Task Scheduling in Cloud Computing Environment Based on Dwarf Mongoose Optimization

Olanrewaju Lawrence Abraham[1]*, Md Asri Ngadi[2], Johan Bin Mohamad Sharif[3],
Mohd Kufaisal Mohd Sidik[4], Ogunyinka Taiwo Kolawole[5]
Department of Computer Science, Universiti Teknologi Malaysia, Johor, Malaysia[1, 2, 3, 4]
Information Technology Services Department, Gateway (ICT) Polytechnic Saapade, Ogun State, Nigeria[1]
Department of Computer Science, Gateway (ICT) Polytechnic Saapade, Ogun State, Nigeria[5]

***Abstract*—The rapid advancement of the Internet and Internet of Things (IoT) technologies has significantly increased the demand for scalable and efficient cloud computing solutions. Task scheduling, a critical aspect of cloud computing, directly impacts system performance by influencing resource utilization, execution time, and operational costs. However, scheduling tasks in large-scale, dynamic cloud environments remains an NP-hard problem, with existing metaheuristic methods often struggling with scalability, convergence, and adaptability. This study proposes a novel task scheduling approach based on the dwarf mongoose optimization (DMO) algorithm. To assess its effectiveness, we conduct two experimental scenarios. The results demonstrate that, compared with existing algorithms, the proposed DMO algorithm offers faster convergence and higher accuracy in identifying optimal task scheduling solutions, particularly under large-scale task loads. We evaluated the method using the Google Cloud Jobs (GoCJ) dataset, and the findings confirm that DMO outperforms prior state-of-the-art techniques in terms of reducing makespan.**

***Keywords*—Task scheduling; cloud computing; virtual machines; dwarf mongoose optimization algorithm; Cloudsim; makespan**

I. Introduction

The exponential expansion of the Internet and the Internet of Things has resulted in a significant increase in network data, rendering cloud computing (CC) increasingly essential. Unlike other distributed computing models like grid computing, CC offers users more flexible and scalable services. To efficiently meet the rising computational demands of large-scale applications, CC now enables the rapid deployment of such applications by offering flexible and resilient resources on a pay-per-use basis [1]. CC has transformed how computational resources are allocated and used, providing unmatched scalability, flexibility, and cost-efficiency for a wide range of applications and services. These massive applications involve a vast number of jobs or tasks processed within service-oriented cloud environments. Among the various cloud service models, the most prominent are Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [2]. Fig. 1 highlights the key entities in cloud computing that contribute to ensuring continuous service delivery.

In the SaaS paradigm, cloud applications are provided to customers via the Internet and accessed by client programs such as web browsers on desktops or workstations. SaaS is frequently utilized for services including webmail, video-sharing platforms, social networking sites, and online document editors. Concurrently, PaaS provides a development environment tailored for the construction, testing, and deployment of applications. Furthermore, IaaS offers users scalable and adaptable computer resources to facilitate the deployment of extensive applications. In the IaaS paradigm, users employ virtual machines (VMs) that are provisioned with pre-configured CPU, storage, memory, and bandwidth to fulfill their individual requirements. Various VM configurations are available at different price points, giving users control over their computing resources. IaaS offers three main advantages: 1) users pay only for what they use, similar to utilities like electricity or water, allowing flexible resource scaling based on application demands; 2) direct provisioning of resources, which boosts application performance; and 3) global accessibility to rented resources at any time according to the required service level. However, determining the optimal number of resources needed to execute large-scale tasks on an IaaS cloud remains a significant and unresolved challenge [3, 4].

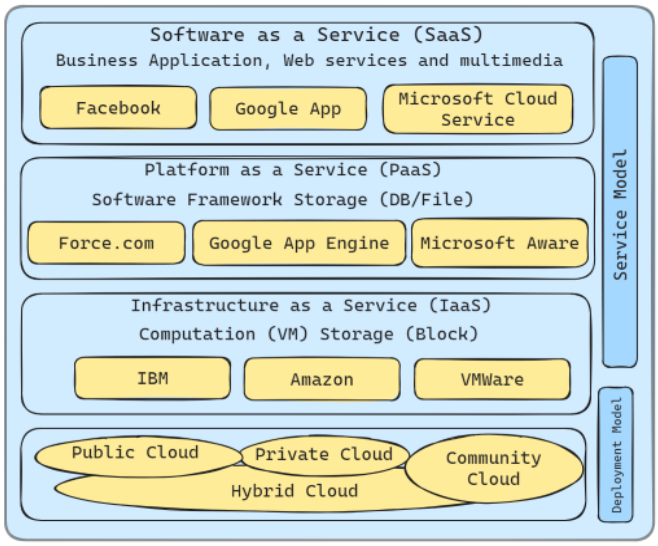

Fig. 1. Cloud Computing Architecture.

Central to CC is the effective management of resources, especially task scheduling across distributed and virtualized systems. Task scheduling plays a vital role in optimizing resource utilization, minimizing execution time, and enhancing overall system performance. However, the complexity and

*Corresponding author.

constantly changing conditions in cloud environments present major challenges for conventional scheduling methods. CC operates by allocating computational tasks to VMs within a resource pool to meet user demands and minimizing task completion time. The scheduling process generally has two primary stages: first, allocating user-submitted tasks to a selection of accessible VMs; second, establishing or relocating virtual machines onto suitable hosts [5]. The efficiency of task-to-VM scheduling directly impacts the overall processing capability of a cloud computing system, making it a critical area of study. Among the various performance metrics, makespan, which represents the total time required to complete all tasks, is one of the most important indicators for evaluating system efficiency [6, 7]. Therefore, an efficient task scheduling algorithm should aim to minimize makespan. However, as the number of tasks and VMs grows, the problem becomes an NP-hard optimization challenge, often leading to dimensional complexity issues.

Due to the practical applications and challenges associated with executing large-scale tasks, task scheduling for massive-scale applications has become a major research focus in cloud computing, gaining considerable scholarly attention. Various metaheuristic algorithms have been employed to address task scheduling and other optimization problems, often producing promising results for small-scale tasks. Nonetheless, when the problem size and variable count escalate, the quality of candidate solutions produced by these algorithms deteriorates drastically. Furthermore, numerous existing techniques fail to satisfy diverse Quality of Service (QoS) requirements, which are essential for both scientific and industrial applications. Recent research has employed metaheuristic techniques, including grey wolf optimization (GWO) [8], synergistic swarm optimization (SSO) [9], particle swarm optimization (PSO) [10], symbiotic organism search (SOS) [11], and African Vultures optimization Algorithm (AVOA) [12] to address task scheduling challenges. Despite their effectiveness in improving performance and constraining the solution search space, these methods still struggle with challenges such as excessive computational time, premature convergence to local optima, poor balance between exploration and exploitation, and unreliable performance across larger search spaces [13,14].

Owing to the constraints of existing scheduling techniques in large-scale dynamic cloud environments, this research proposes a DMO-based task scheduling algorithm. To the best of our knowledge, this is the first application of the DMO algorithm for optimizing execution time (makespan) in cloud computing [15]. By harnessing DMO's strong balance between exploration and exploitation, motivated by the foraging and social habits of dwarf mongooses, the proposed approach aims to enhance cloud system performance, reduce makespan, and improve scheduling efficiency for large-scale applications. This novel application of DMO addresses the critical need for high-performance, scalable scheduling solutions in modern cloud environments.

The study focuses on the design, development, and performance evaluation of an innovative task scheduling algorithm tailored for cloud computing environments. The effectiveness of the proposed method is benchmarked against various state-of-the-art algorithms using recognized task scheduling scenarios. Through extensive experimentation and comparative analysis, the algorithm's capability in scalability, convergence speed, and solution quality is clearly shown. The proposed task scheduling method, founded on the DMO algorithm, delivers an efficient and robust solution for task scheduling in cloud computing environments. In large-scale and dynamic cloud environments, it enhances overall system efficacy and resource utilization. This study's primary contributions can be succinctly presented as follows:

- Clearer Presentation of DMO Procedures: We provide a structured and detailed explanation of the DMO algorithm, enhancing its interpretability for cloud-based optimization problems.
- Design and Implementation of a Discrete DMO Variant: A discrete version of the DMO algorithm is proposed and implemented specifically for task scheduling in cloud computing, enabling it to handle non-continuous scheduling spaces effectively.
- Experimental Validation on GoCJ Dataset: The proposed algorithm undergoes extensive testing on the widely known GoCJ benchmark dataset, showcasing its efficacy and computing efficiency in realistic circumstances.
- Comprehensive Performance Evaluation: To evaluate performance, the algorithm is analyzed through essential metrics like makespan, response time, and degree of imbalance among VMs, demonstrating its impact on efficient resource utilization and balanced workload distribution.

