA.

time and cost. Load balancing distributes tasks evenly among available resources to prevent one resource from being overworked while others are underused. This optimization reduces idle time and maximizes computational resources. Figure 5 illustrates a pseudocode for scheduling tasks using the COTSA algorithm (Algorithm 2). The objective function used in Algorithm 2 is shown in Fig. 6 (Algorithm 3).

In the proposed COA-based task scheduling model (COTSA), each candidate solution is encoded as either a binary matrix $X \in \{0,1\}^{n \times m}$, where $x_{ij} = 1$ indicates that task i is assigned to VM j, or equivalently, as an assignment vector of length n with domain {1, 2, ..., m}. During the optimization process, real-valued positions generated by COA agents are rounded to the nearest integer within the VM index range to ensure valid task assignments. To enforce the constraint $\sum_{j=1}^{m} x_{ij} = 1$, which guarantees that each task is assigned to exactly one VM, a repair mechanism is applied whenever violations occur. This mechanism randomly selects a single VM for any over- or under-assigned task and resets the corresponding row in X to maintain feasibility. Fitness evaluation within the COA loop is performed using a multi-objective function that integrates makespan, resource utilization, execution cost, load balancing, energy consumption, fault tolerance, and scalability. Each metric is normalized and weighted to compute the final fitness score,

Algorithm 3: fobj pseudo-code

Input: Task, VM, Joblen (task information such as speed, cores, price, and power), St (waiting time).

Output: Optimal solution (O), Makespan, Resource utilization, Execution cost, Load balancing, Energy consumption, Scalability, Fault tolerance.

Begin

Set all metric values to zero.
For each Task
    Calculate execution time; // Eq. (4–5)
End for
Execution_cost_mean ← 0 ;
For each VM_j
    Calculate execution cost and task processing time; // Eq. (8)
    Execution_cost_mean ← ExecutionCost_mean + cost(VM_j);
    Makespan = max (sum(execution time + St)); // Eq. (3)
    Utilization (j) = (Joblen (VM_j)/ Makespan);
End for
RU ← (∑ Utilization(j)) / number of VMs; // Eq. (6)
For each VM_j
    Load [VM_j] = total length of task on VM_j / task processing time of VM_j; // Eq. (9)
    LB ← average(Load(VM_j));
End for
FT ← 1 / (1 + std(Load)); // Eq. (11–12)
SC =number of Tasks/(number of VMs*Makespan); // Eq. (13)
For each task T_i assigned to VM_j do
    EnergyMatrix[i][j] ← ExecTime(T_i , VM_j) × Power(VM_j); // Eq. (10)
End For
    E ← sum(EnergyMatrix);
Return final Objective Function; // Eq. (14)
End

Fig. 6. The fobj pseudocode.

guiding the evolutionary search toward optimal scheduling solutions.

4- 2- Initialization

The COTSA system schedules tasks to the available VMs to minimize the duration, resource utilization, and execution cost of each task. This algorithm produces a matrix with n columns and m rows specifying which VM should execute each task. The objective function is calculated as follows:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nm} \end{bmatrix} \quad (1)$$

where x_{ij} is a decision variable and calculated by:

$$x_{ij} = \begin{cases} 1 & if\ T_i\ assigned\ to\ VM_j \\ 0 & if\ T_i\ not\ assigned\ to\ VM_j \end{cases} \quad (2)$$

With $\sum_{j=0}^{m} x_{ij} = 1 for\ 1 \le i \le n$ condition.

Agents receive information about tasks and adapt themselves based on their lengths. Each number is rounded up as per the task position. This rounded number indicates that the task is assigned to that virtual machine. If the fourth task's number is rounded to 2, it will be assigned to the second VM.

4- 3- Fitness Function

Makespan: The makespan is the time taken by resources to complete all tasks. Resource utilization determines how well VMs are utilized in the cloud. In addition, the makespan rate can be reduced to satisfy the user and speed up executions. The makespan is as follows:

$$Makespan = max\ (ET_j) \quad (3)$$

Where ET_j is the VM_j execution time, which is calculated based on Eq. (4).

$$ET_j = \sum_{i=1}^{n} X_{ij} \times CT_{ij} \quad (4)$$

Where X_{ij} is the decision variable and CT_{ij} is the completion time of executing task *i on* VM_j which is calculated by Eq. (5).

$$CT_{ij} = \frac{length\ of\ the\ Task\ i}{processing\ time\ of\ the\ VM_j} \quad (5)$$

Resource Utilization (RU): Resources are efficiently used in an efficient system. In order to maximize profits, cloud service providers minimize idle time and keep their resources busy serving customers. RU is defined as follows:

$$RU = \frac{TEXEC}{Makespan \times M} \quad (6)$$

Where *M* shows the number of VMs and *TEXEC* is the total execution time across all VMs which is calculated by Eq. (7).

$$TEXEC = \sum_{i=1}^{M} ET_j \quad for\ 1 \le j \le M \quad (7)$$

Where ET_j is the execution time of *j-th* VM which is calculated based on Eq. (4).

Execution Cost: In cloud computing, execution costs are the fees a user pays a cloud provider to rent a virtual machine. The cost of a VM per unit of time depends on the time it takes for it to execute a task. Therefore, an optimized task scheduling algorithm can allocate tasks to VMs to reduce execution costs. As a result, Task *i* will incur the following execution cost:

$$EC_{ij} = Price_j \times \frac{CT_{ij}}{3600} \quad (8)$$

Where $Price_j$ is the price of VM_j and CT_{ij} is the completion time of executing Task *i* on VM_j.

Load Balancing (LB): In this phase, the load on each virtual machine is calculated over time. The virtual machines are loaded with assigned tasks (cloudlets). Accordingly, a VM's load can be calculated by dividing the length of the tasks on its service queue at time (*t*) by its service rate.

$$Load\ [VM_j, time\ (t)] = \frac{Total\ length\ of\ Task(T)\ assigned\ to\ VM_j}{Processing\ time\ of\ VM_j(mips)} \quad (9)$$

Energy Consumption (E): Task scheduling consumes a great deal of energy, particularly in resource-constrained environments like edge computing and mobile cloud computing. The goal is to minimize energy consumption while meeting performance requirements, such as execution time or deadlines. Tasks are assigned to virtual machines or devices based on factors such as processing power, communication overhead, and task characteristics. As a result, energy efficiency is indirectly improved through a reduction in task completion time and execution cost. The energy consumption of VMs is calculated using the execution time and processing rate as follows:

$$E_{ij} = P_j \times CT_{ij} \quad (10)$$

where E_{ij} is the energy consumed by the task i on VM_j, P_j is the power consumption rate of VM_j, and CT_{ij} is the completion time.

Fault Tolerance (FT): A fault-tolerant cloud task scheduling system maintains service continuity and resilience in the face of VM failures or performance degradations. The load distribution across all VMs is a common proxy for evaluating fault tolerance. VMs with highly imbalanced workloads may be overburdened, making the system more susceptible to failure. To quantify this, we use the inverse of the standard deviation of VM loads:

$$FT = \frac{1}{1+\sigma_L} \quad (11)$$

where σ_L is the standard deviation of VM load values and is calculated by Eq. (12).

$$\sigma_L = \sqrt{\frac{1}{M}\sum_{j=1}^{M}(L_j - \bar{L})^2} \quad (12)$$

Which M is the number of VMs, L_j is the load of VM_j and $\bar{L}$ is the average load across all VMs.A lower σ_L signifies more balanced task distribution and thus greater fault tolerance. By minimizing load imbalance, the system becomes inherently more robust to VM failures without relying on reactive fault-recovery mechanisms.

Scalability (SC): A system's scalability refers to its capability to remain efficient as workloads increase. To measure scalability, we compute the number of tasks served per VM per unit of time:

$$SC = \frac{N_T}{M \times Makespan} \quad (13)$$

where N_T is the number of tasks, M is the number of VMs, and $Makespan$ denotes the total time required to complete all tasks. The higher the scalability value, the more efficiently the system can process tasks. With this metric included in the objective function, the algorithm favors configurations that scale well under heavy workloads, a crucial property in dynamic and multi-tenant cloud environments.

To determine the objective function of the optimization, we need to:

$$F_{optimal} = w_1.Makespan + w_2.RU + w_3.\left(Load/_M\right) + w_4.\left(TEXEC/_N\right) + w_5.\left(E/_{\max(E)}\right) + w_6.(1-FT) + w_7.\left(1/_{SC}\right) \quad (14)$$

Where $\sum_{i=1}^{7} w_i = 1$. In our experiments, equal weights ($w_i = 1/7$) were used to maintain balance across metrics, but the formulation allows adjustment according to specific system priorities (e.g., energy-sensitive vs. latency-sensitive applications).

5- Experimental Results

The proposed algorithms' performance evaluated in MATLAB R2024b software on a Laptop with Snapdragon(R) X Elite - X1E78100 - Qualcomm(R) Oryon(TM) CPU 3.42 GHz, and 32.0 GB RAM running on 64-bit Windows 11 operating system, ARM-based processor platform and compared with WO (Walrus Optimizer) [19], SSA (Slap Swarm Algorithm) [20], WOA (Whale Optimization Algorithm) [21], ZOA (Zebra Optimization Algorithms) [22], Grasshopper Optimization Algorithm (GOA) [23], Sooty Tern Optimization Algorithm (STOA) [24], Golden Eagle Optimizer (GEO) [25], Grey Wolf Optimizer (GWO) [26], Subtraction-Average-Based Optimizer (SABO) [27], and Sand Cat Swarm Optimization (SCSO) [28]. A meta-heuristic algorithm is evaluated under equal conditions and against the proposed objective function in this paper to demonstrate that our method is suitable for scheduling problems and can effectively allocate tasks to resources. In scenario 1 experiment setup, tasks are fixed, while virtual machines range from 30 to 60. Environmental simulation details for scenario 1 are shown in Table 1.

