HGWWO: A Hybrid Grey Wolf–Whale Optimizer for Load Balancing in Cloud Computing Environments

Yameng BAI[1]*, Junxia MENG[2], Shuai ZHAO[3], Ruoyu REN[4]
School of Information Engineering, Jiaozuo University, Jiaozuo 454150, China[1, 2]
School of Artificial Intelligence, Jiaozuo University, Jiaozuo 454150, China[3]
School of Safety Science and Engineering, Henan Polytechnic University, Jiaozuo 454150, China[4]

Abstract—**This paper aims to develop an efficient and adaptive load balancing algorithm for cloud computing environments using a novel hybrid meta-heuristic approach. Effective load balancing is necessary for optimum performance and resource utilization in cloud computing systems. Most conventional meta-heuristic algorithms suffer from premature convergence and poor exploration–exploitation tradeoffs. An innovative hybrid meta-heuristic algorithm, Hybrid Grey Wolf–Whale Optimizer (HGWWO), is proposed for efficiently and dynamically balancing cloud load. HGWWO integrates the leadership hierarchy and adaptive hunting strategy of the Grey Wolf Optimizer (GWO) with the spiral-shaped exploitation mechanism of the Whale Optimization Algorithm (WOA), resulting in high convergence rates. The algorithm is implemented in a multi-objective cloud load balancing model to reduce response time, energy usage, and makespan while optimizing resource utilization among virtual machines. The experimental outcomes prove that HGWWO outperforms existing algorithms regarding throughput, waiting time, and execution efficiency. The suggested model has potential for real-time cloud scheduling of resources and is an efficient solution for scalable and heterogeneous cloud environments.**

Keywords—Cloud computing; load balancing; hybrid meta-heuristic; grey wolf optimizer; whale optimization algorithm

I. Introduction

Cloud computing has transformed the delivery of computational resources and services, providing on-demand access to shared, configurable resources like Virtual Machines (VMs), storage, and applications over the Internet [1]. Cloud paradigms facilitate scalability, flexibility, and cost-effectiveness, essential in contemporary business and service organizations [2]. Nevertheless, the dynamic nature and dispersion of the cloud result in issues with providing resources, primarily load balancing [3]. Ineffective load balancing leads to bottlenecks in the system, resulting in poor resource utilization, high response times, and overall poor system performance [4]. Optimizing workload distribution among existing VMs while avoiding system imbalance is an essential design and administration issue in cloud computing infrastructures [5].

These issues, however, have been addressed through the design of numerous load balancing schemes, ranging from simple static approaches to more advanced and adaptive methodologies. Static algorithms in highly dynamic and heterogeneous clouds are ineffective since they cannot adapt to runtime variations in workloads and system resources [6]. In such cases, load balancing algorithms, in the form of multi-objective optimization, become pressing issues. The algorithms attempt to balance multiple objectives concurrently, including makespan, power usage, response time, and resource utilization [7]. Such problem-solving demands state-of-the-art optimization techniques for dealing with highly complicated search spaces and allowing system state changes. Related work across various fields, including machine learning for business forecasting [8], innovative grid system optimization [9], and security threat detection [10], similarly emphasizes the range and promise of state-of-the-art optimization and intelligent decision-making methodologies. Such models provide a compelling argument for highly adaptive solutions that are robust enough for intensive environments, such as cloud computing.

Past research has demonstrated the effectiveness of nature-inspired meta-heuristic algorithms for optimizing complex multi-objective problems in the cloud [11]. Grey Wolf Optimizer (GWO) and the Whale Optimization Algorithm (WOA) are distinguished by their excellent exploration and exploitation capacities. GWO mimics wolves' social hierarchy and hunting strategy [12], while WOA mimics humpback whale bubble-net hunting [13]. While the two algorithms promise particular strengths, they both carry the flaw of premature convergence, or a lack of thoroughness in local search, when employed in isolation. Inspired by the complementarities between the two tactics, this study proposes a hybrid algorithm, HGWWO that combines the social leadership process of GWO with the exploitation effectiveness of WOA to improve load balancing efficiency in the cloud.