The rest of the study is organized as follows: Section II discusses related studies and analyzes existing solutions for task scheduling in cloud computing. Section III defines the optimization model and problem statement adopted in this work. Section IV explains the design and implementation process of the proposed DMO-based algorithm. Section V presents a comprehensive performance evaluation based on simulation results and comparative metrics. Section VI concludes the study with a summary of outcomes and possible future research directions.

II. Related Work

Researchers have formulated diverse scheduling models from various directions to tackle the task scheduling issue in CC environments, as proven in the available literature. These models are generally improved by sophisticated algorithms, yielding favorable outcomes in reducing performance indicators such as average response times, makespan, total execution time, and enhancing load balancing. While numerous approaches have been proposed, this section focuses on summarizing the most recent and commonly used task scheduling algorithms documented in the literature [16]. Despite the progress made, there remains significant room for improvement, particularly in enhancing scalability, solution quality, and execution efficiency in large-scale, dynamic cloud environments.

The research [17] presents a unique methodology termed MALO, which is a hybrid antlion optimization algorithm

integrated with elite-based differential evolution. This strategy is explicitly formulated to address the multi-objective characteristics of task scheduling by concentrating on minimizing makespan and optimizing resource consumption concurrently. The innovation lies in improving the antlion optimization algorithm with a local search technique to better explore potential solutions and avoid suboptimal results. Through experiments using synthetic and real trace datasets with the CloudSim toolkit, MALO demonstrated superior performance compared to other well-known optimization algorithms. It was especially recognized for its accelerated convergence in extensive search areas, rendering it exceptionally appropriate for tackling large-scale scheduling challenges. The efficacy of MALO was further substantiated through statistical analysis, affirming its substantial enhancement in task scheduling results.

Task scheduling in CC is essential for the optimum utilization of resources. It entails organizing multiple tasks among diverse VMs to reduce the overall completion time (makespan) and optimize resource use. This method is intricate, particularly for large-scale data applications, as it constitutes an NP-hard problem, indicating that identifying the optimal approach is exceedingly difficult. In [18], the authors propose a method employing a hybrid dragonfly algorithm (MHDA) designed to optimize task scheduling by minimizing makespan and augmenting resource consumption. The dragonfly algorithm mimics the swarming behavior of dragonflies and is combined with b-hill climbing, a technique that helps in exploring solutions more effectively to avoid getting stuck in less optimal solutions.

The study [19] proposes an adaptive symbiotic organism search (ABFSOS) algorithm that aims to address this issue by adjusting its parameters to better balance exploration and exploitation, leading to faster and more effective solutions. Additionally, the study introduces an adaptive strategy for handling constraints within the scheduling process.

CC is vital for contemporary computational frameworks because of its scalability and adaptability, influencing system efficiency and resource management. The proposed optimization algorithm detailed in [9] integrates the strengths of the Jaya algorithm's capacity to exploit optimal solutions, the collaborative search strategy of SSO, and the stochastic component introduced by Levy flights, thereby offering a synergistic optimization framework.

The paper [20] proposed a novel approach integrating GWO and the genetic algorithm (GA) to optimize task scheduling in cloud computing, aiming to address shortcomings in existing methods and enhance efficiency. The proposed method overcomes issues like premature convergence and suboptimal solutions, resulting in more effective task scheduling in cloud environments, allowing for faster convergence in large scheduling problems, as evaluated using the CloudSim toolkit.

The proposed chameleon and remora search optimization (CRSOA) [21] tackles these difficulties by accounting for MIPS and network bandwidth impacts, which directly influence VM performance. The system simultaneously accounts for uncertainty factors, including task completion rate, load balance, scheduling cost, and makespan, hence improving scheduling processes. The CRSOA model develops a multi-objective optimization strategy for cloud task scheduling with a greedy methodology to replicate actual cloud computing work scheduling. Experimental findings demonstrate that CRSOA diminishes completion time, reduces makespan by 18.96%, lowers cost by 22.18%, and enhances load balancing among VMs by 20.54% relative to other metaheuristic algorithms.

The study [22] proposes a new multi-objective task scheduling algorithm named DCOHHOTS, which is an enhancement of the Harris Hawks Optimizer (HHO) through the integration of differential evolution (DE), chaotic maps, and opposition-based learning (OBL). The proposed algorithm optimizes the initial positioning of the "hawks" or search agents to improve the efficiency of finding solutions. By prioritizing tasks and efficiently assigning them to resources, the algorithm seeks to reduce makespan, lower energy consumption, and minimize execution costs.

The research [23] integrates the pollination behavior of flowers with the exploratory search abilities of grey wolves, in conjunction with crossover operators from evolutionary algorithms. To enhance the exploration capabilities of the FPA, the grey wolf optimizer is integrated. This strategy mimics the social hierarchy and hunting techniques of grey wolves, improving the algorithm's ability to avoid local optima and ensuring a thorough search of the solution space. The crossover operators from genetic algorithms are employed to facilitate the exploitation phase.

The study [24] introduces a novel task scheduling algorithm that integrates the Harris Hawk Optimization (HHO) algorithm with fuzzy logic to tackle the intricate issue of task scheduling in cloud environments. This combination seeks to optimize allocation of tasks to virtual machines, focusing on enhancing many objectives, including reducing makespan, decreasing energy usage, and minimizing expenses. The efficacy of the suggested fuzzy-HHO approach is substantiated by comprehensive simulations in the CloudSim framework, where it is juxtaposed with two prominent algorithms. The results demonstrate substantial improvements in makespan reduction (up to 47%), energy consumption reduction (up to 73%), and cost savings (up to 19%), particularly in circumstances with a high volume of activities.

Considering the aforementioned studies illustrating enhancements in minimizing the makespan of cloud computing systems, numerous present methodologies remain susceptible to local optima, particularly inside high-dimensional search spaces. The intrinsic complexity and organization of task-to-VM mappings underscore the necessity for ongoing study in task scheduling. A brief summary of the related research is shown in Table I.

TABLE I. AN OVERVIEW OF THE RELATED WORKS

Reference	Technique Used	Contributions	Objective
[17]	Antlion optimization algorithm	Introduced a hybrid antlion optimization method including elite-based differential evolution to address multi-objective task scheduling challenges in cloud computing environments.	Degree of imbalance, response time, and makespan
[18]	Dragonfly algorithm	Integrated the b-hill climbing technique as a local search mechanism to improve DA's exploitation ability and prevent it from getting trapped in local optima.	Makespan measure, response time, resource utilization, and the degree of imbalance
[19]	Adaptive symbiotic organisms search	Introduces an adaptive strategy for handling constraints within the scheduling process.	Makespan, financial cost, hypervolume, and percentage change
[25]	Improved black widow optimization	Introduces a new selection scheme to improve BWO, which resulted in an enhanced performance in cloud task scheduling.	Optimize resource allocation, ascertain the minimal waiting time, and identify least cost for scheduling duties.
[9]	Improved synergistic swarm optimization algorithm	Introduces an adaptive technique that dynamically affects the search process, directing the algorithm towards promising areas of the solution space.	Total execution time, makespan, Resource utilization
[20]	Grey wolf optimization algorithm	Incorporates GWO and combines crossover and mutation operators to enhance optimization performance.	Energy consumption, makespan, and cost
[21]	A chameleon and remora search optimization algorithm	Proposes a model that employs a multi-objective optimization technique for cloud task scheduling, utilizing a greedy methodology to emulate actual cloud computing task scheduling.	Resource utilization, energy consumption, cost, makespan, and degree of imbalance
[22]	Improved harris hawk optimizer	Optimizes the initial positioning of the "hawks" or search agents to improve the efficiency of finding solutions.	Makespan, energy consumption, execution cost, resource utilization
[26]	Enhanced marine predator algorithm	Integrates techniques from the whale optimization algorithm, with a nonlinear inertia weight coefficient and a golden sine technique.	Resource utilization, performance improvement ratio, degree of imbalance, and makespan
[23]	Flower Pollination Algorithm	Presents a proficient resource selection method to alleviate the stagnation of local solutions.	Makespan, resource utilization, degree of imbalance, and throughput
[24]	Improved Harris hawk optimization algorithm	Leverages the strengths of the HHO algorithm to explore and evaluate solutions within a vast solution space through the lens of fuzzy logic.	Makespan, energy consumption, cost, and runtime

III. PROBLEM DESCRIPTION

Upon submission of tasks by users, the Cloud Broker (CB) initially receives them and subsequently requests the Cloud Information Service (CIS) to identify the necessary services for task execution. After finding the relevant services, the CB organizes the tasks on the identified VMs. For example, a set of tasks $\{T_1, T_2, T_3, \ldots, T_n\}$ may be submitted within a given time frame. The available VMs are heterogeneous, each with different processing capabilities and memory capacities, meaning that the same task may have different execution times and costs depending on the VM it is assigned to.