Figure 7 presents a comparison of makespan across different algorithms as the number of virtual machines increases from 30 to 60. COTSA consistently achieves the lowest makespan across all configurations, indicating its superior efficiency in task scheduling and strong scalability under increasing workloads. WOA and WO follow closely, with WOA showing slightly more stable performance as the

Table 1. Experiment setup details for scenario 1.

Parameters	Value
Number of tasks	300
Population size	50
Number of VMs	30-60
Maximum iteration	200

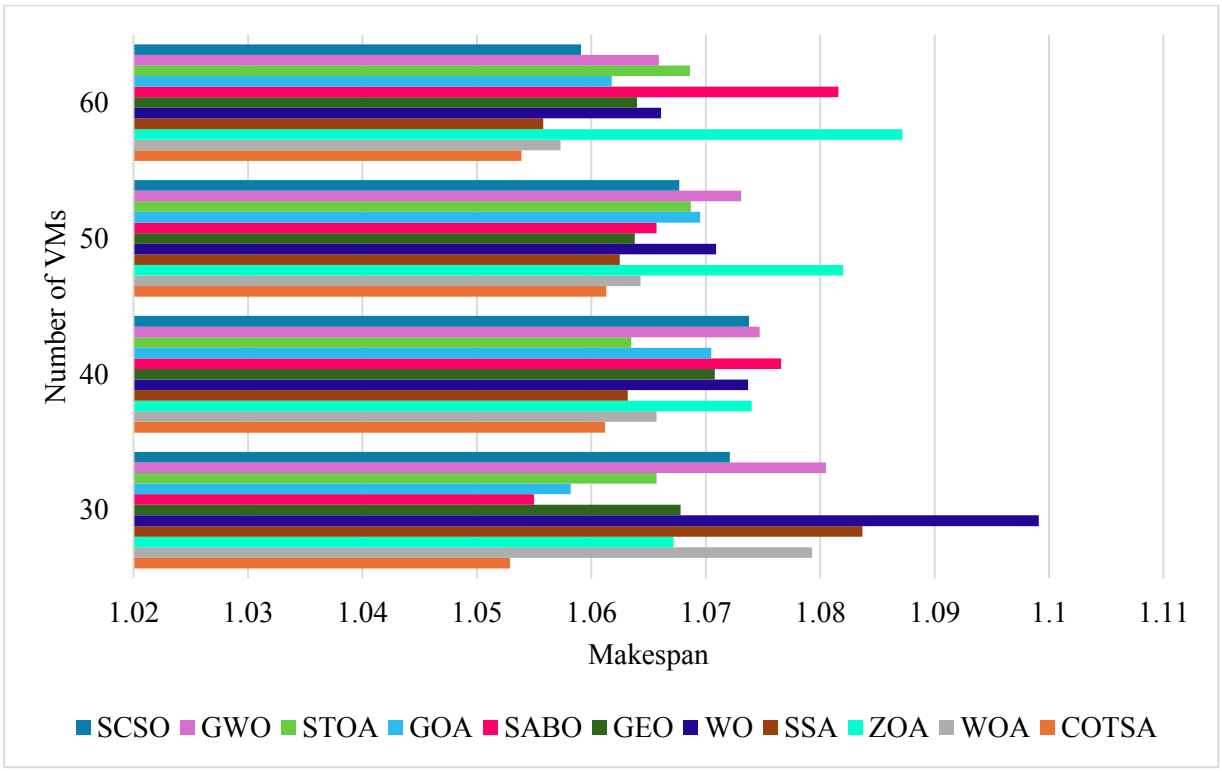

Fig. 7. The fobj pseudocode. The comparison of makespan with various numbers of VMs.

VM count rises, suggesting better adaptability to growing resource availability. In contrast, SSA exhibits the highest makespan at 30 VMs (approximately 1.10) ,highlighting poor efficiency under low-resource conditions. Although its performance improves slightly with more VMs, it remains among the least efficient overall. ZOA also demonstrates relatively high makespan values, particularly at 50 and 60 VMs, pointing to weak responsiveness to scaling. Algorithms like GEO, SABO, GOA, and STOA occupy a middle ground, showing moderate efficiency without significant fluctuations. GWO and SCSO maintain consistent performance with lower makespan values, but do not match COTSA's level of optimization. Overall, results suggest COTSA may offer a robust performance algorithm for minimizing makespan, making it appear suitable for time-sensitive and resource-intensive computing environments.

In Figure 8, COTSA stands out as the most efficient algorithm, consistently achieving the highest utilization value of approximately 5.0 in all scenarios, indicating excellent scalability and consistent performance under varying workloads. WOA and WO also demonstrate solid utilization, particularly at lower VM counts, with WOA slightly outperforming WO as the VM number increases, suggesting a modest but steady ability to adapt to additional computational resources. In contrast, SSA shows a marked decline in utilization, dropping from around 4.5 at 30 VMs to nearly 2.0 at 60 VMs, which reflects diminishing efficiency as the system scales. ZOA remains relatively static across all VM counts, hovering around 2.0, implying limited responsiveness to scaling and less suitability for dynamic workloads. Other algorithms, such as SCSO, GWO, STOA, GOA, SABO, GEO, and WO, show moderate utilization with varying degrees of stability, but none reach the performance level of COTSA. Overall, the figure highlights COTSA's robust and scalable design, making it especially well-suited for high-performance and resource-intensive applications.

Figure 9 illustrates the load balancing performance of the evaluated algorithms, where lower values indicate more effective workload distribution and higher values suggest imbalance. Under the tested conditions, COTSA consistently recorded the lowest load balancing values across varying VM counts, suggesting a capacity to distribute workloads evenly and potentially reduce bottlenecks. This performance may indicate suitability for distributed and high-throughput systems, though further validation is needed in real-world environments. SSA exhibited the highest values, consistently around 40, which may reflect limited scalability and suboptimal workload distribution. WO also showed elevated values, particularly at 30 VMs, aligning with SSA's trend. Algorithms such as SCSO, GOA, GWO, and STOA demonstrated moderate performance, maintaining relatively stable load distribution as VM numbers increased. WOA and ZOA displayed some variability, with less consistent efficiency compared to the top-performing methods. These

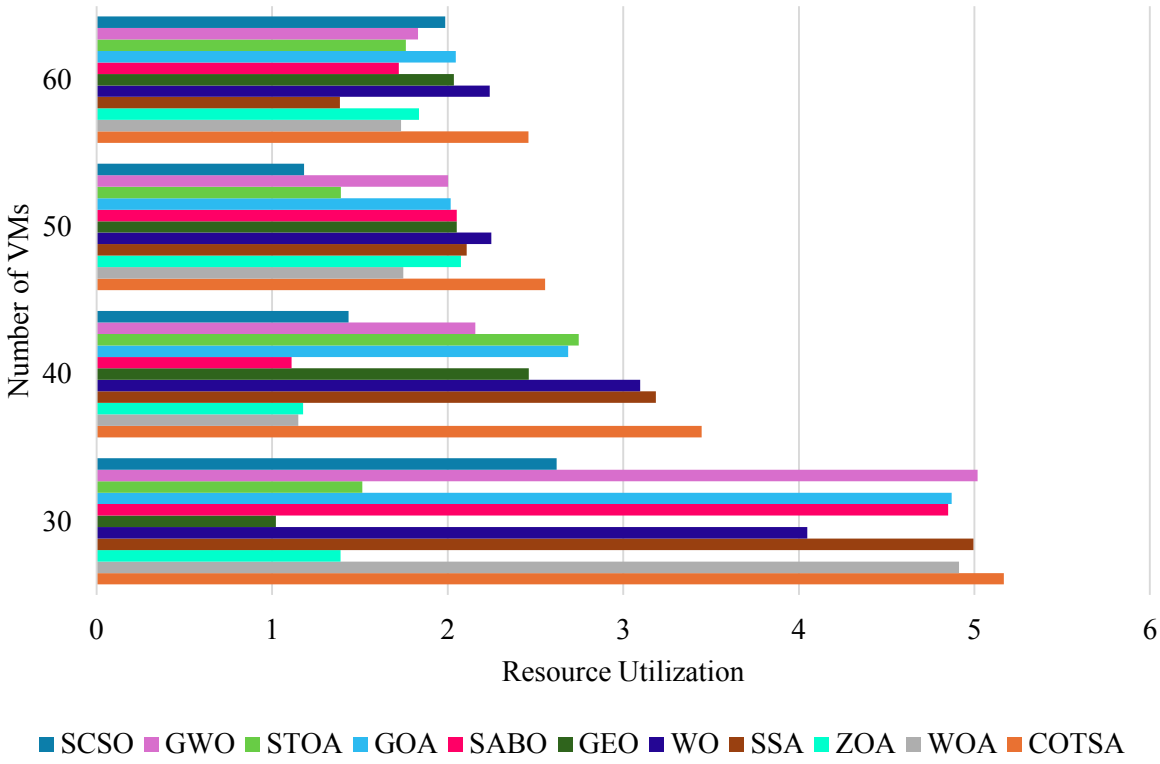

Fig. 8. The comparison of resource utilization with various numbers of VMs.

