Enhanced Harris Hawks Optimization Algorithm for SLA-Aware Task Scheduling in Cloud Computing

Junhua Liu*, Chaoyang Lei, Gen Yin
Hunan Post and Telecommunication College, Changsha 410015, Hunan, China

***Abstract*—Cloud computing has revolutionized how Software as a Service (SaaS) suppliers deliver applications by leasing shareable resources from Infrastructure as a Service (IaaS) suppliers. However, meeting users' Quality of Service (QoS) parameters while maximizing profits from the cloud infrastructure presents a significant challenge. This study addresses this challenge by proposing an Enhanced Harris Hawks Optimization (EHHO) algorithm for cloud task scheduling, specifically designed to satisfy Service Level Agreements (SLAs), meet users QoS requirements, and enhance resource utilization efficiency. Drawing inspiration from Harris's falcon hunting habits in nature, the basic HHO algorithm has shown promise in finding optimal solutions to specific problems. However, it often suffers from convergence to local optima, impairing solution quality. To mitigate this issue, our study enhances the HHO algorithm by introducing an exploration factor that optimizes parameters and improves its exploration capabilities. The proposed EHHO algorithm is assessed against established optimization algorithms, including Genetic Algorithm (GA), Ant Colony Optimization (ACO), and Particle Swarm Optimization (PSO). The results demonstrate that our method significantly improves the makespan for GA, ACO, and PSO by 19.2%, 17.1%, and 20.4%, respectively, while also achieving improvements of 17.1%, 17.3%, and 17.2% for BigDataBench workloads. Furthermore, our EHHO algorithm exhibits a substantial reduction in SLA violations compared to PSO, ACO, and GA, achieving improvements of 55.2%, 41.4%, and 33.6%, respectively, for general workloads, and 61.9%, 23.1%, and 52.7%, respectively, for BigDataBench workloads.**

Keywords*—Cloud computing; scheduling; optimization; SLA; SaaS

I. Introduction

Cloud computing represents an approach that facilitates migrating or deploying users' current physical infrastructure into a cloud-based environment. Users can access a wide array of services within this paradigm, including network, storage, computing, and memory, per their on-demand requirements [1], [2]. Virtualization technology plays a crucial role in provisioning a virtual infrastructure for users within a cloud environment. Service Level Agreement (SLA) serves as the contractual agreement between users and cloud providers, outlining the terms of service subscription [3]. Based on the established SLA, the cloud provider provisions the necessary services to meet users' needs. A distinguishing feature of the cloud computing environment is its inherent scalability, enabling services to be dynamically scaled up or down as required [4]. Resource pooling is a significant attribute within the cloud computing paradigm, wherein resources are shared and assigned to users under their specific demands. The cloud provider employs an automated approach to allocate virtual resources to users in compliance with the established SLA and the pay-per-usage policy [5]. A well-designed scheduling scheme is essential to facilitate resource allocation, enabling the automatic distribution of virtual resources to users. Furthermore, establishing a relationship between user requests and virtual machines (VMs) becomes crucial for efficient resource allocation. Given the diverse user base in the cloud computing environment, the implementation of an optimal task-scheduling mechanism becomes imperative. Additionally, a reliable and scalable resource provisioning mechanism is necessary to allocate resources to a large number of users automatically [6].

In the cloud computing environment, user requests are diverse in terms of sizes and types, including streaming data, video, images, text, etc. These requests can originate from different heterogeneous resources [7]. Therefore, a robust task-scheduling algorithm is required to schedule these heterogeneous, variable, and dynamic users' requests onto suitable VMs. Effective task scheduling is crucial to prevent Quality of Service (QoS) degradation and ensure compliance with SLA parameters that establish trust between users and cloud providers. A well-designed task scheduling algorithm should maximize QoS while maintaining SLA requirements, thus enhancing trust between users and cloud providers[8]. In recent years, several research works have focused on task scheduling in the cloud computing domain, utilizing metaheuristic approaches. These metaheuristic optimization algorithms are employed because task scheduling is a complex problem categorized as NP-hard. Using metaheuristic algorithms helps find near-optimal or feasible solutions for scheduling tasks to appropriate VMs in the cloud computing environment. By leveraging metaheuristic optimization algorithms, researchers aim to address the challenges posed by the NP-hard nature of task scheduling in cloud computing, ultimately improving the efficiency and effectiveness of resource allocation and meeting user requirements.

This paper proposes an innovative approach based on the Enhanced Harris Hawks Optimization (EHHO) algorithm. The EHHO algorithm draws inspiration from the hunting behavior of Harris's falcons in nature, which has shown remarkable abilities in finding optimal solutions for specific problems. By utilizing the EHHO algorithm, we aim to achieve improved task scheduling performance, enhanced resource utilization, and better compliance with SLAs and users' QoS requirements. The primary objective of this study is to investigate the efficacy of the EHHO algorithm in cloud task scheduling and assess its performance compared to existing optimization algorithms. We conduct extensive simulations and evaluations, considering both general workloads and specific BigDataBench workloads, to comprehensively analyze the performance of the proposed algorithm. The remainder of this paper is organized as follows: Section II provides an overview of related work in cloud task scheduling and optimization algorithms. Section III presents the methodology and details of the Enhanced Harris Hawks Optimization algorithm, including the enhancements made to mitigate convergence issues. Section IV describes the experimental setup and evaluation metrics used to assess the performance of the EHHO algorithm. Section V presents the results and analysis of the simulations. Finally, Section VI summarizes the findings, discusses their implications, and outlines future research directions.