Although more meta-heuristic algorithms have recently been applied to cloud load balancing, previous work often fails to adequately resolve the exploration-exploitation tradeoff under highly variable workloads. When used standalone, algorithms such as GWO or WOA tend to converge prematurely or become stuck in a local optimum. Although hybrid methodologies also exist, they tend to incur heavy computational overhead or be specialized for low scales, thereby limiting their generality. This work closes this gap by proposing HGWWO as a hybrid meta-heuristic that combines GWO's adaptive leader structure with spiral-guided localized searching from WOA, with a focus on being particularly appropriate for variable heterogeneous clouds. By integrating the merits of both algorithms, HGWWO will deliver a computationally efficient and generic solution for resolving the multi-objective cloud load balancing problem. Therefore, the primary research question guiding this study is: How can a hybrid meta-heuristic algorithm be developed to dynamically handle workload in heterogeneous clouds with minimal makespan, energy usage, and load deviation?

The rest of the paper is as follows: Section II reviews the literature on load balancing and meta-heuristic optimization. Section III describes the mathematical model of the load balancing problem and its objective functions. Section IV details the HGWWO algorithm. Section V provides simulation results, comparison analysis, and performance assessment. Section VI provides a critical discussion of the findings, including improvements and limitations. Finally, Section VII concludes the paper and offers directions for further work.

II. Literature Review

Cloud computing load balancing has garnered significant research interest due to its importance in optimizing resource usage, minimizing response time, and ensuring Quality of Service (QoS). Several heuristic and meta-heuristic methods have been proposed for addressing the dynamic and multi-objective nature of the problem. Muteeh, et al. [14] presented an Ant Colony Optimization (ACO)-based multi-resource load balancing algorithm to solve the issues in the scheduling of scientific workflows in cloud computing. Their approach optimized the makespan and cost, ensuring balanced load distribution among cloud resources. Experimental verification showed enhanced resource utilization and a decrease in both the price and the running time.

Sefati, et al. [15] proposed load balancing using GWO, focusing on the algorithm for identifying overloaded and underutilized nodes. The algorithm includes using the fitness value and reliability of the nodes to direct resource assignment. CloudSim experiments demonstrated significant reductions in response time and execution cost compared to other typical methods. Ramya and Ayothi [16] proposed a hybrid Dingo and Whale Optimization Algorithm (HDWOA-LBM) by integrating dingo and whale hunting behaviors to achieve effective task allocation. The algorithm enhances both exploration and exploitation capabilities. Simulation results validated its efficiency, showing improvements in makespan (22.9%), throughput (21.2%), and reliability (25.4%).

Geetha, et al. [17] created an innovative Intercrossed Chimp and Bald Eagle Algorithm (IC&BA) incorporating chimp optimization and bald eagle searching. Their approach considers concurrently energy usage, execution cost, makespan, and turnaround time. Study results validated improved balancing and performance in heterogeneous environments.

Emara, et al. [18] presented an advanced Harris Hawks Optimization (HHO) for task scheduling in cloud computing. Their approach enhanced the exploration and exploitation stages through mutation-based diversification, thereby outdoing the conventional HHO and other metaheuristics in makespan, throughput, and load variance.

Haris and Zubair [19] developed a hybrid Battle Royale and Deep Reinforcement Learning (BRDRL) dynamic load balancing algorithm. The approach combines job migration through DRL and VM selection through BRO. Simulations demonstrated that BRDRL resulted in a 3.9% improvement in throughput and a 15.3% improvement in response time compared to existing models and hybrids.

Tabagchi Milan, et al. [20] suggested an Artificial Bee Colony (ABC)-derived method for green cloud computing. Overloaded VM tasks are migrated according to the bee behavior to enhance QoS and energy savings. The experimental results indicate improvement in energy and makespan.

While all the above approaches support load balancing in cloud computing, most algorithms are plagued with tradeoffs between convergence rates and exploitation accuracy, as highlighted in Table I. For example, ACO and GWO-based algorithms converge prematurely or become stuck in local optima in dynamic environments. Hybrid algorithms like HDWOA and BRDRL improve performance at the expense of being computationally intensive and having limited generalizability for use in other cloud infrastructures.