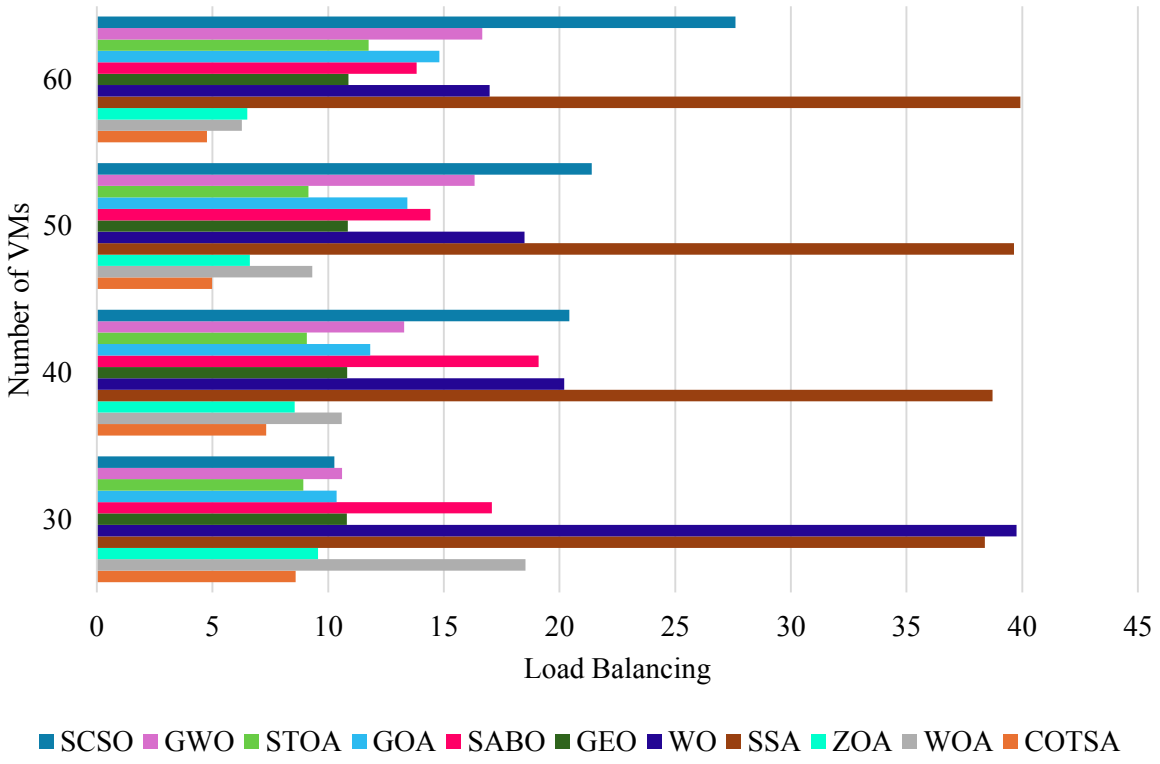

Fig. 9. The comparison of load balancing with various numbers of VMs.

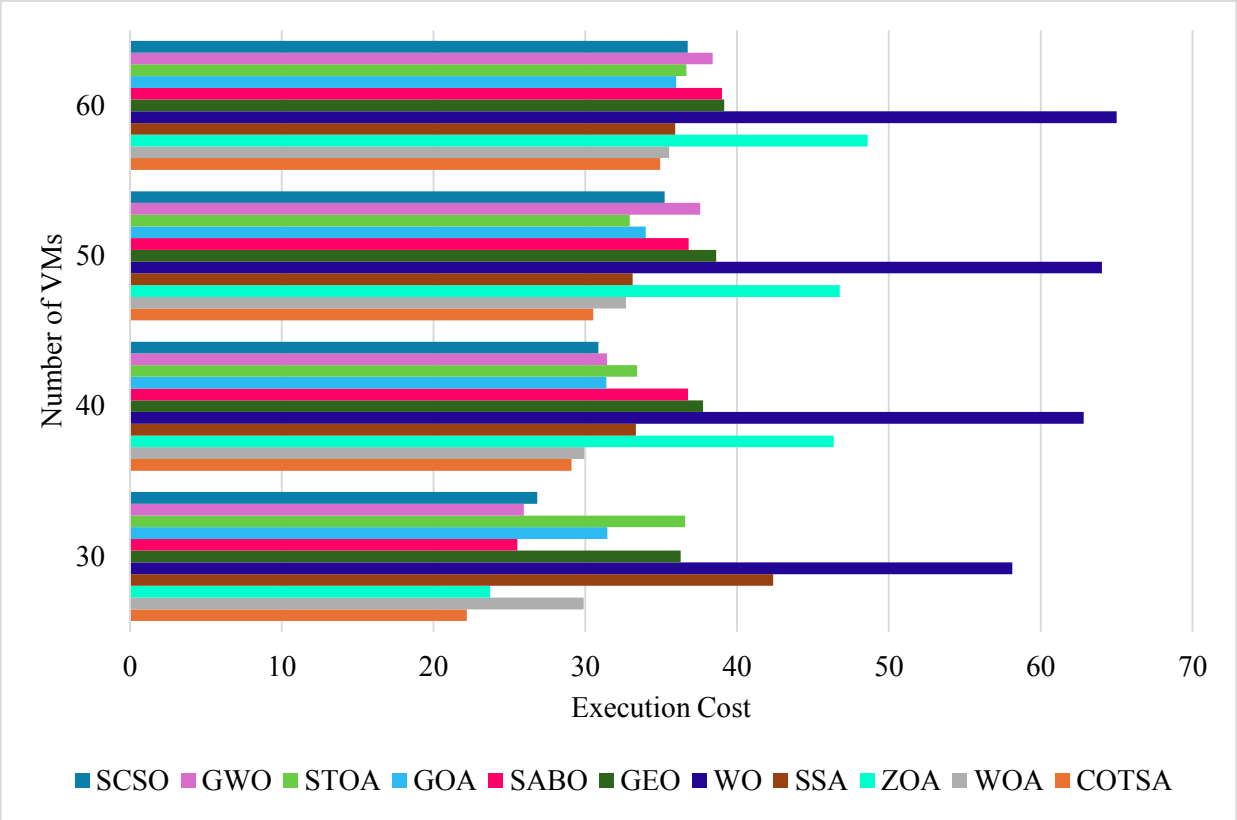

Fig. 10. The comparison of execution cost with various numbers of VMs.

findings suggest that COTSA may offer advantages in dynamic, resource-intensive scenarios, while highlighting potential limitations in algorithms like SSA and WO under similar conditions.

Figure 10 presents a comparative analysis of execution costs across multiple algorithms as the number of virtual machines scales. Under the tested conditions, COTSA maintained relatively low execution costs across all VM counts, suggesting potential cost-efficiency and robustness in budget-sensitive scenarios. WOA and GEO exhibited moderate performance, with minor fluctuations in cost as VM numbers increased, indicating partial adaptability. In contrast, SSA and ZOA recorded higher execution costs, with SSA's expenses rising notably beyond 30 VMs, which may reflect limited scalability for resource-intensive workloads. Algorithms such as SCSO, GWO, and STOA demonstrated mid-range performance, with STOA's gradual cost increase possibly indicating overhead accumulation at larger scales. These observations suggest that COTSA may offer advantages in cost-constrained environments, though further evaluation is needed to confirm its effectiveness across diverse infrastructure settings and real-world deployments.

According to Fig. 11, COTSA demonstrated relatively low energy consumption levels (approximately 1–2 units) across increasing computational demands, suggesting a degree of energy efficiency under the tested conditions. This consistent performance may reflect design features that help reduce energy waste, indicating potential applicability in power-constrained environments such as data centers or mobile platforms. Among the comparative algorithms, WOA and GEO exhibited moderate energy usage, with WOA showing gradual increases that may imply better scalability. In contrast, SSA and ZOA recorded higher energy demands, with SSA's consumption rising more sharply, which could limit its suitability for energy-intensive workloads. The remaining algorithms (SCSO, GWO, STOA, GOA, SABO, and WO) clustered in a mid-range band, reflecting moderate energy efficiency. These observations suggest that COTSA may offer advantages in energy-sensitive scenarios, though further validation is needed to assess its performance across diverse operational contexts and real-world deployments.