II. Related Work

The paper in [9] proposed a novel algorithm called Interval Multi-objective Cloud Task Scheduling Optimization (I-MCTSO) to effectively address uncertainty in cloud task scheduling. They transformed ambiguous variables into precisely defined interval parameters, considering factors such as makespan, task completion rate, load balancing, and scheduling cost. To implement the I-MCTSO approach, the researchers devised a new Interval Multi-objective Evolutionary approach (InMaOEA). They integrated a distinct interval credibility approach to enhance convergence performance and augmented population diversity by incorporating overlap and hyper-volume assessments alongside the interval congestion distance method. Empirical simulations were conducted to evaluate the performance of the InMaOEA algorithm against existing algorithms. The results provided compelling evidence supporting the high effectiveness and superiority of the proposed approach. The methodologies furnish a framework that provides decision-makers with robust guidelines for allocating cloud job scheduling, enabling well-informed decisions. These advancements represent a significant progression in cloud computing resource management and potentially elevate operational efficiency and effectiveness.

This study [10] proposed an innovative enhancement to the initialization process of the PSO algorithm by integrating heuristic techniques. They incorporated the Minimum Completion Time (MCT) and Longest Job to Fastest Processor (LJFP) algorithms into the initialization phase of the PSO algorithm, aiming to improve its overall efficiency. The researchers comprehensively evaluated the formulated MCT-PSO and LJFP-PSO algorithms, considering several crucial metrics. These metrics included the minimization of makespan, reduction in overall energy consumption, mitigation of imbalance, and decrease in total execution time. These metrics served as pivotal benchmarks to assess the effectiveness of the proposed algorithms in the context of task scheduling. Through extensive simulations, the researchers presented evidence demonstrating the notable superiority and efficacy of the suggested MCT-PSO and LJFP-PSO approaches compared to traditional PSO methods and other contemporary task-scheduling algorithms. These findings underscored the potential of these enhancements to significantly improve the optimization capabilities of task scheduling methods based on the PSO algorithm. Consequently, this research contributes significantly to advancing the efficient and effective management of cloud computing resources.

In research [11], it introduced a task scheduling method called Chemical Reaction PSO. This method offers a hybrid approach that efficiently allocates multiple independent tasks among a collection of VMs in cloud computing environments. The proposed method combines the advantages of traditional chemical reaction optimization and particle swarm optimization, creating a unique synergy that leads to an optimal sequence for task scheduling. This sequence considers both task demand and deadline considerations, thereby improving outcomes across various parameters such as cost, energy consumption, and makespan. To evaluate the effectiveness of the proposed algorithm, extensive simulation experiments were conducted using the CloudSim toolbox. The experimental results highlighted the benefits of the Chemical Reaction PSO algorithm. The average execution time was rigorously assessed by comparing studies involving different quantities of VMs and jobs. The results demonstrated substantial improvements in execution duration, ranging from 1% to 6%, with specific instances showing even more significant improvements exceeding 10%. The makespan results also exhibited noteworthy gains, ranging from 5% to 12%, while the overall cost factor demonstrated enhancements of 2% to 10%. Furthermore, there was a significant increase in the rate of energy consumption, ranging from 1% to 9%.

The paper in [12] developed the Enhanced Sunflower Optimization (ESFO) algorithm as an innovative methodology to enhance the effectiveness of existing job scheduling techniques. The ESFO algorithm aims to achieve optimal scheduling within polynomial time complexity. The proposed ESFO approach underwent comprehensive scrutiny and was subjected to a battery of task scheduling benchmarks to evaluate its strengths and limitations. Simulation studies were conducted to assess the performance of the ESFO algorithm compared to existing algorithms. The outcomes of these studies demonstrated the superior performance of the ESFO algorithm. It exhibited significant proficiency in optimizing task scheduling outcomes, particularly in critical parameters such as energy usage and makespan. The algorithm's robust performance across these parameters highlighted its effectiveness in improving resource allocation and system efficiency.

The authors in [4] introduced the Enhanced Marine Predator Algorithm (EMPA) as a means to enhance scheduling efficiency. The proposed methodology consists of several crucial stages, including formulating a task scheduling model that considers both makespan and resource utilization. Each element within the algorithm represents a potential solution for task scheduling, aiming to identify the most favorable scheduling solution. To improve its performance, the EMPA algorithm integrates various components derived from the Whale Optimization Algorithm (WOA), incorporating operator functions, nonlinear inertia weight coefficients, and the golden sine function. To evaluate its effectiveness, the EMPA algorithm undergoes extensive comparative assessments against established optimization algorithms, such as WOA, PSO, SCA, and GWO, across diverse settings considering different workloads in the GoCJ and synthetic datasets. The empirical evaluation conducted in this study highlights the advantages of the EMPA algorithm, demonstrating notable strengths in resource utilization, degree of imbalance, and makespan. These findings provide empirical evidence supporting the efficacy of the Enhanced Marine Predator Algorithm in optimizing task scheduling outcomes. As a result, these results contribute significantly to the field of scheduling approaches and can potentially enhance resource management in various applications.