After receiving tasks, the CB assigns them to available virtual machines $\{V_1, V_2, V_3, \ldots, V_m\}$. In the baseline configuration, task execution follows the First-Come, First-Serve (FCFS) policy. The main goal of this work is to enhance this scheduling process to improve VM utilization and minimize makespan. For this purpose, the Expected Time to Compute (ETC) metric is utilized as the basis for task assignment. The ETC value for a task on a VM is calculated by dividing the task's length by the VM's processing capacity in millions of instructions per second (MIPS). These values are organized in a matrix structure, where rows correspond to tasks and columns to VMs, and each entry represents the estimated execution time for a particular task-VM pair. The execution time of task j on virtual machine i is denoted as C_{ij}, where $i \in \{1,2,\ldots,m\}$ and $j \in \{1,2,\ldots,n\}$. Here, m corresponds to the total number of VMs and n to the total number of tasks.

To resolve challenges faced by current task scheduling techniques, including high computational overhead, poor scalability, and premature convergence, this work utilizes the DMO algorithm to obtain efficient task-to-VM assignments. Modeled after the cooperative and adaptive behaviors of dwarf mongoose colonies, DMO maintains an effective trade-off between exploration and exploitation. Integrating the ETC matrix into the scheduling process allows the proposed approach to minimize makespan while enhancing resource utilization across heterogeneous cloud environments.

IV. DMO FOR THE TASK SCHEDULING PROBLEM

The DMO algorithm [27] draws inspiration from the social structure and foraging patterns of dwarf mongooses. By simulating the coordinated behavior of scouts, alpha leaders, and babysitters, the algorithm maintains a robust balance between exploration and exploitation across the search space. Owing to these features, DMO is well-suited for addressing complex optimization tasks such as task scheduling in cloud computing, aiming to achieve near-optimal task-to-VM allocations under heterogeneous and dynamic conditions.

A. Overview of DMO Algorithm

DMO simulates a colony of dwarf mongooses navigating the solution space in search of food sources, which are analogous to optimal or near-optimal solutions. The algorithm operates through iterative updates of a population of candidate solutions, where exploration is guided by scout mongooses, and exploitation is controlled by the alpha mongoose. The babysitter mongoose ensures stability in the population by managing diversity and maintaining solution feasibility. Each candidate solution in the DMO model encodes a distinct task-to-VM mapping. Its fitness value is calculated using a scheduling criterion, most often the makespan, obtained from the ETC matrix. The goal is to minimize the makespan, ensuring balanced workload distribution and efficient resource usage across VMs.

The DMO algorithm begins by initializing a population matrix X of dimensions $(n \times d)$, where n denotes the population size (rows) and d represents the number of problem dimensions (columns), as shown in Eq. (1). This matrix models the mongoose population.

$$X = [x_{ij}]_{n*d}, i = 1,2,\dots,n; j = 1,2,\dots,d \quad (1)$$

In Eq. (2), the population matrix X is initialized randomly, where x_{ij}represents the position of the j^{th} dimension for the i^{th} individual. Here, n is the total population size and d is the number of problem dimensions.

$$x_{ij} = unifrnd(VarMin, VarMax, VarSize) \quad (2)$$

The function unifrnd generates random values uniformly distributed between $VarMin$ and $VarMax$, which define the lower and upper bounds of the problem, respectively. Once the initial population is generated, the fitness of each individual is evaluated. The probability associated with each individual's fitness is computed according to Eq. (3), which is then used to select the alpha female (α) based on these probabilities.

$$\alpha = \frac{fit_i}{\sum_{i=1}^{n} fit_i} \quad (3)$$

The parameter $n - bs$ denotes the number of mongooses in the alpha group, where bs corresponds to the number of babysitters. The alpha female's vocalization, referred to as $Peep$, influences the group's movement direction. Eq. (4) is used to generate a candidate food location, guiding the search process.

$$X_{i+1} = X_i + \phi * peep \quad (4)$$

Here, ϕ is a uniformly distributed random number in the range $(0,1)$. After each iteration, the sleeping mound value is calculated using Eq. (5):

$$sm_i = \frac{fit_{i+1} - fit_i}{max\{|fit_{i+1}, fit_i|\}} \quad (5)$$

The average sleeping mound value across all individuals is then obtained from Eq. (6):

$$\phi = \frac{\sum_{i=1}^{n} sm_i}{n} \quad (6)$$

Scout mongooses perform exploration by searching for new sleeping mounds and preventing revisits to previous locations. This exploratory movement is modeled by Eq. (7):

$$X_{i+1} = \begin{cases} X_i - CF * \phi * rand * [X_i - \vec{M}], & if\ \phi_{i+1} > \phi_i \\ X_i + CF * \phi * rand * [X_i - \vec{M}], & elsewhere \end{cases} \quad (7)$$

The control factor CF is defined in Eq. (8) and decreases gradually over iterations to balance exploration and exploitation.

$$CF = \left(1 - \frac{iter}{Max_{iter}}\right)^{\left(\frac{2*iter}{Max_{iter}}\right)} \quad (8)$$

Finally, Eq. (9) defines the movement vector $\vec{M}$, which directs the mongoose population toward new sleeping mound positions.

$$\vec{M} = \sum_{i=1}^{n} \frac{x_i \times sm_i}{x_i} \quad (9)$$

Algorithm 1 presents the pseudocode representation of the DMO algorithm, detailing its procedural logic and computational operations that can be directly applied for task scheduling in cloud computing environments.

Algorithm 1: Pseudocode for DMO

Input: Task set, VM set, population size n, maximum iterations Max_iter, control parameters

Output: Optimal task-to-VM mapping (minimum makespan)

1. Initialize all control parameters.
2. Generate the initial mongoose population X=[x_ij]_(n×d) using Eq. (2).
3.Divide the population into three functional groups: scouts, babysitters, and alphas.
4. Determine the number of active search agents by excluding babysitters from the total population.
5. Assign the babysitter exchange rate L.
6. Repeat until the stopping condition or Max_iter is reached:
   a. Evaluate the fitness of each mongoose using the scheduling objective (makespan).
   b. Identify the alpha group based on selection probability using Eq. (3).
   c. Generate a new candidate food source location using Eq. (4).
   d. Compute the sleeping mound value for each mongoose using Eq. (5).
   e. Calculate the average sleeping mound value and movement vector M ⃗ using Eq. (6) and (9).
   f. Update scout mongoose positions using Eq. (7) based on exploration conditions.
   g. Perform babysitter exchange according to rate L.
   h. Update the best solution found so far if improvement occurs.

Return the final best solution as the optimal task scheduling configuration.

V. Experimental Evaluation

The efficiency and scalability of the proposed task scheduling technique based on the DMO algorithm are evaluated and compared against benchmark techniques reported in the literature. As outlined earlier, this study primarily focuses on assessing the scheduling performance of the proposed technique. Accordingly, all tasks considered in the experimental evaluation are assumed to be pre-decomposed and mutually independent. The experimental configuration, performance metrics, and evaluation procedures are described in detail in the following subsections.

A. Experimental Settings

The benchmarking process involved comparing the proposed DMO algorithm with several state-of-the-art metaheuristic approaches, including MALPSO [28], EMPA [26], and GWOEM [29]. Each algorithm was executed independently 20 times to reduce the influence of stochastic variations and to ensure the statistical reliability of the obtained results. All experiments were performed using the CloudSim simulation toolkit, which enables the modeling and analysis of cloud computing environments. The simulations were

conducted on a system equipped with a 12th Gen Intel(R) Core (TM) i7-12700H processor operating at 2.70 GHz, 16 GB of RAM, and running the Windows 11 operating system.

TABLE II. PARAMETER IMPLEMENTATION

Algorithm	Parameter	Value
GBOEM	Population size	100
	a	Linearly decrease from 2 to 0
MALPSO	Population size	100
	c_1, c_2	$[2.1, 2.1]$
	w_{max}, w_{min}	$[0.9, 0.6]$
EMPA	Population size	100
	FADs	0.2
	P	0.5
DMO	Population size	100

The experimental parameter settings are summarized in Table II. To ensure consistency, all algorithms were executed with a population size of 100 and a maximum of 100 iterations, following configurations commonly adopted in recent studies. Although increasing the number of iterations can improve solution quality in terms of makespan, it also results in higher computational overhead.

B. Dataset Description

This study employs the Google Cloud Jobs (GoCJ) dataset to establish the task workload in order to more accurately simulate the task scheduling scenario in cloud computing. In this dataset, the computational capability of each VM is expressed in millions of instructions per second (MIPS), while task sizes are characterized by the number of instructions required for completion. The GoCJ repository comprises multiple datasets, each stored in a text file format. Every dataset contains a specific number of tasks, as summarized in Table III. Each text file comprises a sequence of lines, each of which denotes a task and contains a single value representing the task's magnitude in MI.