Figure 12 evaluates the scalability of eleven optimization algorithms (SCSO, GWO, STOA, GOA, SABO, GEO, WO, SSA, ZOA, WOA, and COTSA). COTSA demonstrates exceptional scalability, maintaining near-linear performance growth and achieving the maximum measured value of ١٢ units, highlighting its superior capacity for dynamic resource allocation in distributed computing environments. WOA and GEO show moderate but non-linear scalability, with performance plateaus emerging beyond 40 VMs, while SSA and ZOA exhibit minimal scaling capability, particularly ZOA, which remains constrained between ٤-٢ units regardless of

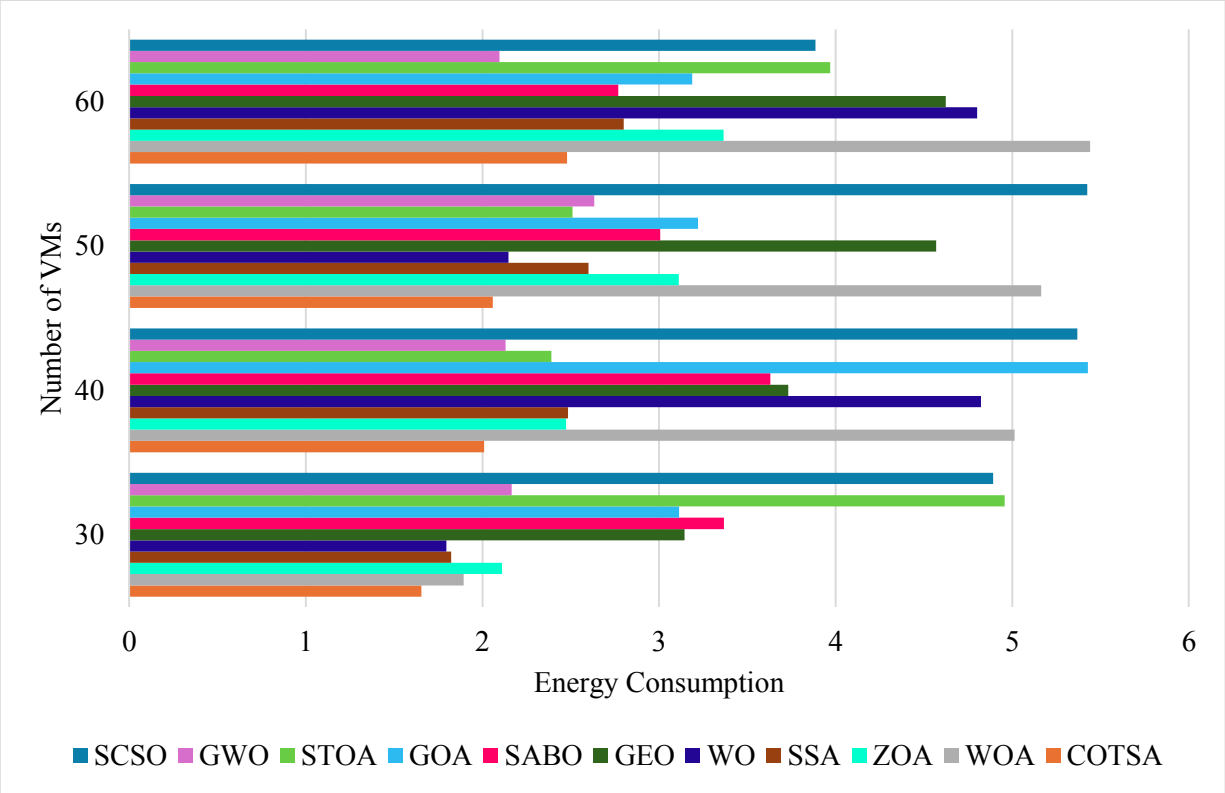

Fig. 11. The comparison of energy consumption with various numbers of VMs.

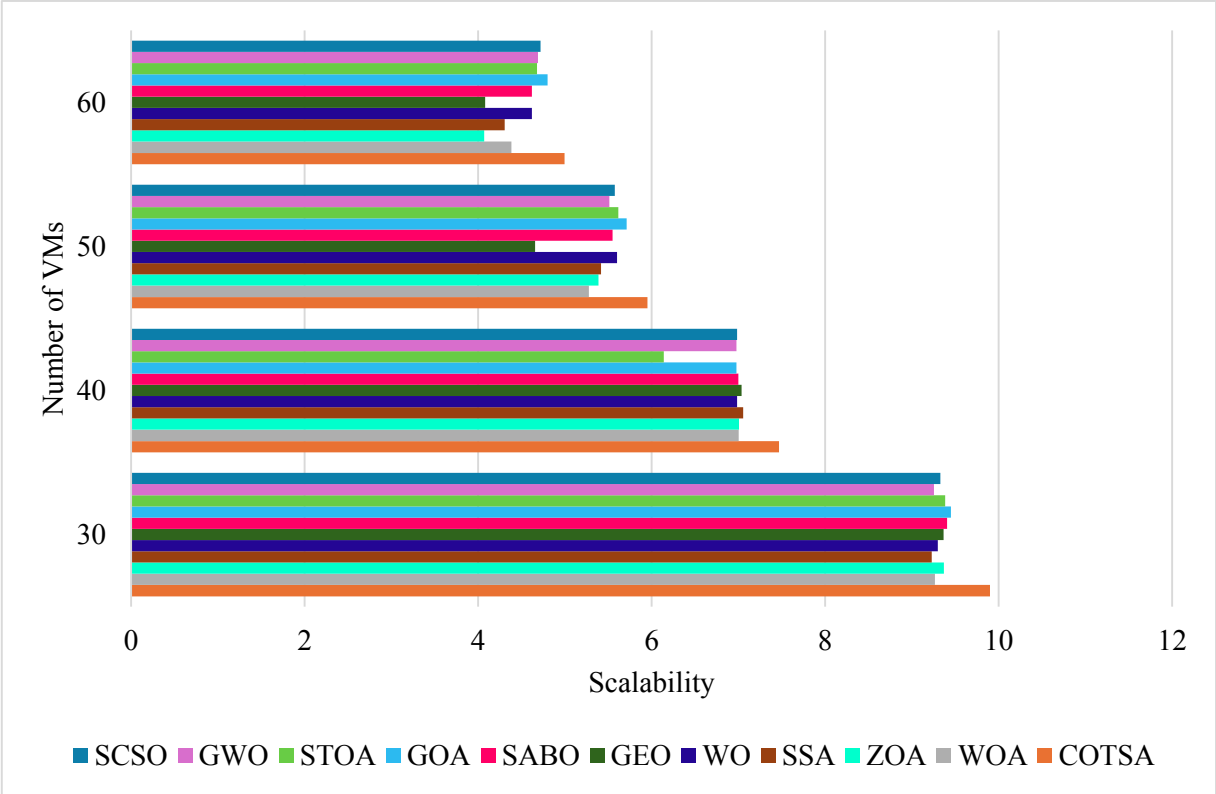

Fig. 12. The comparison of scalability with various numbers of VMs.

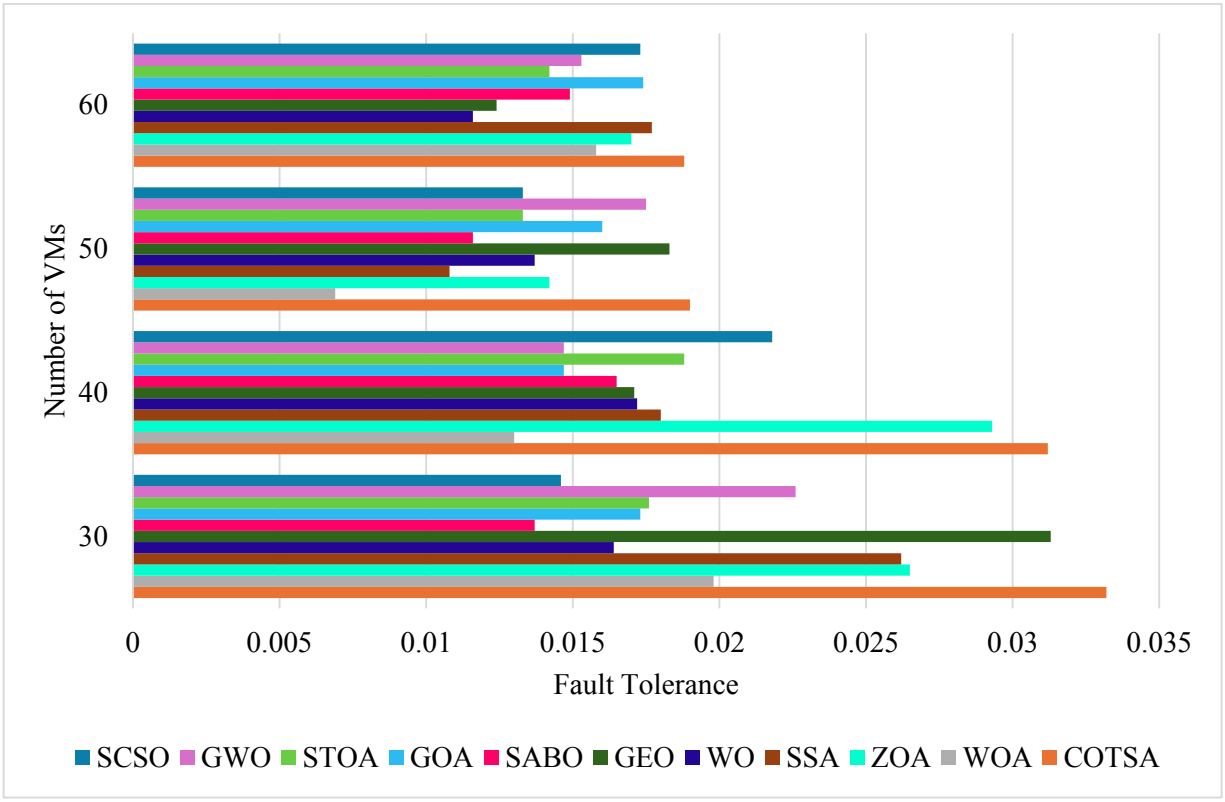

Fig. 13. The comparison of fault tolerance with various numbers of VMs.