The paper in [13] proposed a multi-objective scheduling algorithm called MSITGO, which aims to optimize three conflicting objectives: idle resource costs, energy consumption, and batch task completion time. Drawing inspiration from Invasive Tumor Growth Optimization (ITGO), the MSITGO algorithm incorporates tumor cell growth modeling principles and integrates Pareto optimum and packing problem models. This integration enables a comprehensive and efficient exploration of potential solutions, expanding the range of ideas and accelerating the consensus-building process. Moreover, the MSITGO framework encompasses the entire task-processing operation by dividing it into two distinct stages: machine assignment and timeslot allocation. This refined framework enhances job scheduling efficiency and mitigates improper allocations. To validate its practical application, MSITGO undergoes empirical validation using real cluster data obtained from Alibaba. The experimental results demonstrate the superiority of MSITGO over existing techniques in addressing the multi-objective task scheduling problem. The framework exhibits its ability to provide more efficient solutions, highlighting its potential to make significant contributions to optimizing task scheduling across various applications.

III. Problem Statement and System Model

In this section, we define the problem statement and introduce the proposed architecture for task scheduling. The problem at hand revolves around the mapping of a set of n tasks, represented as tn = {t1, t2, ..., tn}, onto the m VMs vmm = {vm1, vm2, ..., vmm}, exist within the Hk hosts Hk = {H1, H2, ..., Hk}, which are situated within the Dn datacenters Dn = {D1, D2, ..., Dn}. During this mapping process, the priorities of both VMs and tasks are taken into account. The primary objectives of this mapping are to minimize the makespan and prevent SLA violations.

Fig. 1 provides a visual representation of the proposed system architecture. The process begins with simultaneous user queries being submitted to the cloud administration dashboard and broker, which act as users' agents. The task manager then validates these requests, which considers the specified SLA requirements. If the requests meet the criteria and are deemed valid, they are placed in a waiting queue and subsequently forwarded to the task scheduler. Within this architecture, the task manager is crucial in calculating the priorities of diverse and heterogeneous tasks. These priorities are determined based on factors such as task size, run-time capacity, and the preferences of the VMs.

Additionally, the VM priorities are determined by considering the unit cost of electricity associated with each VM. After determining the priorities of tasks and VMs, they are placed in a waiting line. The task scheduler then assigns the highest-priority task to the highest-priority VM. The scheduler tries to reduce the makespan and prevent SLA breaches by categorizing the requests based on these priorities. The task scheduler plays a crucial role in efficiently mapping tasks to VMs while considering their priorities. It takes into account the optimization objectives of minimizing the makespan and ensuring compliance with SLAs. By intelligently assigning tasks to VMs based on their priorities, the scheduler aims to achieve an optimal task scheduling assignment, leading to improved system performance and user satisfaction.

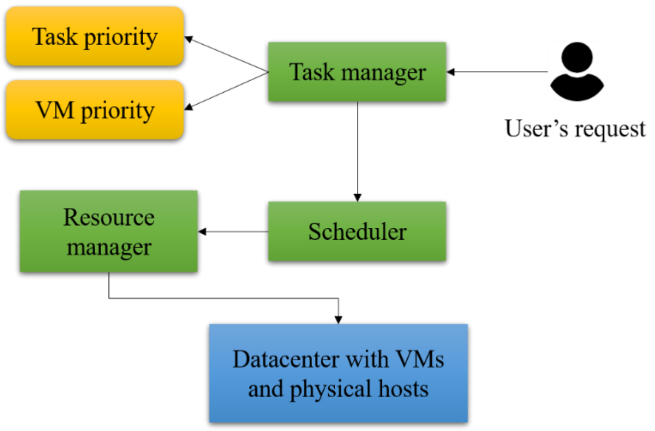

Fig. 1. System architecture.

To evaluate the priorities of tasks, the workload on all VMs is calculated using Eq. (1), where lom represents the workload on m VMs residing in the set of Hk hosts. Consequently, the total workload on hosts is calculated using Eq. (2).

$$lo_{vm_m} = \sum lo^m \tag{1}$$

$$lo_{H_k} = \frac{lo_{vm_m}}{\sum H_k} \tag{2}$$

To determine whether user requests or tasks can be processed on a specific VM, the processing capacity of a VM needs to be defined. This is indicated by Eq. (3), where prono represents the number of processing elements and proMIPS stands for the processing capacity based on the number of instructions processed per second.

$$pro_{ca_{vm}} = pro_{MIPS} \times pro_{no} \tag{3}$$

For the task scheduler to map tasks to specific VMs, it requires knowledge of the task size, which is calculated using Eq. (4). Subsequently, the priorities of all tasks are calculated using Eq. (5), while the priorities of VMs, based on unit electricity cost, are determined using Eq. (6).

$$t_k^{len} = t_{pr_k} \times t^{MIPS} \tag{4}$$

$$t_{pr_k} = \frac{t_k^{len}}{pro_{k_{vm}}} \tag{5}$$

$$vm_{pr_n} = \frac{elecost^{high}}{elecost_{d_i}} \tag{6}$$

The primary goals of this research endeavor encompass the proper mapping of tasks to virtual resources, with a focus on minimizing the makespan and avoiding any violations of service level agreements (SLAs). To evaluate the makespan, Eq. (7) is employed as the metric. Subsequently, the determination of SLA violations becomes the next objective. SLA violations are influenced by two key factors: the active time of a host and performance degradation. These factors are quantified using Eq. (8) and (9), respectively. By utilizing these equations, the calculation of SLA violations can be performed, as expressed in Eq. (10).