TABLE I. An Overview of Recent Cloud Load Balancing Algorithms

Ref	Algorithm	Features	Limitations
[14]	Ant colony optimization	It supports multi-resource allocation, workflow-aware scheduling, and effectively reduces cost and execution time.	May experience slow convergence and exhibit limited adaptability to dynamic load variations.
[15]	Grey wolf optimizer	It accounts for node reliability and adapts to load conditions, including underloaded and overloaded nodes.	Prone to entrapment in local optima and characterized by limited exploitation capability.
[16]	Hybrid dingo and whale optimization algorithm	The approach enhances exploration and exploitation mechanisms, improving makespan and increasing system reliability.	Exhibits high computational complexity, with potential performance degradation under task overload or in the presence of noisy objective functions.
[17]	Intercrossed chimp and bald eagle algorithm	The method employs a multi-criteria load balancing approach that accounts for cost, energy consumption, and response time through a hybrid strategy.	The approach may require extensive parameter tuning and demonstrates limited scalability for general-purpose use.
[18]	Harris hawks optimization	It employs adaptive mutation to strengthen global search efficiency, improving task scheduling accuracy.	Remains sensitive to the initial population and incurs additional computational overhead.
[19]	Hybrid battle royale and deep reinforcement learning	It combines DRL for job transfer with the BRO algorithm for VM selection, facilitating dynamic and adaptive decision-making.	It necessitates large volumes of training data, and integrating deep models contributes to substantial computational overhead.
[20]	Artificial bee colony	It utilizes bio-inspired task migration techniques to reduce energy consumption while maintaining or improving QoS.	Demonstrates reduced effectiveness under high-load conditions and prioritizes energy efficiency at the expense of load balancing.

Moreover, in a unified architecture, most algorithms individually optimize performance measures, power consumption, or both, excluding the multiple objective parameters. In response, we introduce HGWWO, the union of the exploration power in GWO and the exploitation power in WOA. The new approach guarantees better load balancing performance by simultaneously minimizing makespan, energy consumption, and load deviation.

III. Problem Formulation

VMs are the primary entities for executing user tasks and managing workload distribution in cloud computing environments. Each VM has distinct resource capacities such as memory, processing power, disk space, and bandwidth. Let us define the set of VMs as $V = \{v_1, v_2, \ldots, v_m\}$, where m denotes the total number of virtual instances available in the system. Each VM can be modeled using a multi-dimensional vector that captures the required levels of different resources.

The infrastructure comprises several identical physical servers, with one designated as the master controller that delegates incoming task requests to appropriate VMs. VM creation is conditional on memory availability. A host lacking adequate memory cannot spawn a new VM instance. In this scenario, we assume there are n independent tasks to be executed across m available VMs.

Efficient task scheduling is critical to ensure equitable load distribution and to enhance performance metrics. Each user request is characterized by a parameter vector $U = (\rho, \sigma, \gamma, \mu, \chi)$, where ρ stands for the average frequency of online user requests, σ is the size of each request, γ denotes the required CPU resources, μ signifies the needed memory for task execution, and χ refers to the request frequency per minute. The memory load for VM *i* is given by Eq. (1).

$$\mathcal{M}_i = \text{Rem}_i + \frac{\omega_i}{\Omega_i} \times 100\% \tag{1}$$

Where Rem_i is the residual memory before assignment, ω_i is the task memory usage, and Ω_i is the total available memory.

The CPU load for VM *i* is calculated using Eq. (2).

$$\mathcal{C}_i = \text{Cpu}_i + \frac{\tau_i}{\Theta_i} \times 100\% \tag{2}$$

Where Cpu_i is the CPU capacity left before execution, τ_i is the CPU demand of tasks, and Θ_i is the total CPU available.

The overall workload on VM *i* is the weighted combination of memory and CPU load, as expressed in Eq. (3).

$$\Phi_i = \alpha \cdot \mathcal{M}_i + \beta \cdot \mathcal{C}_i \tag{3}$$

Where $\alpha + \beta = 1$ are the memory and CPU weight factors.

The total load on host *j* is derived by summing all VM loads on the host, as shown in Eq. (4).

$$\Psi_j = \sum_{i=1}^{m_j} \Phi_i = \sum_{i=1}^{m_j} \left(\alpha \cdot \mathcal{M}_{ij} + \beta \cdot \mathcal{C}_{ij}\right) \tag{4}$$

The average load across all physical machines in the network is calculated using Eq. (5).

$$\bar{\Psi} = \frac{1}{p} \sum_{j=1}^{p} \Psi_j = \frac{1}{p} \sum_{j=1}^{p} \sum_{i=1}^{m_j} \left(\alpha \cdot \mathcal{M}_{ij} + \beta \cdot \mathcal{C}_{ij}\right) \tag{5}$$

The first fitness objective, aiming to minimize the load deviation across hosts, is formulated in Eq. (6).