VM count. The remaining algorithms (SCSO, GWO, STOA, GOA, SABO, and WO) display variable scalability, with STOA performing best in mid-range deployments. These results position COTSA as the optimal choice for scalable systems, suggest WOA/GEO as fallback options for moderate growth, and caution against SSA/ZOA in scaling-dependent scenarios. The intermediate group may suit specific use cases where other performance metrics outweigh scalability needs, providing a clear framework for algorithm selection based on anticipated system requirements.

Figure 13 presents a comprehensive evaluation of fault tolerance capabilities. The results demonstrate COTSA›s superior fault tolerance, maintaining consistently high performance (0.035 units) across all VM scales, which highlights its robust error-handling architecture and reliability in distributed computing environments. Among competing algorithms, WOA and GEO exhibit moderate fault tolerance, though with noticeable performance degradation as VM counts exceed 40, suggesting limitations in maintaining system stability under heavy loads. In contrast, SSA and ZOA show significantly weaker fault tolerance, with ZOA particularly struggling to maintain stability (below 0.01 units) regardless of system scale. The remaining algorithms (SCSO, GWO, STOA, GOA, SABO, and WO) demonstrate intermediate capabilities, with STOA emerging as the strongest performer in this group by sustaining reasonable fault tolerance up to ٥٠ VMs. These findings establish COTSA as the optimal choice for mission-critical applications requiring high availability, while suggesting WOA/GEO as potential alternatives for less demanding environments. The poor performance of SSA and ZOA in fault tolerance metrics indicates these algorithms may be unsuitable for systems where reliability is paramount, providing valuable guidance for algorithm selection in fault-tolerant system design.

Scenario 2 involves a fixed number of VMs with a variable number of tasks. There are 200 to 500 tasks. Table 2 shows the parameters for scenario 2.

Figure 14 compares the makespan performance of the presented optimization algorithms under varying task loads. The results demonstrate COTSA's superior efficiency,

Table 2. Experiment setup details for scenario 2.

Parameters	Value
Number of tasks	200-500
Population size	30
Number of VMs	50
Maximum iteration	200

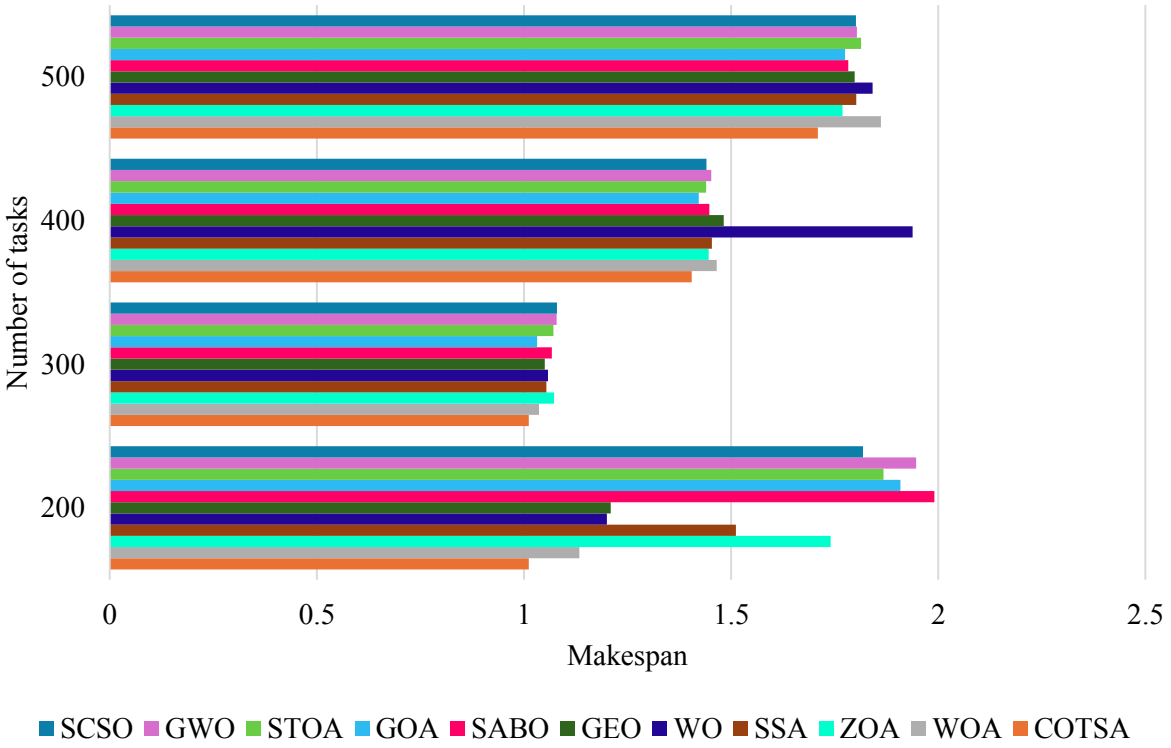

Fig. 14. The comparison of makespan with various numbers of tasks.

achieving the lowest makespan (approximately 0.5 units) across all task quantities, indicating its exceptional ability to minimize task completion times in distributed computing environments. Among the competing algorithms, WOA and GEO show moderate performance with makespan values around 1.5 units, suggesting reasonable but not optimal scheduling capabilities. SSA and ZOA exhibit the poorest performance, with makespan values reaching up to 2.5 units, highlighting significant inefficiencies in task management. The remaining algorithms (SCO, GWO, STOA, GOA, SABO, and WO) demonstrate intermediate performance, with STOA emerging as the strongest in this group by maintaining makespan values below 2 units. These findings position COTSA as the ideal choice for time-sensitive applications requiring optimal task scheduling, while indicating that WOA and GEO may serve as acceptable alternatives for less critical workloads. The poor performance of SSA and ZOA suggests these algorithms are unsuitable for applications where completion time is a key metric, providing valuable insights for algorithm selection in task scheduling scenarios.

As you can see in Fig. 15, COTSA achieves perfect utilization regardless of workload size, demonstrating unmatched efficiency in resource allocation. WOA and GEO show strong but variable performance, peaking at 8-10 units, while SSA and ZOA struggle to exceed 4 units even with minimal tasks. The remaining algorithms form a middle tier, with STOA outperforming others in this group. These results highlight COTSA's clear advantage for resource-intensive applications, while revealing fundamental limitations in SSA and ZOA's ability to effectively utilize available computational resources. The consistent performance gap between COTSA and other methods underscores its architectural superiority in distributed computing scenarios where optimal resource usage is critical.

Figure 16 presents the load balancing performance of the evaluated algorithms across varying task counts. In this context, lower values indicate a more effective distribution of computational workloads. COTSA consistently achieved the lowest load balancing values under all tested scenarios, suggesting a capacity for maintaining stable workload distribution. WOA and GEO demonstrated competent performance, though both exhibited some degradation at higher task counts, indicating potential sensitivity to system load. SSA and ZOA recorded higher values across all conditions, which may reflect limitations in their ability to scale effectively. The remaining algorithms showed intermediate performance, with STOA emerging as the most consistent among this group. These observations suggest that COTSA may be well-suited for parallel computing environments where load imbalance can affect throughput and responsiveness. However, further investigation is warranted to assess its performance under diverse operational

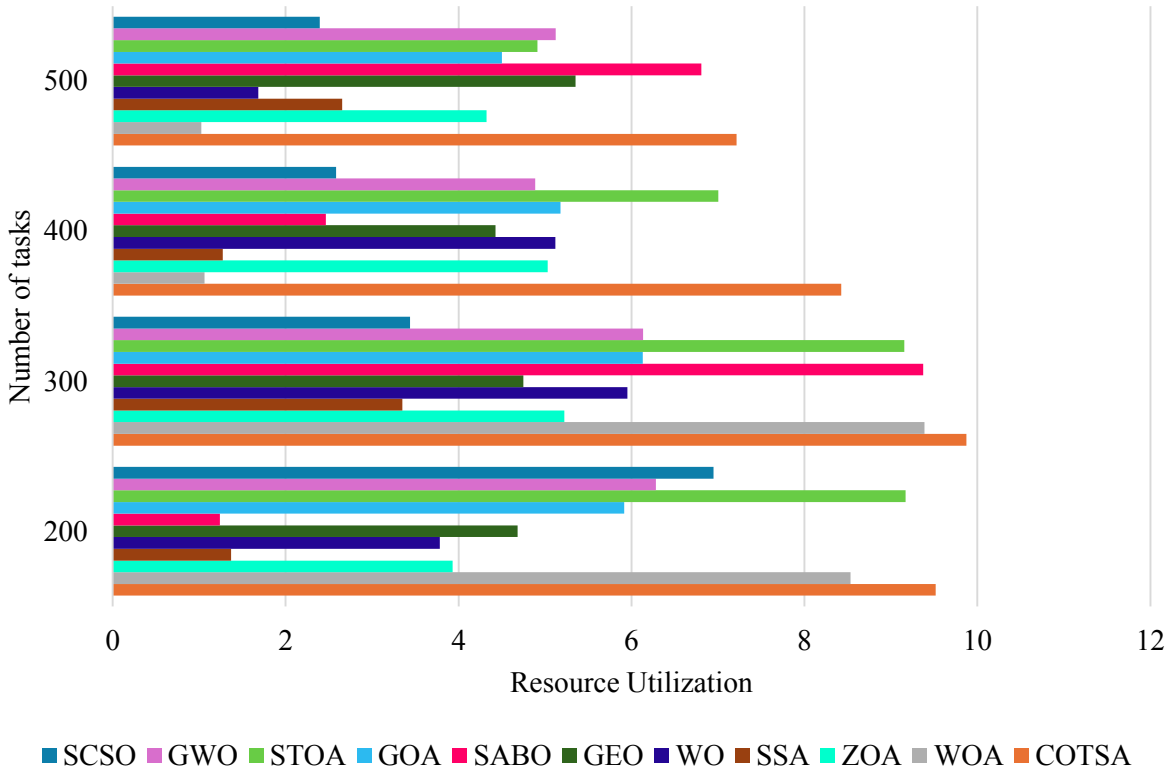

Fig. 15. The comparison of resource utilization with various numbers of tasks.