$$ms^k = e^k + ava^n \tag{7}$$

$$AT_{H_i} = \frac{1}{p}\sum_{s=1}^{p}\frac{vio\ time_{H_i}}{AT_{H_i}} \tag{8}$$

$$pe_{dg} = \frac{1}{n}\sum_{a=1}^{n}\frac{pe_{dg}^{p}}{to_{vm}^{p}} \tag{9}$$

$$SLA_{vio} = pe_{dg} \times AT_{H_i} \tag{10}$$

IV. Enhanced HHO for Task Scheduling

The HHO algorithm draws inspiration from the cooperative hunting and pursuit behaviors observed in Harris's hawks, specifically their strategic hunting tactics like "surprise pounces" or "the seven kills"[14]. In cooperative attacks, multiple hawks collaborate to pursue a rabbit that has revealed itself, aiming to catch the prey swiftly. However, the hunt might include repeated rapid dives near the prey, depending on the prey's reactions and its potential to escape. Harris's hawks display various hunting strategies based on the changing circumstances and the prey's escape patterns. Tactics are often altered if the lead hawk fails to pursue the prey, allowing another team member to continue the chase, often used to confuse escaping rabbits. Notably, the rabbit is unable to regain its defensive skills when a new hawk initiates the chase, and it cannot escape the attacking team as the most experienced hawk captures and shares the exhausted rabbit.

The different phases of the HHO are depicted in Fig. 2, illustrating how hawks trace, encircle, and ultimately attack their prey. The mathematical model mirrors these hunting behaviors, encompassing three phases: exploration, transition between exploration and exploitation, and exploitation. Throughout each phase, Harris's hawks represent potential solutions, and the target prey represents the optimal solution. Hawks use two exploration techniques to locate the prey. In one, they select a location based on other hawks' positions and the prey's location. In the second strategy, hawks perch randomly on tall trees. Eq. (11) simulates these methods with equal probabilities using random numbers.

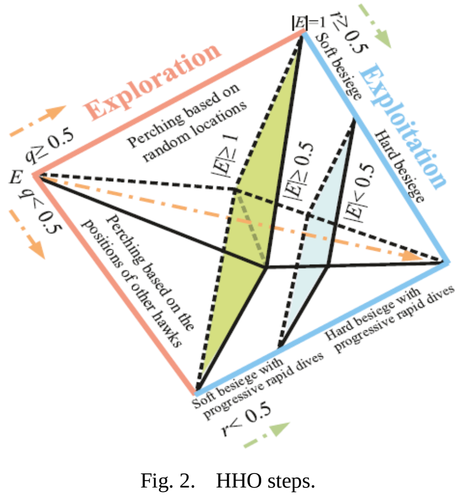
Exploration
Exploitation
|E|=1
r≥ 0.5
Soft besiege
Hard besiege
q≥ 0.5
Perching based on random locations
E
q< 0.5
Perching based on the positions of other hawks
|E|≥ 1
|E|≥ 0.5
|E|< 0.5
Soft besiege with progressive rapid dives
Hard besiege with progressive rapid dives
r< 0.5

Fig. 2. HHO steps.

$$x(t+1) = \begin{cases} x_{random}(t) - x_1|x_{random}(t) - 2r_2x(t)|, q \geq 0.5 \\ x_{rabbit}(t) - x_{mean}(t) - r_3\big(LB + r_4(UB - LB)\big), q < 0.5 \end{cases} \tag{11}$$

Eq. (12) calculates the average hawk population position. The algorithm switches from exploration to exploitation based on the rabbit's energy, as expressed in Eq. (13). When the rabbit's escaping energy $|E|\geqslant 1$, hawks explore more areas; otherwise, exploitation begins. Eq. (14) - Eq. (17) determine whether hawks perform a soft or hard siege based on the rabbit's energy and escape success. A soft siege involves repeated dives, simulating the rabbit's successful escape, while a hard siege is calculated differently.

$$x_{mean}(t) = \frac{1}{N}\sum_{i=1}^{N} x_i(t) \tag{12}$$

$$E = 2E_0(1 - \frac{t}{Max\_iter}) \tag{13}$$

$$x(t+1) = \Delta x(t) - E|J._{xrabbit}(t) - x(t)| \tag{14}$$

$$\Delta x(t) = x_{rabbit}(t) - x(t) \tag{15}$$

$$J = 2(1 - random) \tag{16}$$

$$x(t+1) = x(t) - E|\Delta x(t)| \tag{17}$$

Eq. (18) - Eq. (21) governs the soft-siege rapid dives, utilizing Lévy flights to mimic the prey's behaviour. Eq. (18) and (19) calculate the hawks' actions during the dive, while Eq. (20) and Eq. (21) reflect the final soft-siege rapid dives and the parameters k and z during a hard siege, respectively.

$$k = x_{rabbit}(t) - E|J.x_{rabbit}(t) - x(t)| \tag{18}$$

$$z = k + RandomVector.L(dim) \tag{19}$$

$$x(t+1) = \begin{cases} k, iff(k) < f(x(t)) \\ z, iff(z) < f(x(t)) \end{cases} \tag{20}$$

$$k = x_{rabbit}(t) - E|J.x_{rabbit}(t) - x_{mean}(t)| \tag{21}$$

In the exploration phase of the HHO algorithm, the calculations pertaining to positions, specified in Eq. (11) and Eq. (12), are influenced by random values r1 and r3 within the range of (0, 1). While this stochastic approach fosters randomness in each step during the global search, it lacks the necessary variability. During this phase, the original HHO algorithm operates under the assumption that hawks, with their keen eyes, can generally track and detect prey; however, there are moments when prey is elusive and might not be detected easily, sometimes even after several hours. In light of these observations, it seems plausible to consider adjusting these parameters to render them more adaptable.