C. Experimental Results and Analysis

This subsection presents the experimental findings derived from the proposed DMO-based task scheduling method and compares them with the results of three established algorithms: MALPSO, EMPA, and GWOEM. A comprehensive analysis of the comparative performance is also included. The simulations are conducted under two separate circumstances to ensure a thorough evaluation. A constant quantity of VMs is utilized across all datasets. In the second scenario, the quantity of VMs is altered for each dataset to evaluate the algorithm's adaptability and scalability under varying resource constraints.

TABLE III. THE NUMBER OF TASKS CONTAINED IN EACH DATASET CATEGORY

Dataset category	Number of tasks
GoCJ_100	100
GoCJ_200	200
GoCJ_300	300
GoCJ_400	400
GoCJ_500	500
GoCJ_600	600
GoCJ_700	700
GoCJ_800	800
GoCJ_900	900
GoCJ_1000	1000

1) Performance comparison with a constant number of VMs: A fixed number of VMs is used in the experiment for every dataset. The assessment is centered on a diverse array of activities extracted from the aforementioned GoCJ datasets, which encompass tasks of varied magnitudes. The goal of optimization is to effectively allocate these jobs to 50 VMs. Performance for each algorithm under comparison, including the proposed DMO-based technique, is evaluated using the makespan metric to ascertain the overall time necessary to accomplish all assigned jobs.

Table IV presents the makespan results for four task scheduling algorithms evaluated across different task sizes using a fixed set of 50 virtual machines. The DMO-based method is compared against MALPSO, GWOEM, and MPA. The evaluation includes the Best, Worst, and Mean values obtained from 20 independent runs for each algorithm. From the results, it is evident that DMO consistently achieves lower Mean values across nearly all task sizes. Specifically, DMO obtains the lowest Mean makespan in all cases, ranging from 228.2 (for 200 tasks) to 727.96 (for 1000 tasks), outperforming the competing algorithms.

To provide a more intuitive comparison, the cumulative ranks based on the Mean makespan values across 10 datasets are illustrated in Fig. 2. For each dataset, the algorithms are ranked from best to worst in terms of average performance, where a lower bar height indicates better overall performance. The top-performing algorithm in each dataset receives a rank of 1, and the ranks are summed to produce the cumulative values shown in the figure. As depicted in Fig. 2, the proposed DMO-based scheduling algorithm consistently outperforms the competing methods, achieving the lowest cumulative rank of 10, which indicates it ranked first in all datasets. While MALPSO, GWOEM, and MPA accumulate ranks of 19, 28, and 37, respectively, reflecting their lower relative performance. These results further validate the effectiveness of DMO in solving the cloud task scheduling problem.

TABLE IV. MAKESPAN ACROSS ALL DATASETS WITH 50 FIXED VMS

Task	MALPSO			GWOEM			MPA			DMO		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
100	180.1	180.21	180.1	180.1	180.1	180.1	180.1	180.21	180.1	180.1	180.11	180.1
200	215.4	255.57	238.99	219.7	251.1	240.46	225.55	269.7	250.1	216.5	237.2	228.69
300	282.14	343.52	321.35	296.3	344.9	324	303.1	361.6	334.63	281.9	314.7	298.32
400	333.1	388.3	369.44	343.4	394.9	370.31	346.1	407.1	377.27	311.3	359.84	339.78
500	405.55	468.52	441.26	384.93	462.3	443.77	421.9	474.72	451.65	378.4	426.1	408.05
600	488.4	582.8	545.94	516.8	576.86	549.28	514.85	593.12	558.57	484.95	532.91	509.06
700	505.92	594.41	556.27	526.6	591.31	560.39	515	615.8	570.15	492.6	534.4	518.05
800	566.05	644.5	611.21	569.97	651.3	618.65	588.01	662.5	628.62	538.1	590.94	573.81
900	656.11	764.6	727.52	670.31	768.3	734.85	695.6	789.9	741.67	630.7	701.26	680.6
1000	727.61	832.7	776.72	737.31	830.4	785.59	748.6	831.71	795.9	694.46	749.71	729.25

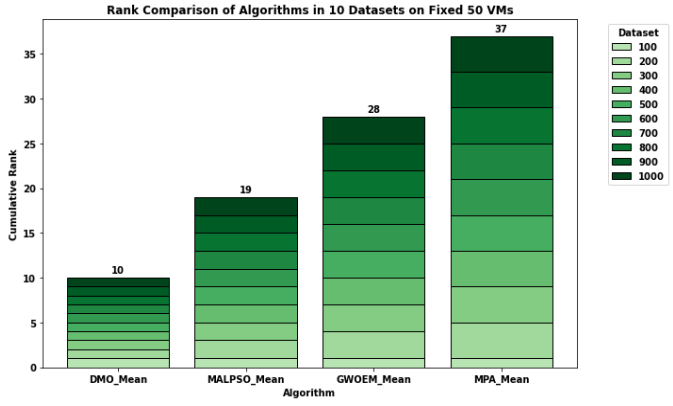

Fig. 2. Mean rank comparison across datasets with fixed VMs.

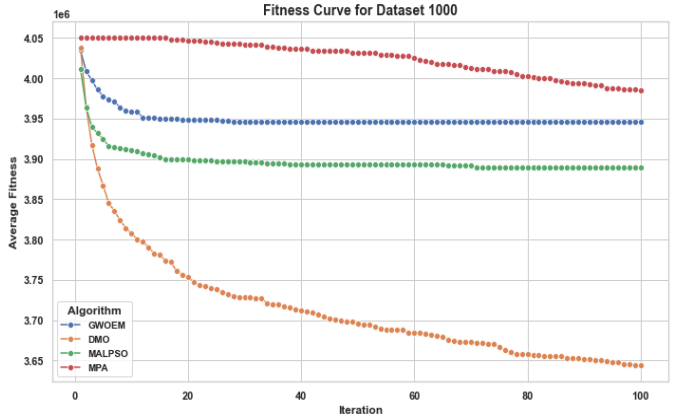

Fig. 3. Fitness convergence curves of algorithms on fixed number of VMs.

Fig. 3 presents the fitness convergence curve of the proposed DMO algorithm in comparison with GWOEM, MALPSO, and MPA for the 1000-task dataset. As observed from the curve, DMO achieves a significantly lower average makespan than the competing algorithms. Although MALPSO converges faster in the early iterations, it plateaus earlier and stabilizes at a higher makespan compared to DMO. In contrast, DMO continues refining its solution steadily throughout the search process, reflecting a balanced exploration-exploitation dynamic that avoids premature convergence. GWOEM and MPA show comparatively slower convergence and higher final makespan values, indicating their limited ability to escape local optima or adapt efficiently under heavier task loads. DMO, however, demonstrates strong robustness and convergence stability, consistently maintaining a downward trend in fitness even toward later iterations.

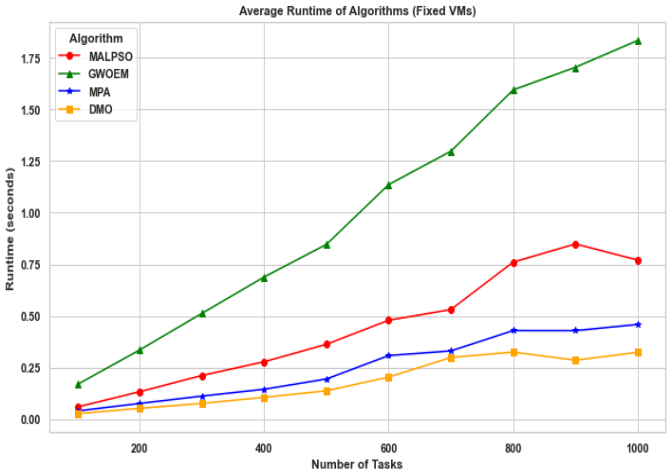

Fig. 4. Average runtime per dataset with fixed VMs.

Fig. 4 displays the average runtime of each algorithm: DMO, MALPSO, GWOEM, and MPA across increasing task sizes under a fixed number of virtual machines. The x-axis represents the number of tasks (ranging from 100 to 1000), while the y-axis indicates the average computational time in seconds required to complete one run. As shown in the figure, the DMO algorithm consistently demonstrates the lowest runtime across all task sizes, indicating its superior computational efficiency. The consistently shorter execution time of DMO, coupled with its high-quality results observed in earlier evaluations, shows its practical suitability for large-scale task scheduling applications particularly in real-time or resource-constrained cloud environments.