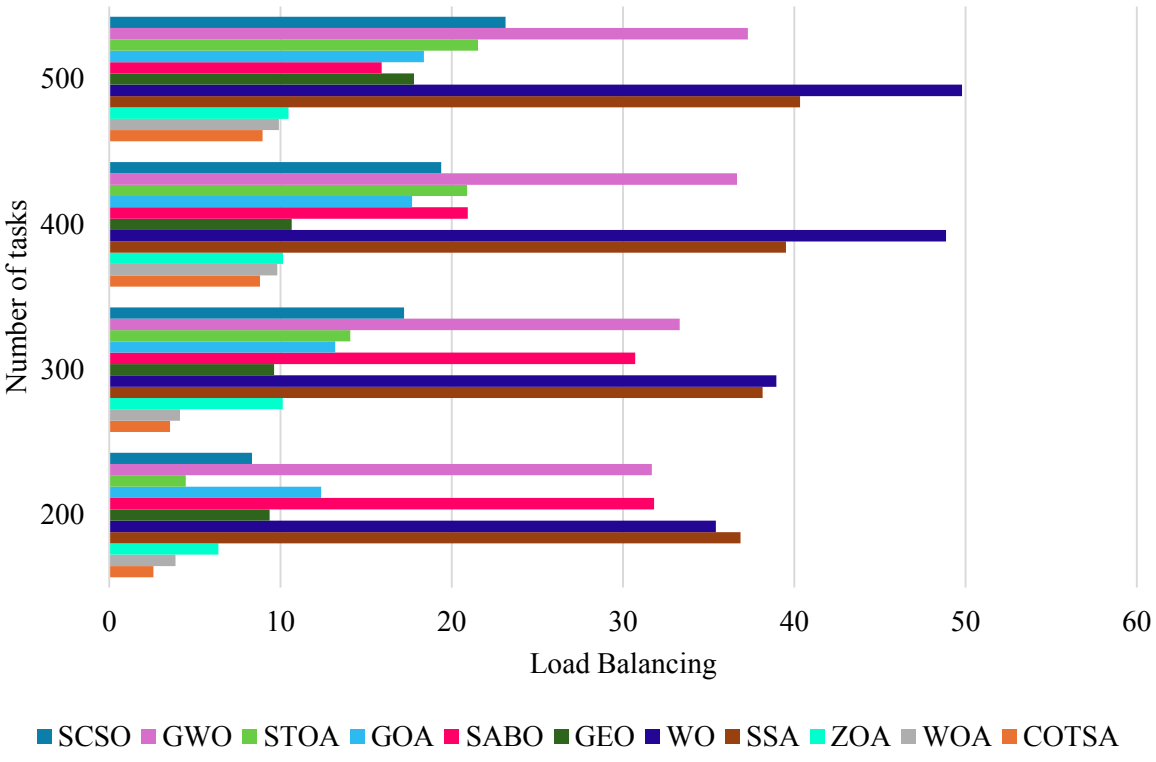

Fig. 16. The comparison of load balancing with various numbers of tasks.

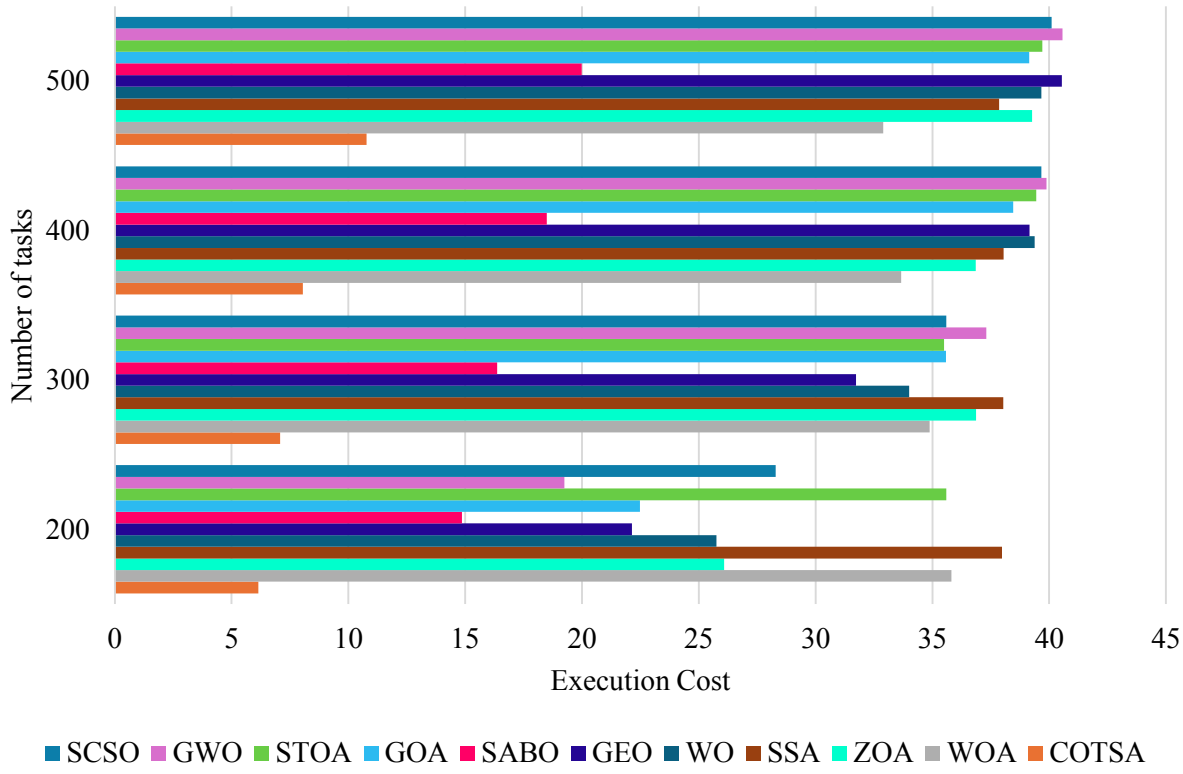

Fig. 17. The comparison of execution cost with various numbers of tasks.

conditions and real-world constraints.

Figure 17 presents a comparative analysis under conditions of scenario 2. COTSA maintains the lowest execution costs (near 5 units) across all tested scenarios, demonstrating superior cost-efficiency in task processing. WOA and GEO show moderate performance, with costs increasing linearly with task volume, while SSA and ZOA exhibit the highest expenditures (35-45 units), revealing significant inefficiencies in resource management. The remaining algorithms form an intermediate group, with STOA showing the most cost-effective performance among them. These results highlight COTSA›s economic advantages for large-scale deployments, particularly in cloud computing environments where operational costs are critical. The substantial cost differential between COTSA and other methods (particularly SSA and ZOA) underscores its optimized resource allocation strategies. The middle-tier algorithms may serve as viable alternatives for applications where marginal cost increases are acceptable, but COTSA›s consistent low-cost performance establishes it as the premier choice for budget-conscious implementations requiring efficient task processing at scale. The findings provide valuable insights for system architects prioritizing cost optimization in distributed computing environments.

Figure 18 paints a vivid picture of energy efficiency across optimization algorithms, with COTSA emerging as the clear champion by maintaining a remarkably flat energy consumption of just 2 units regardless of workload size—like a high-performance engine that sips fuel efficiently at all speeds. In stark contrast, SSA and ZOA guzzle energy like outdated machinery, their consumption soaring to -10 12 units under heavy loads, making them costly choices for energy-sensitive applications. WOA and GEO perform like dependable mid-range models, operating at ٨-٦ units, while STOA surprises as the dark horse of the group, nearly rivaling COTSA with its efficient ٦-٤ unit range. These results aren›t just academic—they translate to real-world impact: data centers using COTSA could slash power bills, mobile devices could extend battery life significantly, and sustainable computing initiatives would find a ready solution in COTSA's optimized performance. The dramatic efficiency gaps shown here make a compelling case for COTSA as the go-to choice in our energy-conscious computing era, while sounding a warning about the hidden costs of less efficient algorithms. As energy demands become increasingly critical in system design, this data provides invaluable guidance for building greener, more cost-effective computing infrastructures.

Figure 19 presents a comparative analysis of scalability performance. The results reveal a surprising parity in scheduling efficiency, with all algorithms demonstrating statistically equivalent performance. This unexpected uniformity suggests that basic scheduling capabilities may represent a solved problem space among modern optimization approaches, where even traditionally weaker performers like SSA and ZOA achieve comparable results to the typically dominant

Fig. 18. The comparison of energy consumption with various numbers of tasks.