We propose to conceptualize r1 and r3 as indicative of the step length, where larger values imply swifter movement for the hawks, and conversely, smaller values correspond to slower movement. There exist two scenarios for a hawk to find prey: one scenario involves immediate detection, while the other involves a prolonged search. In the former, it is essential to account for the variability in step length, whereas, in the latter scenario, the overall variability of the step length should diminish. As time progresses, the likelihood of a hawk finding prey increases; therefore, initially, hawks should explore a wider range with larger steps, gradually transitioning to a more methodical search in later iterations. Thus, we propose an update to r1 and r3 using an exploration factor represented by Eq. (17). Consequently, the modified Eq. (11) is updated as follows Eq. (18):

$$ef = (b \times rand - \frac{b}{2}) \times cos(\frac{\pi}{2} \times (\frac{t}{T})^2) \tag{22}$$

$$X(t+1) = \begin{cases} X_{rand}(t) - ef|X_{rand}(t) - 2r_2X(t)|, q \geq 0.5 \\ (X_{rabbit}(t) - X_m(t)) - ef(LB + r_4(UB - LB)), q < 0.5 \end{cases} \tag{23}$$

Here, the value of b is set to 2 based on favorable results from experimental tests. The term (b ∗ rand − b/2) introduces randomness in the step length by generating random numbers within the interval of (−b/2, b/2). In essence, the exploration factor initially widens the step length range from (0, 1) to (−b/2, b/2) to support expansive exploration. As the number of iterations increases, it gradually shifts the exploration process from a broad range to a more constrained one. Ultimately, this approach maintains the essential randomness in the step length while adapting it dynamically over the course of iterations.

The choice of parameters in EHHO algorithm is critical for optimizing its performance in task scheduling within cloud environments. The parameter *b* is set to 2 based on favorable outcomes from preliminary experimental tests, which suggests that this value effectively balances the exploration and exploitation phases of the algorithm. The exploration factor (*ef*), introduced in Eq. (22), modifies the step length of hawk movements, thereby enhancing the algorithm's ability to search for optimal solutions dynamically. The term *(b×rand−b/2)* adds randomness within the interval (−b/2, b/2), initially broadening the step length to support wide-ranging exploration and then gradually narrowing it to facilitate a more focused search as iterations progress. This adaptation ensures the algorithm maintains its stochastic nature while becoming more methodical over time. The experimental design rationale involves simulating the EHHO algorithm against established optimization algorithms like GA, ACO, and PSO, across varying workloads to evaluate its efficacy. The validation process entails comparing key performance metrics, such as makespan and SLA violations, demonstrating significant improvements in both general and BigDataBench workloads.

V. Experimental Results

This section discusses the configuration settings for simulation and presents the simulation results. The simulation was conducted using the CloudSim toolkit, which provides an accurate environment for simulating the cloud paradigm. The simulation environment utilized in this study was implemented on a machine with an Intel Core i5 processor and 8 GB of RAM. Table I shows configuration settings for simulation. Table III outlines the specific standard configuration settings utilized in the simulation.

Table II presents the computation of SLA violations for different algorithms, including PSO, ACO, GA, and our proposed algorithm (EHHO), considering varying task quantities.

TABLE I. Configuration Settings for Simulation

Parameter	Value
Datacenter count	5
Operating system	Linux
Virtual machine monitor	Xen
VM bandwidth	5 Mbps
VM memory	1024 MB
VM count	20
Network bandwidth	1000Mbps
Host storage capacity	5 TB
Host memory	16 GB
Task length	780,000
Task count	100-1000

TABLE II. SLA Violations for Randomly Generated Workloads

Task count	GA	ACO	PSO	EHHO
100	15	12	17	7
500	12	18	25	9
1000	21	22	28	18

The selection of GA, ACO, and PSO for comparison against our proposed EHHO algorithm is rooted in the distinct strengths and prevalent application of these algorithms in the field of optimization and task scheduling. Each of these algorithms represents a different heuristic approach to solving complex optimization problems, making them ideal benchmarks for assessing the performance of EHHO. The Genetic Algorithm (GA) is an evolutionary algorithm that simulates the process of natural selection. It operates through mechanisms inspired by biological evolution, such as selection, crossover, and mutation. GA's robustness in exploring large search spaces and finding near-optimal solutions is well-documented, making it a common choice for various scheduling and optimization tasks. By comparing EHHO to GA, we can evaluate how well our algorithm performs in terms of scalability and efficiency, especially in complex environments where traditional methods might struggle.