A paired t-test was conducted to statistically assess the performance of the proposed DMO algorithm against MALPSO, GWOEM, and MPA on the GoCJ dataset under fixed VM configurations as shown in Table V, using a

significance level of $\alpha = 0.05$. For small-scale workloads (100 tasks), no statistically significant differences were observed ($p > 0.05$), indicating comparable performance due to limited scheduling complexity. However, for task sizes of 200 and above, DMO consistently achieved significantly lower makespan values, as reflected by negative mean differences, large-magnitude t-values, and extremely small p-values (often $< 10^{-29}$), leading to rejection of the null hypothesis. The magnitude and consistency of these improvements increase with problem size, particularly for medium to large workloads (300–1000 tasks), demonstrating DMO's robustness and scalability in high-dimensional scheduling spaces. These statistically significant results confirm that the observed performance gains are not due to random variation and provide strong empirical evidence supporting the effectiveness of DMO for large-scale cloud task scheduling.

TABLE V. STATISTICAL T-TEST RESULTS FOR MAKESPAN COMPARISON ON THE GOCJ DATASET WITH FIXED VMS

Task Size	Comparison	Mean Difference	Std Difference	t-value	p-value	Better Algorithm	Significant
100	DMO vs MALPSO	-0.001	0.011055	-0.90453	0.367908227	DMO	FALSE
100	DMO vs GWOEM	0.0001	0.001	1	0.319748474	GWOEM	FALSE
100	DMO vs MPA	-0.000998	0.011035	-0.90435	0.368003029	DMO	FALSE
200	DMO vs MALPSO	-10.296463	8.083912	-12.737	1.40E-22	DMO	TRUE
200	DMO vs GWOEM	-11.771234	7.242589	-16.2528	1.07E-29	DMO	TRUE
200	DMO vs MPA	-21.408265	9.20326	-23.2616	6.51E-42	DMO	TRUE
300	DMO vs MALPSO	-23.035018	12.02804	-19.1511	4.61E-35	DMO	TRUE
300	DMO vs GWOEM	-25.681148	11.8629	-21.6483	2.52E-39	DMO	TRUE
300	DMO vs MPA	-36.306611	15.15638	-23.9547	5.49E-43	DMO	TRUE
400	DMO vs MALPSO	-29.663394	12.11074	-24.4935	8.32E-44	DMO	TRUE
400	DMO vs GWOEM	-30.523941	13.7605	-22.1823	3.39E-40	DMO	TRUE
400	DMO vs MPA	-37.484893	14.85506	-25.2338	6.53E-45	DMO	TRUE
500	DMO vs MALPSO	-33.212818	14.36265	-23.1244	1.07E-41	DMO	TRUE
500	DMO vs GWOEM	-35.719619	13.81348	-25.8585	7.95E-46	DMO	TRUE
500	DMO vs MPA	-43.595315	15.41503	-28.281	3.20E-49	DMO	TRUE
600	DMO vs MALPSO	-36.88152	19.63754	-18.7811	2.10E-34	DMO	TRUE
600	DMO vs GWOEM	-40.214484	17.52061	-22.9527	1.99E-41	DMO	TRUE
600	DMO vs MPA	-49.507718	18.89599	-26.2001	2.55E-46	DMO	TRUE
700	DMO vs MALPSO	-38.222648	18.76097	-20.3735	3.44E-37	DMO	TRUE
700	DMO vs GWOEM	-42.340123	18.12233	-23.3635	4.51E-42	DMO	TRUE
700	DMO vs MPA	-52.092789	19.99794	-26.0491	4.21E-46	DMO	TRUE
800	DMO vs MALPSO	-37.393152	21.04933	-17.7645	1.49E-32	DMO	TRUE
800	DMO vs GWOEM	-44.840048	19.5918	-22.8871	2.53E-41	DMO	TRUE
800	DMO vs MPA	-54.803758	19.53037	-28.0608	6.37E-49	DMO	TRUE
900	DMO vs MALPSO	-46.924606	24.33758	-19.2807	2.72E-35	DMO	TRUE
900	DMO vs GWOEM	-54.256754	21.77357	-24.9186	1.92E-44	DMO	TRUE
900	DMO vs MPA	-61.076888	21.59429	-28.2838	3.17E-49	DMO	TRUE
1000	DMO vs MALPSO	-47.467969	27.49125	-17.2666	1.26E-31	DMO	TRUE
1000	DMO vs GWOEM	-56.336122	19.23301	-29.2914	1.43E-50	DMO	TRUE
1000	DMO vs MPA	-66.645948	22.46125	-29.6715	4.54E-51	DMO	TRUE

Fig. 5 illustrates the average makespan values obtained by four algorithms as the number of tasks increases from 100 to 1000. As depicted, the makespan values increase proportionally with the number of tasks which is expected due to the growing computational workload. However, the DMO algorithm consistently achieves the lowest makespan across all dataset sizes, demonstrating its superior scheduling capability. Notably, the gap between DMO and the other methods becomes more prominent as task volume increases, highlighting DMO's scalability and robustness under larger workloads.

Fig. 6 presents the degree of imbalance of the task scheduling solutions generated by the four compared algorithms under a fixed number of VMs. DoI reflects how evenly tasks are distributed across available resources, with lower values indicating better load balancing. As illustrated, DMO consistently achieves the lowest imbalance across all task volumes, demonstrating its strong ability to maintain load uniformity among VMs. While all algorithms show a downward trend in DoI as the task count grows, indicating improved balance with larger workloads DMO outperforms others in both convergence speed and final balance quality.

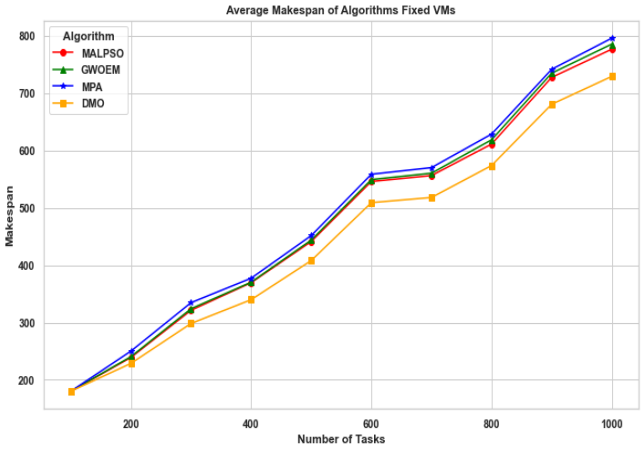

Fig. 5. Average makespan per dataset with fixed VMs.

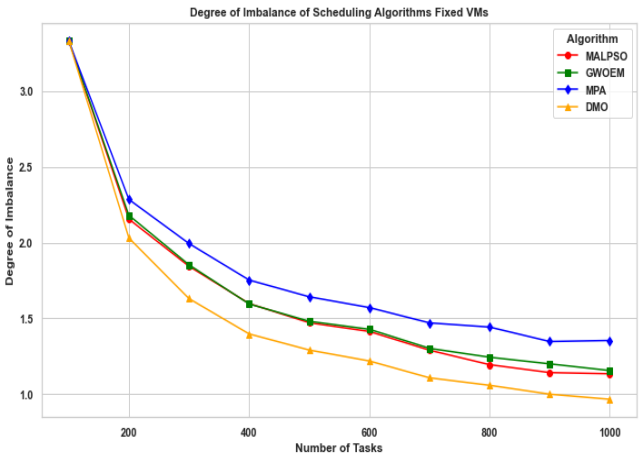

Fig. 6. Degree of imbalance per dataset with fixed VMs.

Fig. 7. Average throughput per dataset with fixed VMs.

Fig. 7 illustrates the average throughput achieved by each of the four algorithms across increasing numbers of tasks under a fixed number of VMs. Throughput measures the number of tasks successfully completed per unit time and is a critical metric for evaluating the responsiveness and efficiency of scheduling algorithms in cloud environments. From the figure it is observed that MALPSO, GWOEM, and MPA exhibit similar patterns, with slightly lower throughput values. While all algorithms show improved throughput with increased task sizes due to higher resource utilization DMO maintains a clear performance lead, particularly beyond 400 tasks, where its optimization capabilities yield a noticeable efficiency advantage.

2) Scenario with varying number of VMs: To further assess the scalability and performance of the proposed algorithm under varying resource conditions, a second set of experiments was conducted using different numbers of VMs for each dataset category. The allocation of VMs for each dataset was determined to be 10% of the total number of tasks, yielding VM counts between 10 and 100. The number of VMs designated for each dataset is summarized in Table VI. This experimental configuration aims to evaluate how changes in resource availability impact key performance indicators, particularly makespan. By correlating scheduling efficiency with increasing task-to-resource ratios, the results provide insight into the scalability and robustness of the algorithms under more realistic and dynamic cloud environments.