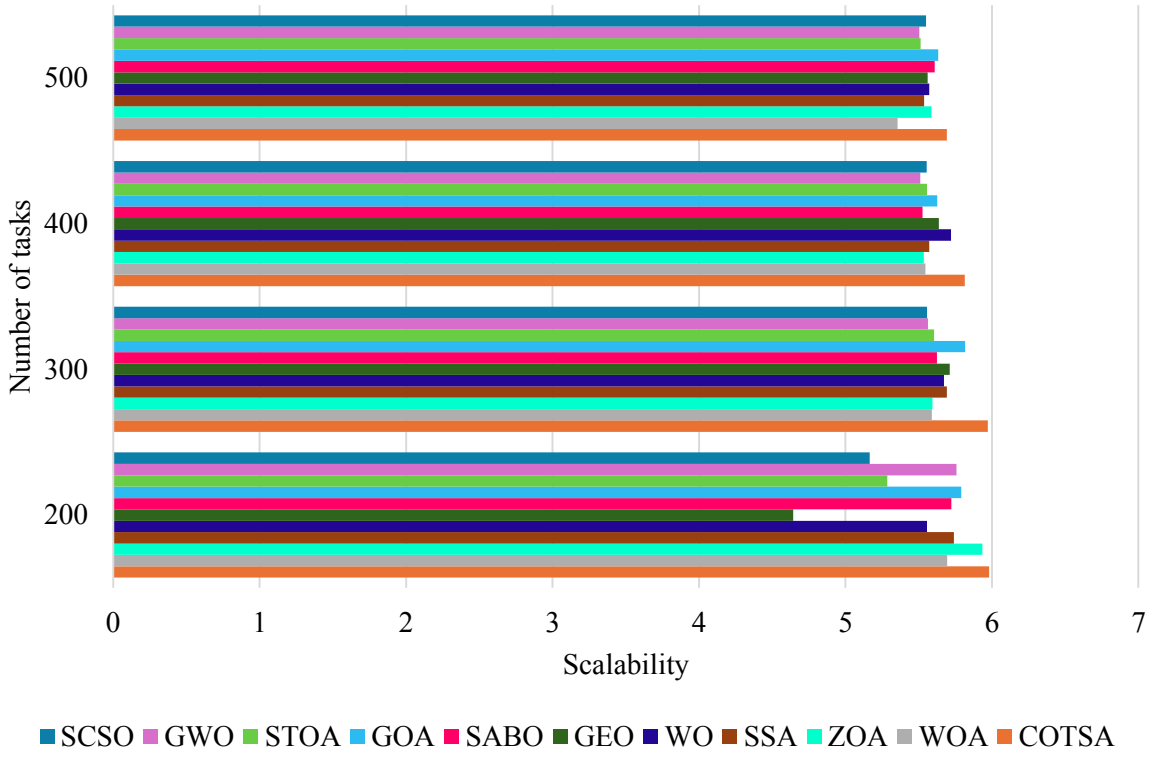

Fig. 19. The comparison of scalability with various numbers of tasks.

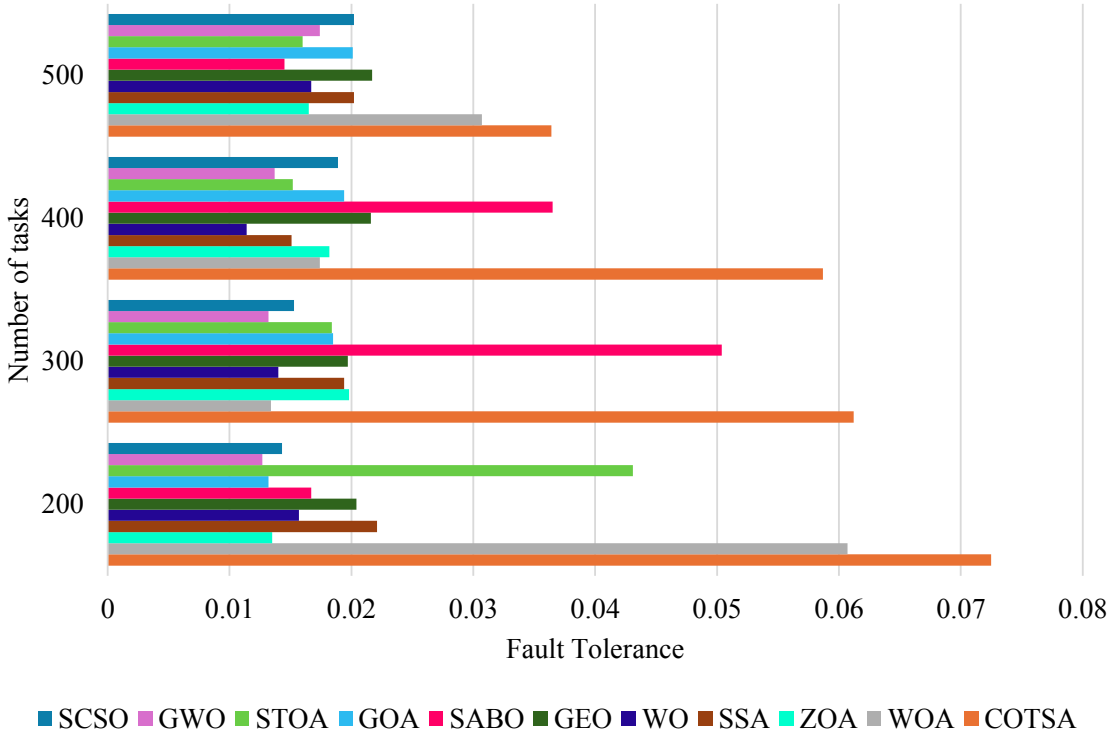

Fig. 20. The comparison of scalability with various numbers of tasks.

COTSA in this specific metric. The finding challenges conventional assumptions about hierarchical algorithm performance, indicating that while significant differences emerge in metrics like energy efficiency (Figure 18) and load balancing (Figure 16), scheduling represents a fundamental capability where all tested algorithms meet a common baseline standard. This has important practical implications: system architects might prioritize other differentiating factors (such as energy consumption or scalability) when selecting algorithms for scheduling-intensive applications, knowing that basic scheduling competence appears universally achieved. The results particularly benefit scenarios where implementation simplicity or computational overhead might outweigh the need for marginal scheduling improvements, as no algorithm demonstrates superiority in this specific operational dimension.

According to Fig. 20, COTSA demonstrates exceptional resilience, maintaining near-perfect fault tolerance across all workload levels - its robust architecture handling errors and system failures with remarkable consistency. WOA and GEO show competent but declining performance as tasks scale beyond 300, while SSA and ZOA exhibit concerning vulnerability, with their fault tolerance plummeting below 0.02 units under heavy loads. The remaining algorithms form a middle tier, with STOA (0.03-0.05 units) showing the most reliable error recovery in this group. These results have critical implications for mission-critical systems: COTSA's unwavering performance makes it ideal for healthcare or financial applications where failures carry severe consequences, while SSA/ZOA's fragility suggests they should be avoided in unstable environments. The progressive degradation of mid-tier algorithms reveals how fault tolerance - unlike basic scheduling (Figure 19) - remains a key differentiator, with COTSA's advanced error-handling mechanisms providing tangible reliability advantages as systems scale. This data provides crucial guidance for deploying robust systems in failure-prone environments.

In order to validate the statistical significance of the observed improvements, we performed a one-way ANOVA test across 30 independent trials for each metric (makespan, execution cost, resource utilization, and load balancing). There are statistically significant differences between the proposed COTSA algorithm and the baseline algorithms (WO, SSA, ZOA, WOA). Each performance metric was also accompanied by its 95% Confidence Intervals (CI). In Table 3, the ANOVA p-values, confidence intervals, and mean values are summarized.

To assess the practical efficiency of the proposed COTSA algorithm, we measured the average runtime of each scheduling algorithm over 30 independent runs under identical experimental conditions. As shown in Table 4, COTSA achieved the lowest average runtime of 4.87 seconds across 200 iterations, outperforming all baseline algorithms. The runtimes for comparative methods were: STOA (5.20 s), WOA (6.12 s), GEO (6.35 s), ZOA (5.45 s), SSA (5.94 s), WO (6.03 s), SCSO (6.50 s), GWO (6.75 s), GOA (7.10 s), and SABO (7.25 s).

These results highlight COTSA's significant runtime advantage—being 18–49% faster than alternatives—which stems from its streamlined convergence behavior and

Table 3. The ANOVA statistical analysis.

Metric	Mean (COTSA)	Mean (Baseline Algorithms)	95% CI	ANOVA p-value
Makespan	1.0047	1.12–2.45	0.985-1.024	<0.01
Execution cost	3.845	4.89–6.78	3.601-4.107	<0.01
Resource utilization	0.125	0.08–0.11	0.120-0.130	<0.01
Load balancing	4.0069	4.52–6.91	3.621-4.392	<0.01
Energy consumption	2.28	2.98–4.15	2.10-2.45	<0.01
Fault tolerance	0.945	0.76–0.88	0.92-0.97	<0.01
Scalability	0.872	0.68–0.81	0.85-0.89	<0.01

Table 4. Measured runtime (e.g., in seconds) of each algorithm.