The ACO and PSO were chosen due to their distinct nature and widespread use in optimization problems. ACO is inspired by the foraging behavior of ants and is particularly effective in finding optimal paths and solutions through a collaborative approach. Its performance in scheduling tasks is noteworthy, making it a suitable candidate for comparison. PSO, on the other hand, simulates the social behavior of birds flocking or fish schooling. It is known for its simplicity and fast convergence rates, making it a popular choice for various optimization problems, including resource scheduling and allocation. By including ACO and PSO in our comparative analysis, we cover a broad spectrum of heuristic optimization techniques. This allows us to comprehensively assess the efficiency, scalability, and robustness of EHHO in minimizing SLA violations and makespan across different workload scenarios, thereby highlighting its potential advantages and areas of improvement in real-world applications.

When subjected to randomly generated workloads, the SLA violations recorded for the PSO algorithm were 17%, 25%, and 28%, respectively. For ACO, the corresponding SLA violations are 12%, 18%, and 22%. GA yields SLA violations of 15%, 12%, and 21%, while EHHO results in SLA violations of 7%, 9%, and 18%. In Table III, we present the assessment of SLA violations incurred by different algorithms across varying task quantities. These evaluations were conducted using the BigDataBench workload as the basis for generating tasks. For PSO, the SLA violations are 18%, 21%, and 29%. ACO yields SLA violations of 10%, 12%, and 18%. GA generates SLA violations of 18%, 21%, and 29%. EHHO results in SLA violations of 9%, 11%, and 13%. It is evident that EHHO significantly reduces SLA violations over other algorithms. By considering the priority of VMs and tasks, our algorithm efficiently schedules the tasks, resulting in a minimized makespan.

TABLE III. SLA VIOLATIONS FOR BIGDATABENCH WORKLOADS

Task count	GA	ACO	PSO	EHHO
100	18	10	18	9
500	21	12	21	11
1000	29	18	29	13

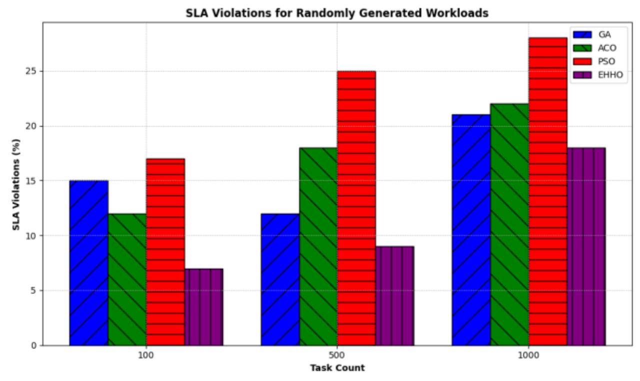

Fig. 3. Visual representation of SLA violations for randomly generated workloads.

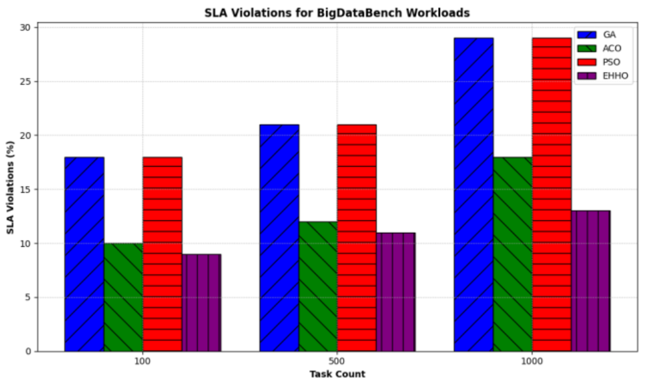

Fig. 4. Visual representation of SLA violations for bigdata bench workloads.

Table IV presents the calculated makespan values for different algorithms for three task quantities. In the case of randomly generated workloads, the makespan values obtained for PSO were 1289, 1678, and 1989, respectively, for the three task quantities 100, 500, and 1000. For ACO, the corresponding makespans are 1156, 1563, and 2146. GA yields makespans of 1543, 1475, and 1934, while the proposed algorithm results in makespans of 976, 1281, and 1814. Table V presents the calculated makespan values for different algorithms using the BigDataBench workload, considering task quantities of 100, 500, and 1000. For PSO, the makespans are 1367, 1747, and 2045. ACO yields makespans of 1243, 1643, and 2387. GA generates makespans of 1437, 1532, and 2243, while the proposed algorithm results in makespans of 1087, 1407, and 1882. Fig. 3, 4, 5 and 6 show visual representation for different workloads.

TABLE IV. MAKESPAN FOR RANDOMLY GENERATED WORKLOADS

Task count	GA	ACO	PSO	EHHO
100	1543	1156	1289	976
500	1475	1563	1678	1281
1000	1934	2146	1989	1814

TABLE V. MAKESPAN FOR BIGDATABENCH WORKLOADS

Task count	GA	ACO	PSO	EHHO
100	1437	1243	1367	1087
500	1532	1643	1747	1407
1000	2243	2387	2045	1882

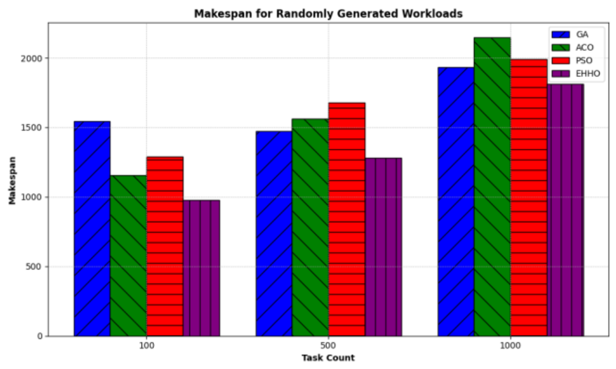

Fig. 5. Visual representation of makespan for randomly generated workloads.