$$F_1 = \sum_{j=1}^{p} \left|\Psi_j - \bar{\Psi}\right| \tag{6}$$

The task assignment is represented by the binary variable defined in Eq. (7).

$$\delta_{ij} = \begin{cases} 1, & if\ task\ i\ is\ assigned\ to\ VM_j \\ 0, & otherwise \end{cases} \tag{7}$$

The total execution time on VM j is computed as per Eq. (8).

$$\mathcal{T}_j = \sum_{i=1}^{n} \delta_{ij} \cdot \mathcal{T}_{ij} \tag{8}$$

The completion time for task i on VM j is evaluated using Eq. (9).

$$\mathcal{T}_{ij} = \frac{L_i}{\text{Proc}_j} \tag{9}$$

Where L_i is the task length in MIPS, and Proc_j is the processing speed of the VM.

The makespan represents the maximum execution time among all VMs and is defined in Eq. (10).

$$\mathcal{M}_s = \max(\mathcal{T}_j), \quad 1 \le j \le m \tag{10}$$

The second objective function, total energy consumption, accounts for both active and idle states of VMs and is formulated in Eq. (11).

$$F_2 = \sum_{j=1}^{m} \left[\mathcal{T}_j \cdot \alpha_j + \left(\mathcal{M}_s - \mathcal{T}_j\right) \cdot \beta_j\right] \cdot \text{Proc}_\text{j} \tag{11}$$

Where α_j and β_j are the energy coefficients (joules per instruction) in active and idle states, respectively.

The third fitness objective considers task instruction overhead and delay penalty. It is denoted as F 3 F 3 and defined in Eq. (12).

$$F_3 = w_1 \cdot \frac{\text{Instr}_i}{\text{MaxInstr}} + w_2 \cdot D_i \tag{12}$$

Where Instr_i is the instruction count of task i, MaxInstr is the maximum throughput of any processor, and D_i is the delay penalty. The weights are fixed as $w_1 = 0.7$ and $w_2 = 0.3$.

The final goal is to minimize the overall fitness function F, a weighted sum of the three sub-objectives. This is mathematically given by Eq. (13).

$$F = \lambda_1 \cdot F_1 + \lambda_2 \cdot F_2 + \lambda_3 \cdot F_3 \quad (13)$$

where $\lambda_1 = 1$, $\lambda_2 = 0.5$, and $\lambda_3 = 0.5$ are the respective weights for load deviation, energy consumption, and task allocation cost.

IV. Proposed Method

A new hybrid meta-heuristic algorithm, HGWWO, is presented here to solve the complex multi-objective load balancing problem in cloud computing, reducing makespan, energy usage, and load deviation. HGWWO combines the best features of two popular optimization algorithms: GWO and WOA. The HGWWO is explicitly formulated to optimize task-to-VM allocation in dynamic cloud environments to improve convergence accuracy and prevent premature stagnation.

A. Grey Wolf Phase: Resource-aware Encircling Strategy

GWO mimics the leadership-driven hunting and encircling behavior observed in natural grey wolf packs. In the context of this study, where the primary goal is to allocate independent cloud computing tasks to VMs efficiently, each "grey wolf" represents a potential solution, a vector encoding task-to-VM assignments. The metaphorical "prey" is the optimal solution that minimizes makespan, balances resource utilization, and reduces energy consumption.

Wolves are hierarchically classified into four roles, Alpha (α) that represents the best-known task scheduling solution at the current iteration; beta (β) and delta (δ) that represent the second and third-best solutions, contributing diversity and assisting in refining the search direction; and omega (ω) as follower solutions that are guided by the leading trio to explore the broader search space.

The optimization begins with the encircling behavior, where wolves adjust their positions relative to the prey, i.e., the current best-known scheduling pattern. Each VM allocation solution is iteratively updated using the encircling formulation, which introduces stochastic coefficients to balance intensification and diversification. The position update rule is given by Eq. (14).

$$\overrightarrow{X_{\text{new}}}(t+1) = \overrightarrow{X_{\text{prey}}}(t) - \vec{A} \cdot \vec{D} \quad \text{where} \quad \vec{D} = \left|\vec{C} \cdot \overrightarrow{X_{\text{prey}}}(t) - \vec{X}(t)\right| \quad (14)$$