Table VII presents the makespan results of four algorithms: DMO, MALPSO, GWOEM, and MPA. The evaluation metrics include the Best, Worst, and Mean of the makespan over 20 independent runs. From the data in Table VII, it is evident that the proposed DMO consistently achieves the best Mean makespan values across all datasets. For instance, in the largest dataset (1000 tasks), DMO yields a Mean makespan of 468.68, which is significantly lower than the corresponding results from GWOEM (505.52), MPA (514.72), and MALPSO (503.82). DMO also performs strongly in the Best-case results, outperforming the other algorithms in most datasets, reflecting its superior ability to locate high-quality solutions.

TABLE VI. VM DISTRIBUTION PER DATASET SIZE

Dataset size	Number of VMs
100	10
200	20
300	30
400	40
500	50
600	60
700	70
800	80
900	90
1000	100

TABLE VII. MAKESPAN ACROSS ALL DATASETS WITH VARIABLE VM COUNTS

Task	MALPSO			GWOEM			MPA			DMO		
	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean
100	305.1	347.56	327.91	303.3	361.1	335.61	314.2	379	351.35	294.2	323.1	311.75
200	354.9	414.9	389.57	370.93	422.6	396.29	361.31	449.2	410.52	342.21	382.87	364.22
300	381.7	465.1	432.9	388.5	465.4	434.7	423.3	492.6	451.1	372.72	422.12	403.97
400	383.7	446.9	419.35	396	446.51	424.54	400.33	456.4	432.59	374.2	405.61	390.58
500	393.77	462.24	438.06	402.3	465.3	440.23	413.42	479	450.84	374.1	428.14	408.18
600	436.13	525.21	488.6	443.64	515.3	490.46	462.68	530.94	501.29	420.44	476.1	455.19
700	404.71	475.67	451.87	430.74	480.34	456.76	432.43	491.9	462.34	394.02	435.06	417.94
800	413.32	482.84	458.24	413.12	493.11	461.07	427.71	499.3	470.37	404.01	442.67	426.23
900	464.9	527.85	499.9	465.44	533.42	503.5	474.53	546.8	510.91	443.21	486.13	468.56
1000	460.74	528.95	503.82	465.21	530.93	505.5	467.96	543.04	514.7	441.21	491.2	468.68

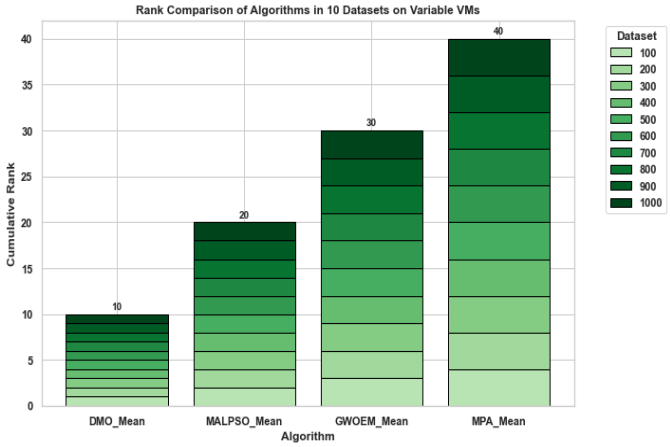

Fig. 8. Mean rank comparison across datasets with variable VMs.

To better visualize the comparative performance, Fig. 8 ranks each algorithm based on its Best, Worst, and Mean. The proposed DMO achieves the lowest cumulative rank in three of the four subplots, especially excelling in the Best and Mean categories. This confirms DMO's capability to consistently find better solutions and maintain overall stability. Even in the Worst and Std categories, DMO ranks among the top performers, further supporting its robustness.

Fig. 9 shows the fitness convergence of DMO, GWOEM, MALPSO, and MPA on the 1000-task dataset using a variable number of VMs. DMO exhibits the fastest and most stable convergence, achieving the lowest average fitness. While other algorithms plateau early, DMO continues improving, showing strong exploitation capability. These results confirm DMO's efficiency and robustness in high-load, dynamic VM environments.

To further examine the robustness of the proposed DMO algorithm, a paired t-test was performed on the makespan results obtained from the GoCJ dataset with varying numbers of VMs as shown in Table VIII. Unlike the fixed-VM scenario, all comparisons across task sizes from 100 to 1000 exhibit statistically significant differences ($p < 0.05$), with DMO consistently identified as the superior algorithm. The uniformly negative mean differences and large-magnitude t-values indicate substantial and reliable reductions in makespan when DMO is applied. Notably, the statistical significance persists even for small-scale workloads (100 tasks), highlighting DMO's strong adaptability to dynamic resource availability. As task sizes increase, the magnitude of the mean differences remains consistently high, demonstrating that DMO effectively maintains solution quality despite increased scheduling complexity and VM variability. These results confirm that DMO is not only scalable but also resilient to changes in cloud infrastructure, making it particularly suitable for real-world cloud environments characterized by fluctuating resource configurations.

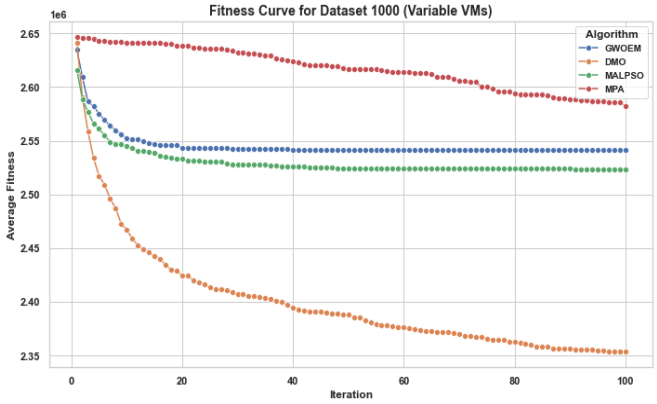

Fig. 9. Fitness convergence curves of algorithms on variable number of VMs.

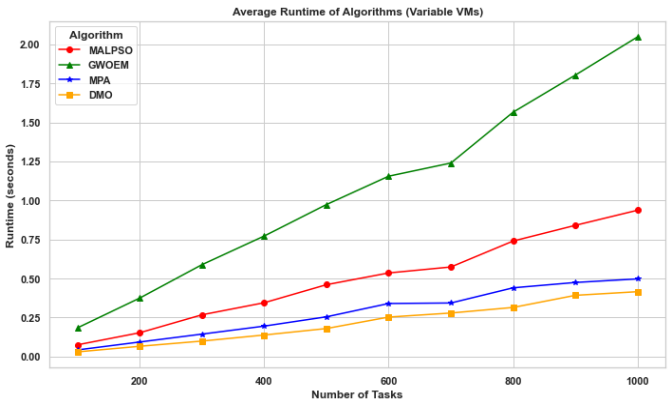

Fig. 10. Average runtime per dataset with variable VMs.

TABLE VIII. STATISTICAL T-TEST RESULTS FOR MAKESPAN COMPARISON ON THE GOCJ DATASET WITH VARYING VMS

Task Size	Comparison	Mean Difference	Std Difference	t-value	p-value	Better Algorithm	Significant
100	DMO vs MALPSO	-16.1569	11.04326	-14.6306	1.72E-26	DMO	TRUE
100	DMO vs GWOEM	-23.8587	11.50375	-20.7399	8.20E-38	DMO	TRUE
100	DMO vs MPA	-39.5934	15.05279	-26.303	1.82E-46	DMO	TRUE
200	DMO vs MALPSO	-25.3435	16.30967	-15.539	2.65E-28	DMO	TRUE
200	DMO vs GWOEM	-32.0692	13.95662	-22.9778	1.82E-41	DMO	TRUE
200	DMO vs MPA	-46.2917	18.73325	-24.711	3.91E-44	DMO	TRUE
300	DMO vs MALPSO	-28.9261	17.46182	-16.5654	2.69E-30	DMO	TRUE
300	DMO vs GWOEM	-30.7284	17.15124	-17.9161	7.83E-33	DMO	TRUE
300	DMO vs MPA	-47.1226	16.66213	-28.2812	3.20E-49	DMO	TRUE
400	DMO vs MALPSO	-28.7707	13.0834	-21.9902	6.95E-40	DMO	TRUE
400	DMO vs GWOEM	-33.9668	13.17658	-25.7781	1.04E-45	DMO	TRUE
400	DMO vs MPA	-42.0102	12.98203	-32.3603	1.88E-54	DMO	TRUE
500	DMO vs MALPSO	-29.8839	16.69724	-17.8975	8.47E-33	DMO	TRUE
500	DMO vs GWOEM	-32.0542	13.75281	-23.3074	5.52E-42	DMO	TRUE
500	DMO vs MPA	-42.6606	14.48119	-29.4593	8.60E-51	DMO	TRUE
600	DMO vs MALPSO	-33.4175	18.49725	-18.0662	4.15E-33	DMO	TRUE
600	DMO vs GWOEM	-35.2753	14.874	-23.7161	1.28E-42	DMO	TRUE
600	DMO vs MPA	-46.1056	19.00228	-24.2632	1.86E-43	DMO	TRUE
700	DMO vs MALPSO	-33.9315	16.89139	-20.088	1.06E-36	DMO	TRUE
700	DMO vs GWOEM	-38.8176	14.08342	-27.5626	3.08E-48	DMO	TRUE
700	DMO vs MPA	-44.3964	17.91821	-24.7772	3.12E-44	DMO	TRUE
800	DMO vs MALPSO	-32.0093	15.58234	-20.542	1.78E-37	DMO	TRUE
800	DMO vs GWOEM	-34.8438	15.05719	-23.141	1.01E-41	DMO	TRUE
800	DMO vs MPA	-44.1438	14.65385	-30.1243	1.18E-51	DMO	TRUE
900	DMO vs MALPSO	-31.3355	18.51359	-16.9257	5.53E-31	DMO	TRUE
900	DMO vs GWOEM	-34.9396	17.65528	-19.7899	3.48E-36	DMO	TRUE
900	DMO vs MPA	-42.3484	16.74164	-25.2952	5.30E-45	DMO	TRUE
1000	DMO vs MALPSO	-35.1394	17.35888	-20.2429	5.75E-37	DMO	TRUE
1000	DMO vs GWOEM	-36.8164	15.39813	-23.9096	6.44E-43	DMO	TRUE
1000	DMO vs MPA	-46.0187	18.92846	-24.3119	1.57E-43	DMO	TRUE