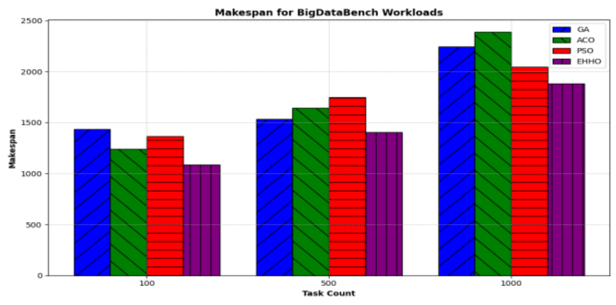

Fig. 6. Visual representation of makespan for bigdatabench workloads.

VI. Discussion

The EHHO algorithm has shown significant improvements over traditional algorithms like GA, ACO, and PSO in optimizing makespan and reducing SLA violations, which suggests it has a strong foundation for handling larger and more complex workloads. The inherent design of the EHHO, which draws from the cooperative hunting strategies of Harris's hawks, allows it to dynamically adjust its exploration and exploitation phases. This dynamic adjustment is crucial for scalability because it enables the algorithm to maintain efficiency as the number of tasks and VMs scales up. The exploration factor introduced in the EHHO enhances its capability to search a wider solution space initially and then focus on more promising areas, which is beneficial when dealing with large-scale environments.

Cloud computing environments are highly dynamic, with workloads and resource availability fluctuating rapidly. The adaptability of the EHHO algorithm in such conditions is supported by its enhanced exploration mechanism, which allows for a more flexible search process. The algorithm can adjust its step lengths and exploration range based on the iteration progress and current solution quality, helping it adapt to sudden changes in workload patterns and resource distribution.

The scalability of the proposed EHHO algorithm is a critical factor for its practical application in diverse cloud computing environments, characterized by varying loads and resource distribution patterns. Scalability in this context refers to the algorithm's ability to maintain or improve its performance as the size of the cloud environment increases and as it adapts to changing conditions.

Moreover, the use of random values in the EHHO's exploration phase fosters a level of stochasticity that can be beneficial in diverse environments. This randomness ensures that the algorithm does not become overly dependent on specific patterns and can handle unexpected changes more effectively. While the EHHO algorithm has demonstrated improved performance metrics, its scalability also depends on managing computational overhead. The algorithm's complexity, particularly in large-scale environments, could potentially introduce significant computational costs. To mitigate this, the EHHO can be parallelized and optimized to run on distributed cloud infrastructure, leveraging the parallel processing capabilities of modern cloud systems. This parallelization can distribute the computational load, ensuring that the algorithm remains efficient even as the scale of the environment increases.

For addressing real-world scenarios challenges, implementing the EHHO algorithm for cloud task scheduling in real-world scenarios presents several potential challenges. One of the primary challenges is the dynamic and unpredictable nature of cloud environments. Cloud infrastructures often experience varying workloads and resource availability, making it difficult to maintain consistent performance and SLA adherence. The EHHO algorithm, although optimized for exploration and preventing convergence to local optima, may still need continuous adjustments and fine-tuning to handle these dynamic changes effectively. Additionally, integrating the EHHO algorithm with existing cloud management platforms can be complex, requiring significant modifications to accommodate its unique optimization processes. This integration process must ensure minimal disruption to ongoing services and avoid introducing new inefficiencies.

Another challenge is the potential computational overhead introduced by the EHHO algorithm. While EHHO aims to optimize resource utilization and task scheduling, the algorithm itself can be computationally intensive, especially when handling large-scale cloud environments with numerous tasks and VMs. This computational demand can offset some of the performance gains achieved through optimized scheduling. Moreover, real-world applications often involve multi-tenant environments where multiple users and applications compete for resources. Ensuring fairness and effective resource allocation while using EHHO to maximize efficiency can be challenging. The algorithm must be designed to respect priority levels, application-specific QoS requirements, and user-specific SLAs, which can add layers of complexity to its implementation.

To address these challenges, several adaptations and enhancements can be incorporated into the EHHO algorithm. Firstly, implementing a feedback mechanism that continuously monitors the cloud environment and dynamically adjusts the EHHO parameters can help maintain optimal performance despite changes in workload patterns and resource availability. This adaptive approach can involve machine learning techniques that predict workload trends and preemptively adjust the EHHO algorithm's exploration and exploitation balance.

Secondly, to mitigate the computational overhead, the EHHO algorithm can be parallelized and optimized to run efficiently on distributed systems. Leveraging the inherent parallelism in cloud infrastructures can distribute the computational load of the EHHO algorithm, ensuring that it scales effectively with the size of the cloud environment. Additionally, introducing a hybrid approach that combines EHHO with other less computationally intensive algorithms can help balance the trade-offs between optimization quality and computational efficiency. For instance, using simpler heuristic methods for initial task scheduling and applying EHHO for fine-tuning can achieve a balance between performance and overhead.