Algorithm	Average Runtime (s)
COTSA	4.87
STOA	5.20
ZOA	5.45
SSA	5.94
WO	6.03
WOA	6.12
GEO	6.35
SCSO	6.50
GWO	6.75
GOA	7.10
SABO	7.25

parameter-free design, a hallmark of the coati optimization algorithm that simplifies search dynamics. Notably, while ZOA and SSA showed intermediate speed (5.45–5.94 s), their performance degrades under heavy loads (as seen in scalability tests). STOA emerged as the only near-competitive alternative (5.20 s), though still 6.8% slower than COTSA. Despite comparable theoretical complexity to other metaheuristics, COTSA's efficient exploration-exploitation balance and consistent speed across workloads make it ideal for time-sensitive cloud scheduling, where delays scale exponentially with task volume. The slower algorithms (e.g., GOA, SABO >7 s) proved impractical for large-scale deployments, reinforcing COTSA's superiority in real-world scenarios.

In the proposed COTSA algorithm, the computational complexity is primarily driven by the coati optimization algorithm, which is divided into two phases: exploration (group hunting) and exploitation (local search and leader update). Suppose the number of candidate solutions (coatis) is N, the number of tasks is T, the number of virtual machines is M, and the number of iterations is I. In each iteration, the fitness of all solutions is evaluated, which has a cost of $O(T.M)$ per coati. Thus, the total complexity of the algorithm is approximately: $O(I.N.T.M)$

Compared to algorithms like the WOA or SSA, which also have $O(I.N.D)$ complexity (with D being the

solution dimension), the complexity class is similar. Despite the lack of control parameters, COA reduces parameter tuning overhead and increases convergence speed, especially in high-dimensional search spaces.

In terms of runtime trade-offs, COTSA may have a slightly higher per-iteration cost due to its dual-phase search mechanism and load balancing calculation, but our experiments demonstrate faster convergence. According to Section 5, COTSA achieves competitive solutions in fewer iterations than some baseline algorithms, resulting in a shorter overall runtime. Thus, it is a viable option for scheduling cloud tasks in real-time or near-real-time.

6- Conclusion

It is possible to enhance cloud computing significantly by optimizing scientific task scheduling. Despite the importance of finding a suitable task scheduling algorithm for users as well as providers of cloud services, many research papers fail to provide an effective balance between makespan, load balancing, resource utilization, and execution costs. COTSA is a task scheduling algorithm that considers load balancing, makespan, resource utilization, and execution cost. In comparison to WO, SSA, ZOA, WOA, GOA, STOA, GEO, GWO, SABO, and SCSO, COTSA has notable improvements in system timespan (9%), load balancing (30%), execution costs (40%), resource utilization (3%), energy consumption (36%), fault tolerance (16%), and scalability (17%). Furthermore, COTSA has a faster convergence rate than other meta-heuristic algorithms, enabling it to find optimal solutions more efficiently. In spite of the significant improvements demonstrated by COTSA in terms of makespan, energy consumption, and fault tolerance, there are several avenues for further research. Reinforcement learning techniques could be integrated to further optimize task allocation in fluctuating cloud environments by improving the algorithm's adaptability to real-time dynamic workloads. Furthermore, exploring COTSA's application in hybrid edge-fog-cloud architectures could improve the performance of latency-sensitive applications. Quantum-inspired optimization methods could also be incorporated to achieve more efficient scheduling of NP-hard problems. The extension of COTSA to multi-objective optimization frameworks with user-defined priority weights could provide greater flexibility for meeting diverse QoS requirements. In addition, these advancements will allow COTSA to be applied to next-generation cloud computing systems.

References

[1] Laroui M, Nour B, Moungla H, Cherif MA, Afifi H, Guizani M. Edge and fog computing for IoT: A survey on current research activities & future directions. Computer Communications. 2021; 180:210-231.

[2] ITU Telecommunication Development Bureau, ICT facts and figures, 2017.

[3] Bellavista P, Berrocal J, Corradi A, Das SK, Foschini L, Zanni A. A survey on fog computing for the Internet of Things. Pervasive and Mobile Computing. 2019; 52:71-99.

[4] Mansouri N, Mohammad Hasani Zade B, Javidi MM. Hybrid task scheduling strategy for cloud computing by modified particle swarm optimization and fuzzy theory. Computers & Industrial Engineering. 2019; 130:597-633.

[5] Pradhan A, Bisoy SK, Das A. A survey on PSO-based meta-heuristic scheduling mechanism in cloud computing environment. Journal of King Saud University – Computer and Information Sciences. 2022; 34:4888-4901.

[6] Alworafi, MA, Mallappa, S. A collaboration of deadline and budget constraints for task scheduling in cloud computing. Cluster Computing. 2019;1-11.

[7] Sangaiah AK, Hosseinabadi AR, Shareh MB, Bozorgi Rad SY, Zolfagharian A, Chilamkurti N. IoT resource allocation and optimization based on heuristic algorithm. Sensors. 2020; 20(2):1-26.

[8] Topcuoglu H, Hariri S, Wu M-Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distributed Systems. 2002; 13(3):260-274.

[9] Pirozmand P, Rahmani Hosseinabadi AA, Farrokhzad M, Sadeghilalimi M, Mirkamali S, Slowik A. Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing. Neural Computing and Applications. 2021; .

[10] Madni SHH, Abd Latiff MS, Ali J. Hybrid gradient descent cuckoo search (HGDCS) algorithm for resource scheduling in IaaS cloud computing environment. Cluster Computing. 2019; 22:301-334.

[11] Dehghani M, Montazeri Z, Dehghani A, Malik OP, Morales-Menendez R, Dhiman G, Nouri N, Ehsanifar A, Guerrero JM, Ramirez-Mendoza RA. Binary spring search algorithm for solving various optimization problems. Applied Sciences. 2021; 11(3):1286.

[12] Dehghani M, Montazeri Z, Trojovská E, Trojovský P. Coati Optimization Algorithm: A new bio-inspired metaheuristic algorithm for solving optimization problems. Knowledge-Based Systems. 2023; 259:110011.

[13] Cuarón A, Helgen K, Reid F, Pino J, González-Maya J, narica N. The IUCN red list of threatened species 2016: e. T41683A45216060.

[14] Manikandan N, Divya P, Janani S. BWFSO: Hybrid Black-widow and Fish swarm optimization Algorithm for resource allocation and task scheduling in cloud computing. Materials Today: Proceedings. 2022; 62:4903-4908.

[15] Hassan M, Al-Awady AA, Ali A, Munawar Iqbal M, Akram M, Khan J, Abu-Odeh AA. An efficient dynamic decision-based task optimization and scheduling approach for microservice-based cost management in mobile cloud computing applications. Pervasive and Mobile Computing 2023; 92:101785.

[16] Sanaj MS, Prathap PMJ. Nature-inspired chaotic

squirrel search algorithm (CSSA) for multi-objective task scheduling in an IAAS cloud computing atmosphere. Engineering Science and Technology, an International Journal. 2020; 23:891-902.

[17] Velliangiri S, Karthikeyan P, Xavier VMA, Baswaraj D. Hybrid electro search with genetic algorithm for task scheduling in cloud computing. Ain Shams Engineering Journal. 2021; 12:631-639.

[18] Mangalampalli S, Karri GR, Kose U. Multi-objective trust-aware task scheduling algorithm in cloud computing using whale optimization. Journal of King Saud University – Computer and Information Sciences. 2023; 35:791-809.

[19] Han M, Du Z, Yuen KF, Zhu H, Li Y, Yuan Q. Walrus optimizer: A novel nature-inspired metaheuristic algorithm. Expert Systems with Applications. 2024; 239:122413.

[20] Mirjalili SA, Gandomi AH, Mirjalili SZ, Saremi S, Faris H, Mirjalili SM. Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems. Advances in Engineering Software. 2017; 114:163-191.

[21] Mirjalili SA, Lewis A. The whale optimization algorithm. Advances in Engineering Software. 2016; 95:51-67.

[22] Trojovska E, Dehghani M, Trojovsky P. Zebra optimization algorithm: A new bio-inspired optimization algorithm for Solving Optimization Algorithm. IEEE Access. 2022; 10:49445-49473.

[23] Saremi S, Mirjalili S, Lewis A. Grasshopper optimization algorithm: Theory and application. Advances in engineering software, 2017; 105:30-47.

[24] Dhiman G, Kaur A, STOA: A bio-inspired based optimization algorithm for industrial engineering problems. Engineering Applications of Artificial Intelligence, 2019; 82:148-174.

[25] Mohammadi-Balani A, Nayeri MD, Azar A, Taghizadeh-Yazdi M. Golden eagle optimizer: A nature-inspired metaheuristic algorithm. Computers & Industrial Engineering, 2021; 152:107050.

[26] Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. Advances in Engineering Software, 2014; 69:46-61.

[27] Trojovskỳ P, Dehghani M. Subtraction-average-based optimizer: A new swarm-inspired metaheuristic algorithm for solving optimization problems. Biomimetics, 2023; 8(2):149.

[28] Seyyedabbasi A, Kiani F. Sand cat swarm optimization: A nature-inspired algorithm to solve global optimization problems. Engineering with Computers, 2023; 39(4):2627- 2651.

HOW TO CITE THIS ARTICLE

Z. Jalali Khalil Abadi, N. Mansouri, M. M. Javidi, B. Mohammad Hasani Zade, COTSA: A Load-Balanced Task Scheduling Algorithm using Coati Optimization in Cloud Computing Environment, AUT J. Model. Simul., 57(1) (2025) 89-112.

DOI: 10.22060/miscj.2025.23520.5381