Fig. 10 compares the runtime of the four algorithms across varying task sizes using a variable number of VMs. The proposed DMO maintains the lowest runtime throughout, showing excellent computational efficiency. GWOEM incurs the highest cost, scaling poorly as task size increases. This confirms DMO's lightweight nature and scalability, making it suitable for real-time cloud scheduling. Even with increased workload, DMO's runtime growth remains modest, reflecting its optimized search process. This efficiency makes it ideal for deployment in latency-sensitive environments.

Fig. 11 presents the average makespan performance of DMO, MALPSO, GWOEM, and MPA as the number of tasks increases under a variable VM setup. The DMO achieves the lowest makespan value across all task sizes, indicating superior task-to-resource mapping. While the makespan naturally increases with task load, DMO maintains a significant margin of improvement over competing algorithms. This trend reflects DMO's efficiency and scalability, especially in dynamically resourced environments. The stability in DMO's curve also suggests consistent performance across heterogeneous workloads.

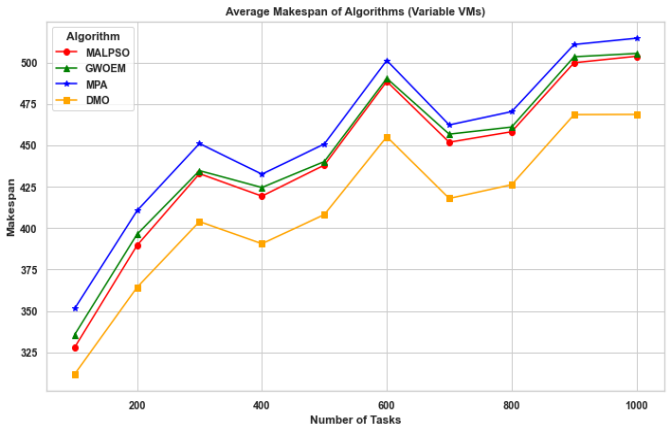

Fig. 11. Average makespan per dataset with variable VMs.

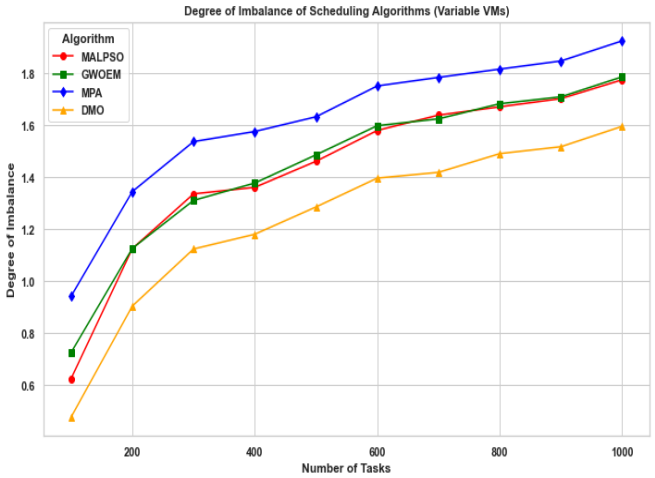

Fig. 12. Degree of imbalance per dataset with variable VMs.

Fig. 12 illustrates the degree of imbalance for four algorithms as the task count increases under variable VM settings. The DMO maintains the lowest imbalance values, indicating more uniform load distribution. Although DoI increases with task volume, DMO's curve rises more slowly, showing greater stability and fairness. In contrast, MPA consistently shows the highest imbalance, highlighting less efficient task scheduling. These results confirm DMO's robustness in balancing workloads, even in dynamic VM environments.

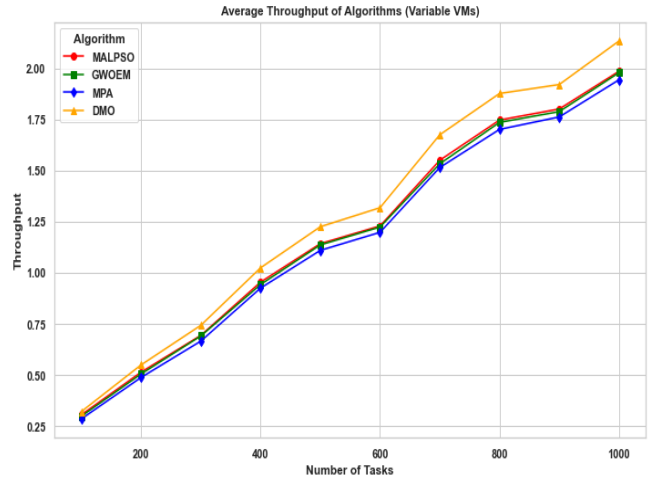

Fig. 13. Average throughput per dataset with variable VMs.

Fig. 13 depicts the average throughput achieved by each algorithm as the number of tasks increases under a variable number of VMs. The proposed DMO consistently outperforms its counterparts, achieving the highest throughput across all task sizes. This indicates more tasks are completed per time unit, highlighting DMO's superior efficiency. The gap becomes more significant as task volume scales, confirming DMO's strong adaptability and parallelism handling. This makes it well-suited for high-throughput cloud environments.

VI. Conclusion and Future Work

Task scheduling is a substantial and complex challenge in CC, with ongoing research aimed at discovering efficient methods for assigning tasks to computational resources to minimize makespan. This study presents a DMO-based task scheduling technique aimed at enhancing system performance through the optimal utilization of available computing resources. The comprehensive execution of the proposed DMO methodology has been delineated, and its efficacy was validated utilizing the GoCJ benchmark datasets. Two experimental situations were executed to evaluate performance, and the outcomes were compared with those of the MALPSO, GWOEM, and MPA algorithms. For each algorithm, essential performance measures such as response time, degree of imbalance, throughput, and makespan were meticulously assessed to gauge overall scheduling efficacy and resource consumption in the cloud context. The analysis demonstrated that the DMO-based method consistently produced superior makespan results, especially under high task loads and varying VM conditions. These findings highlight DMO's capability in efficiently exploring complex solution spaces and maintaining balanced resource distribution. Therefore, the proposed approach significantly enhances task scheduling performance and contributes to improving the overall efficiency and responsiveness of cloud computing systems.

Despite the promising results obtained, this study has several limitations that should be acknowledged to provide a clear and accurate interpretation of the findings. First, the experimental evaluation was conducted using the GoCJ benchmark dataset, which although widely adopted may not fully capture all characteristics of real-world cloud workloads such as bursty task arrivals, task dependencies, or strict deadline constraints. Second, the performance assessment focused primarily on makespan minimization, while other important quality of service metrics such as energy consumption, monetary cost, load balancing, and service level agreement violations were not jointly optimized or analyzed. Third, the experiments were performed under simulated cloud environments with fixed and varying VM configurations and therefore the observed performance may differ in large scale production clouds where network latency, VM failures, and dynamic resource provisioning occur. In addition, the proposed DMO algorithm was compared with a limited set of metaheuristic baselines and comparisons with a broader range of recent scheduling approaches could further strengthen the generality of the conclusions. Finally, algorithm parameter tuning was carried out empirically and alternative tuning strategies may lead to different performance outcomes. Addressing these limitations in future studies through real cloud experimentation, multi objective optimization, and expanded comparative analysis would further enhance the applicability and robustness of the proposed approach.