Lastly, ensuring fairness and effective resource allocation in multi-tenant environments requires incorporating priority-based and QoS-aware scheduling policies into the EHHO algorithm. This can involve designing custom fitness functions that account for user-specific SLAs and QoS requirements, ensuring that the algorithm not only optimizes for overall resource utilization but also respects individual application needs. Regular audits and evaluations of the algorithm's performance in meeting SLAs and QoS parameters can help in making necessary adjustments and improvements, ensuring that EHHO remains effective in real-world cloud environments.

VII. Conclusion

The scheduling of tasks in cloud computing environments presents substantial issues for both cloud providers and customers. In the absence of an efficient scheduler, the diverse and heterogeneous workload can result in prolonged makespan and violations of SLAs, thereby compromising the overall QoS. To tackle these challenges, this study presented a novel task-scheduling algorithm that incorporates the priority of VMs and tasks to achieve optimal task-to-resource mapping. Our scheduling strategy builds upon the existing HHO algorithm, incorporating enhancements to improve its effectiveness. To evaluate and validate our proposed algorithm, we conducted comprehensive simulations and experiments using the CloudSim framework. The efficacy of the suggested algorithm is evaluated in comparison to established methodologies such as PSO, ACO, and GA. Initially, we used randomly generated workloads in the simulation, and later, we utilized a real-time dataset called BigDataBench. The results of our evaluation provide compelling evidence that our proposed algorithm surpasses the previous methods by optimizing SLA violations and makespan.

Despite these promising results, our study has some limitations. Firstly, the algorithm's performance has been tested primarily within simulated environments, which may not fully capture the complexities and variabilities of real-world cloud infrastructures. The computational overhead introduced by the enhanced HHO algorithm also needs further analysis to ensure scalability and efficiency in large-scale cloud deployments. Additionally, the algorithm currently focuses on optimizing makespan and SLA violations but does not explicitly address other crucial factors such as energy consumption, cost efficiency, and fairness in resource allocation among multiple tenants. Future research should aim to address these limitations by conducting real-world implementation and testing, exploring hybrid optimization techniques to balance computational efficiency, and integrating additional optimization objectives such as energy and cost savings. Expanding the algorithm's adaptability to diverse and evolving cloud environments will also be essential for its broader applicability and robustness.

Acknowledgment

This work was supported by Research on the promotion path of the core competitiveness of professional groups under the background of industry education integration (Subject No.: xjk19bzy017).

References

[1] V. Hayyolalam, B. Pourghebleh, M. R. Chehrehzad, and A. A. Pourhaji Kazem, “Single-objective service composition methods in cloud manufacturing systems: Recent techniques, classification, and future trends,” Concurr Comput, vol. 34, no. 5, p. e6698, 2022.

[2] B. Pourghebleh, A. Aghaei Anvigh, A. R. Ramtin, and B. Mohammadi, “The importance of nature-inspired meta-heuristic algorithms for solving virtual machine consolidation problem in cloud environments,” Cluster Comput, vol. 24, no. 3, pp. 2673–2696, 2021.

[3] V. Hayyolalam, B. Pourghebleh, A. A. Pourhaji Kazem, and A. Ghaffari, “Exploring the state-of-the-art service composition approaches in cloud manufacturing systems to enhance upcoming techniques,” The International Journal of Advanced Manufacturing Technology, vol. 105, pp. 471–498, 2019.

[4] R. Gong, D. Li, L. Hong, and N. Xie, “Task scheduling in cloud computing environment based on enhanced marine predator algorithm,” Cluster Comput, pp. 1–15, 2023.

[5] K. Saidi and D. Bardou, “Task scheduling and VM placement to resource allocation in cloud computing: challenges and opportunities,” Cluster Comput, vol. 26, no. 5, pp. 3069–3087, 2023.

[6] B. Kruekaew and W. Kimpan, “Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning,” IEEE Access, vol. 10, pp. 17803–17818, 2022.

[7] S. Mangalampalli, G. R. Karri, and G. N. Satish, “Efficient workflow scheduling algorithm in cloud computing using whale optimization,” Procedia Comput Sci, vol. 218, pp. 1936–1945, 2023.

[8] P. Pirozmand, H. Jalalinejad, A. A. R. Hosseinabadi, S. Mirkamali, and Y. Li, “An improved particle swarm optimization algorithm for task scheduling in cloud computing,” J Ambient Intell Humaniz Comput, vol. 14, no. 4, pp. 4313–4327, 2023.

[9] Z. Zhang, M. Zhao, H. Wang, Z. Cui, and W. Zhang, “An efficient interval many-objective evolutionary algorithm for cloud task scheduling problem under uncertainty,” Inf Sci (N Y), vol. 583, pp. 56–72, 2022.

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

[11] K. Dubey and S. C. Sharma, “A novel multi-objective CR-PSO task scheduling algorithm with deadline constraint in cloud computing,” Sustainable Computing: Informatics and Systems, vol. 32, p. 100605, 2021.

[12] H. Emami, “Cloud task scheduling using enhanced sunflower optimization algorithm,” Ict Express, vol. 8, no. 1, pp. 97–100, 2022.

[13] Q. Hu, X. Wu, and S. Dong, “A two-stage multi-objective task scheduling framework based on invasive tumor growth optimization algorithm for cloud computing,” J Grid Comput, vol. 21, no. 2, p. 31, 2023.

[14] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, “Harris hawks’ optimization: Algorithm and applications,” Future generation computer systems, vol. 97, pp. 849–872, 2019.