Where $\overrightarrow{X_{\text{prey}}}(t)$ is the position vector of the current best task scheduling solution, $\vec{X}(t)$ denotes the position of the candidate VM allocation, $\vec{D}$ refers to the encircling vector (distance between candidate and prey), $\vec{A}$ is an adaptive coefficient vector controlling convergence intensity, $\vec{C}$ is a coefficient vector introducing randomness. The adaptive coefficient $\vec{A}$ is derived as follows:

$$\vec{A} = 2 \cdot \vec{a} \cdot \overrightarrow{r_1} - \vec{a} \quad (15)$$

Where $\vec{a}$ linearly decreases from 2 to 0 across iterations, gradually shifting the search from exploration to exploitation, and $\overrightarrow{r_1}$ is a random vector in [0, 1]. The vector $\vec{C}$ is similarly calculated using Eq. (16).

$$\vec{C} = 2 \cdot \overrightarrow{r_2} \quad (16)$$

With $\overrightarrow{r_2}$ being another random vector in [0, 1]. These stochastic variables help diversify the search paths and prevent premature convergence.

Once the encircling step is complete, the algorithm enters the position updating phase, where each solution is recalibrated by referencing the top three leaders (*α*, *β*, and *δ*). This phase ensures the search agents (candidate solutions) are drawn toward the most balanced and energy-efficient task assignments identified. The distances from each leader are computed, and new intermediate positions are derived as follows:

$$\begin{aligned} \overrightarrow{X_\alpha} &= \overrightarrow{X_\alpha} - \overrightarrow{A_1} \cdot \left|\overrightarrow{C_1} \cdot \overrightarrow{X_\alpha} - \vec{X}\right| \\ \overrightarrow{X_\beta} &= \overrightarrow{X_\beta} - \overrightarrow{A_2} \cdot \left|\overrightarrow{C_2} \cdot \overrightarrow{X_\beta} - \vec{X}\right| \\ \overrightarrow{X_\delta} &= \overrightarrow{X_\delta} - \overrightarrow{A_3} \cdot \left|\overrightarrow{C_3} \cdot \overrightarrow{X_\delta} - \vec{X}\right| \end{aligned} \quad (17)$$

The final updated position for the VM scheduling solution is obtained by averaging the guidance from the three leaders, as defined in Eq. (18).

$$\vec{X}(t+1) = \frac{1}{3}\left(\overrightarrow{X_\alpha} + \overrightarrow{X_\beta} + \overrightarrow{X_\delta}\right) \quad (18)$$

This rule ensures that each task-to-VM mapping benefits from the consensus of the top-performing strategies. In the load balancing context, this corresponds to scheduling tasks such that the CPU and memory demands are evenly distributed, minimizing execution bottlenecks and energy overhead.

B. Whale Phase: Exploitation Via Spiral Search

While the grey wolf phase provides deep global exploration and informed leader-based direction for task delegation, it falls short when dealing with local exploitation, especially in convergence towards remarkably optimized task-VM mapping. The proposed HGWWO incorporates an enhanced local search mechanism adopted from the WOA to increase search depth and overcome stagnation.

WOA simulates humpback whale bubble-net feeding behavior. In the analogy, there is one candidate task schedule for each whale, and the target is the ideal VM assignment with the shortest makespan and evenly distributed CPU/memory usage. WOA alternates between encircling, spiral motion exploitation, and random search, exploring and exploiting the solution space more efficiently.

The first behavioral pattern of WOA involves encircling the best solution discovered so far. In the cloud computing, this action refers to refining task allocations around the best-known task-VM mapping. This behavior mirrors the GWO encircling mechanism but contributes to localized convergence. This step is mathematically represented by Eq. (19).

$$\overrightarrow{X_{\text{new}}}(t+1) = \overrightarrow{X_{\text{best}}}(t) - \vec{A} \cdot \vec{D} \quad \text{where} \quad \vec{D} = \left|\vec{C} \cdot \overrightarrow{X_{\text{best}}}(t) - \vec{X}(t)\right| \quad (19)$$

Where $\overrightarrow{X_{\text{best}}}(t)$ refers to the current best task assignment solution and $\vec{X}(t)$ stands for the position of the current whale (solution).