As future work, we plan to develop a multi-objective performance model that incorporates additional quality service metrics to further enhance task scheduling in cloud computing environments. In particular, optimizing load balancing will be a key focus, as it plays a critical role in maintaining system stability and performance. We also intend to extend the DMO algorithm to support more complex task structures, including

scientific workflows and cloud-based deep learning workloads, thereby improving its adaptability to diverse application scenarios. Furthermore, the applicability of the proposed DMO methodology will be explored in other optimization domains, including workflow scheduling, underwater wireless sensor networks, feature selection, and Internet of Things scheduling problems, to evaluate its generalization capability.

REFERENCES

[1] S. Tang, Y. Yu, H. Wang, G. Wang, W. Chen, Z. Xu, S. Guo, W. Gao, A Survey on Scheduling Techniques in Computing and Network Convergence, IEEE Communications Surveys and Tutorials 26 (2024) 160–195. https://doi.org/10.1109/COMST.2023.3329027.

[2] O.L. Abraham, M.A. Ngadi, J.M. Sharif, M.K.M. Sidik, Task Scheduling in Cloud Environment-Techniques, Applications, and Tools: A Systematic Literature Review, IEEE Access (2024). https://doi.org/10.1109/ACCESS.2024.3466529.

[3] Z. Ahmad, A.I. Jehangiri, M.A. Ala'anzy, M. Othman, R. Latip, S.K.U. Zaman, A.I. Umar, Scientific Workflows Management and Scheduling in Cloud Computing: Taxonomy, Prospects, and Challenges, IEEE Access 9 (2021) 53491–53508. https://doi.org/10.1109/ACCESS.2021.3070785.

[4] M. Menaka, K.S. Sendhil Kumar, Workflow scheduling in cloud environment – Challenges, tools, limitations & methodologies: A review, Measurement: Sensors 24 (2022). https://doi.org/10.1016/j.measen.2022.100436.

[5] G. Lokesh, K.K. Baseer, A meta-heuristic approach-aided multi-objective strategy with optimal resource allocation via fault tolerant and priority-based scheduling for load balancing in cloud, Wireless Networks (2025). https://doi.org/10.1007/s11276-024-03885-0.

[6] O.L. Abraham, M.A. Ngadi, J.B.M. Sharif, M.K.M. Sidik, Multi-Objective Optimization Techniques in Cloud Task Scheduling: A Systematic Literature Review, IEEE Access (2025). https://doi.org/10.1109/ACCESS.2025.3529839.

[7] M.S.R. Krishna, D. Khasim Vali, ADWEH: A Dynamic Prioritized Workflow Task Scheduling Approach Based on the Enhanced Harris Hawk Optimization Algorithm, IEEE Access (2025). https://doi.org/10.1109/ACCESS.2025.3543880.

[8] M.A. Khan, R. ur Rasool, A multi-objective grey-wolf optimization based approach for scheduling on cloud platforms, J Parallel Distrib Comput 187 (2024) 104847. https://doi.org/https://doi.org/10.1016/j.jpdc.2024.104847.

[9] L. Abualigah, A.M.A. Hussein, M.H. Almomani, R.A. Zitar, H. Migdady, A.I. Alzahrani, A. Alwadain, Improved synergistic swarm optimization algorithm to optimize task scheduling problems in cloud computing, Sustainable Computing: Informatics and Systems 43 (2024). https://doi.org/10.1016/j.suscom.2024.101012.

[10] C. Lu, J. Zhu, H. Huang, Y. Sun, A multi-hierarchy particle swarm optimization-based algorithm for cloud workflow scheduling, Future Generation Computer Systems 153 (2024) 125–138. https://doi.org/https://doi.org/10.1016/j.future.2023.11.030.

[11] K. Chakraborty, G. Deb, S. Sharma, Symbiotic organisms search-based multi-objective optimal placement of distributed generators considering source and load uncertainty, SCIENTIA IRANICA 30 (2023) 518–535. https://doi.org/10.24200/sci.2021.56149.4575.

[12] R. Ghafari, N. Mansouri, E-AVOA-TS: Enhanced African vultures optimization algorithm-based task scheduling strategy for fog–cloud computing, Sustainable Computing: Informatics and Systems 40 (2023) 100918. https://doi.org/https://doi.org/10.1016/j.suscom.2023.100918.

[13] J. Liu, R. Sarker, S. Elsayed, D. Essam, N. Siswanto, Large-scale evolutionary optimization: A review and comparative study, Swarm Evol Comput 85 (2024). https://doi.org/10.1016/j.swevo.2023.101466.

[14] A. Mohammadi, F. Sheikholeslam, Intelligent optimization: Literature review and state-of-the-art algorithms (1965–2022), Eng Appl Artif Intell 126 (2023). https://doi.org/10.1016/j.engappai.2023.106959.

[15] O.L. Abraham, M.A. Ngadi, A comprehensive review of dwarf mongoose optimization algorithm with emerging trends and future research directions, Decision Analytics Journal 14 (2025). https://doi.org/10.1016/j.dajour.2025.100551.

[16] M.S.R. Krishna, A Systematic Review on Various Task Scheduling Algorithms in Cloud Computing, EAI Endorsed Transactions on Internet of Things 10 (2024). https://doi.org/10.4108/eetiot.4548.

[17] L. Abualigah, A. Diabat, A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments, Cluster Comput 24 (2021) 205–223. https://doi.org/10.1007/s10586-020-03075-5.

[18] L. Abualigah, A. Diabat, M.A. Elaziz, Intelligent workflow scheduling for Big Data applications in IoT cloud computing environments, Cluster Comput 24 (2021) 2957–2976. https://doi.org/10.1007/s10586-021-03291-7.

[19] M. Abdullahi, M.A. Ngadi, S.I. Dishing, S.M. Abdulhamid, An adaptive symbiotic organisms search for constrained task scheduling in cloud computing, J Ambient Intell Humaniz Comput 14 (2023) 8839–8850. https://doi.org/10.1007/s12652-021-03632-9.

[20] I. Behera, S. Sobhanayak, Task scheduling optimization in heterogeneous cloud computing environments: A hybrid GA-GWO approach, J Parallel Distrib Comput 183 (2024). https://doi.org/10.1016/j.jpdc.2023.104766.

[21] P. Pabitha, K. Nivitha, C. Gunavathi, B. Panjavarnam, A chameleon and remora search optimization algorithm for handling task scheduling uncertainty problem in cloud computing, Sustainable Computing: Informatics and Systems 41 (2024) 100944. https://doi.org/https://doi.org/10.1016/j.suscom.2023.100944.

[22] R. Ghafari, N. Mansouri, Improved Harris Hawks Optimizer with chaotic maps and opposition-based learning for task scheduling in cloud environment, Cluster Comput 27 (2024) 1421–1469. https://doi.org/10.1007/s10586-023-04021-x.

[23] A.N. Malti, M. Hakem, B. Benmammar, A new hybrid multi-objective optimization algorithm for task scheduling in cloud systems, Cluster Comput 27 (2024) 2525–2548. https://doi.org/10.1007/s10586-023-04099-3.

[24] M. Osmanpoor, A. Shameli-Sendi, F. Faraji Daneshgar, Convergence of the Harris hawks optimization algorithm and fuzzy system for cloud-based task scheduling enhancement, Cluster Comput (2024). https://doi.org/10.1007/s10586-023-04225-1.

[25] M.A. Abu-Hashem, M. Shehab, M.K.Y. Shambour, M.Sh. Daoud, L. Abualigah, Improved Black Widow Optimization: An investigation into enhancing cloud task scheduling efficiency, Sustainable Computing: Informatics and Systems 41 (2024) 100949. https://doi.org/https://doi.org/10.1016/j.suscom.2023.100949.

[26] R. Gong, D.L. Li, L. La Hong, N.X. Xie, Task scheduling in cloud computing environment based on enhanced marine predator algorithm, Cluster Comput 27 (2024) 1109–1123. https://doi.org/10.1007/s10586-023-04054-2.

[27] J.O. Agushaka, A.E. Ezugwu, L. Abualigah, Dwarf Mongoose Optimization Algorithm, Comput Methods Appl Mech Eng 391 (2022). https://doi.org/10.1016/j.cma.2022.114570.

[28] P. Pirozmand, H. Jalalinejad, A.A.R. Hosseinabadi, S. Mirkamali, Y. Li, An improved particle swarm optimization algorithm for task scheduling in cloud computing, J Ambient Intell Humaniz Comput 14 (2023) 4313–4327. https://doi.org/10.1007/s12652-023-04541-9.

[29] X. Huang, M. Xie, D. An, S. Su, Z. Zhang, Task scheduling in cloud computing based on grey wolf optimization with a new encoding mechanism, Parallel Comput 122 (2024). https://doi.org/10.1016/j.parco.2024.103111.