To fine-tune the best solutions near optima, whales perform a spiral movement, which enables candidate solutions to follow a logarithmic path toward the best allocation found. This is crucial for minimizing minor inefficiencies in task scheduling, such as slight VM overloading or imbalance in memory usage. This phase probabilistically alternates with the encircling step using a control parameter $p \in [0,1]$. The spiral search is applied if $p \geq 0.5$, and the position update is calculated by Eq. (20).

$$\overrightarrow{X_{\text{new}}}(t+1) = \left|\overrightarrow{X_{\text{best}}}(t) - \vec{X}(t)\right| \cdot e^{bl} \cdot \cos(2\pi l) + \overrightarrow{X_{\text{best}}}(t) \tag{20}$$

Where b is a constant defining the spiral shape, l is a random number in $[-1,1]$ determining movement orientation.

WOA includes a prey search mechanism randomly generating new task scheduling solutions to maintain diversity and avoid local optima traps. This phase becomes dominant when the algorithm needs to escape plateaus in the fitness landscape. The exploration behavior is modeled by Eq. (21).

$$\overrightarrow{X_{\text{new}}}(t+1) = \overrightarrow{X_{\text{rand}}} - \vec{A} \cdot \vec{D} \quad \text{where} \quad \vec{D} = \left|\vec{C} \cdot \overrightarrow{X_{\text{rand}}} - \vec{X}(t)\right| \tag{21}$$

Where $\overrightarrow{X_{\text{rand}}}$ is a randomly selected solution from the population. This component is especially important for diversifying the task assignments and discovering new combinations that reduce energy consumption or execution delay.

In the hybrid HGWWO algorithm, the whale phase is used during the optimization cycle's exploitation phase. Following the population being navigated into fruitful areas in the search area by the GWO, random and spiral updates in WOA refine task allocations, offsetting near-optimal load oversights or inefficiencies that the GWO might overlook. Dynamic load redistribution among underloaded VMs, slight reduction in makespan, and progressive energy-based assignment refinement result from adaptive switching between random exploration and spiral refinement.

C. Combined Update Strategy for Load Balancing

HGWWO integrates the exploratory efficiency of the GWO with the local refinement capability of the WOA to solve the multi-objective load balancing problem in cloud computing. Fig. 1 shows the flowchart of the proposed algorithm. In typical cloud environments, scheduling tasks across multiple heterogeneous VMs must account for CPU utilization, memory demand, execution time, and energy cost. Algorithms that only explore broadly (like GWO) may converge slowly or stagnate near suboptimal allocations. Conversely, algorithms like WOA, which are good at refining solutions, may not escape poor initial guesses. HGWWO resolves these challenges by dynamically adapting its search strategy at each iteration, blending the strengths of both methods.

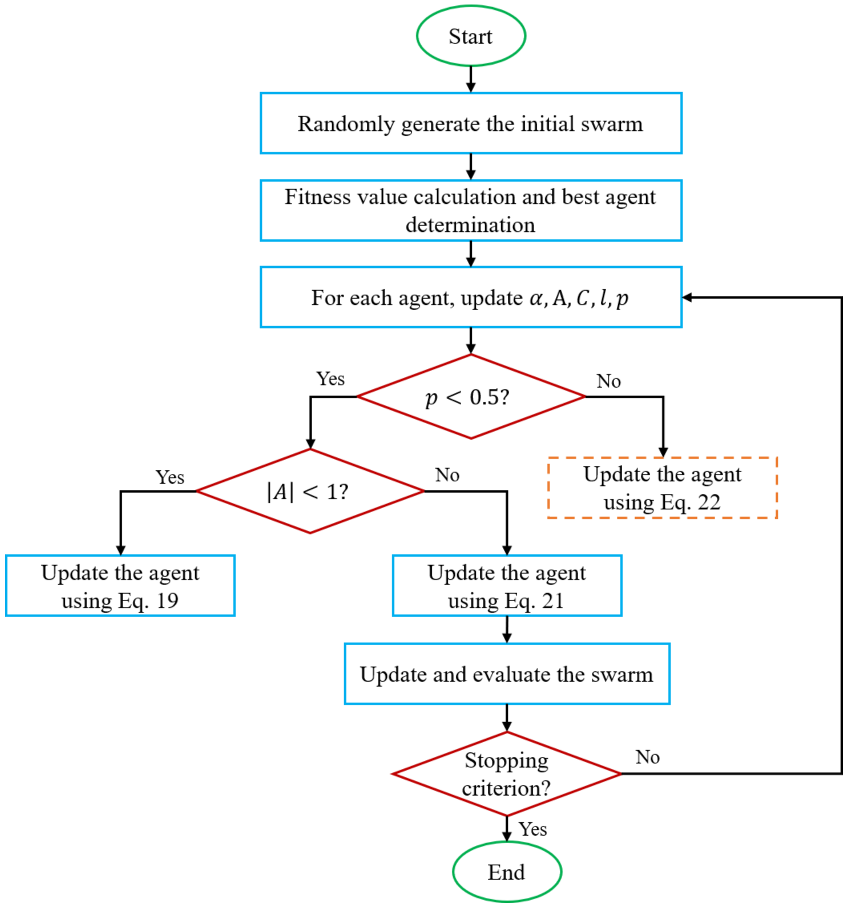
Start
Randomly generate the initial swarm
Fitness value calculation and best agent determination
For each agent, update α, A, C, l, p
$p < 0.5?$
Yes
No
$|A| < 1?$
Yes
No
Update the agent using Eq. 22
Update the agent using Eq. 19
Update the agent using Eq. 21
Update and evaluate the swarm
Stopping criterion?
No
Yes
End

Fig. 1. Flowchart of proposed algorithm.

At every iteration of the optimization process, each candidate solution (task allocation vector) chooses its update mechanism based on a random probability $p \in [0,1]$. If $p < 0.5$, the solution is updated using a GWO-style encircling behavior, otherwise the solution undergoes WOA-style spiral exploitation. This probabilistic strategy allows HGWWO to dynamically balance global and local search throughout the optimization timeline. It avoids premature convergence (via random exploration) and accelerates convergence near optimal points (via spiral tightening). The hybrid update rule is mathematically defined in Eq. (22).

$$\overrightarrow{X_{HGWWO}}(t+1) = \begin{cases} min\left(\overrightarrow{X_{prey}} - \vec{A}\cdot\vec{D}, \overrightarrow{X_{elite}}(t)\right), \\ \quad if\ p < 0.5 \\ min\left(\left|\overrightarrow{X_{prey}} - \vec{X}(t)\right| \cdot e^{bl} \cdot cos(2\pi l)\right) + \\ \overrightarrow{X_{prey}}, \overrightarrow{X_{elite}}(t), \quad if\ p \geq 0.5 \end{cases} \tag{22}$$

Where $\overrightarrow{X_{elite}}(t)$ denotes the best composite solution generated by the GWO leadership strategy.

The elite solution is calculated as a weighted average of the top three solutions identified by GWO: alpha (best), beta (second best), and delta (third best). These roles correspond to the most balanced, energy-efficient task assignments in the current population, given by Eq. (23).

$$\overrightarrow{X_{\text{elite}}}(t) = \frac{1}{3}\left(\overrightarrow{X_\alpha} + \overrightarrow{X_\beta} + \overrightarrow{X_\delta}\right) \tag{23}$$

Each of these leader positions is computed by subtracting a scaled distance from their current state, as described below:

$$\begin{aligned} \overrightarrow{X_\alpha} &= \overrightarrow{X_\alpha} - \overrightarrow{A_1} \cdot \left|\overrightarrow{C_1} \cdot \overrightarrow{X_\alpha} - \vec{X}(t)\right| \\ \overrightarrow{X_\beta} &= \overrightarrow{X_\beta} - \overrightarrow{A_2} \cdot \left|\overrightarrow{C_2} \cdot \overrightarrow{X_\beta} - \vec{X}(t)\right| \\ \overrightarrow{X_\delta} &= \overrightarrow{X_\delta} - \overrightarrow{A_3} \cdot \left|\overrightarrow{C_3} \cdot \overrightarrow{X_\delta} - \vec{X}(t)\right| \end{aligned} \tag{24}$$

Each updated solution is then evaluated against the multi-objective fitness function. This function combines the load deviation across hosts, the energy consumption of active and idle VMs, and the delay and task overhead due to scheduling inefficiencies.

The proposed HGWWO initiates with a diverse population of candidate solutions, each representing a potential task-to-VM mapping configuration in a cloud computing environment. These initial solutions are randomly generated to promote exploration across the search space. The optimization process proceeds through successive iterations, during which each candidate is iteratively refined based on either a randomly chosen peer or the current global best.

A linearly decreasing control parameter, denoted by α, governs the balance between exploration and exploitation. Specifically, α starts at a maximum value of 2 and reduces to 0 throughout iterations. This parameter is used to compute the adaptive coefficient vector $\vec{A}$, which determines the intensity of position updates. When the magnitude $\vec{A} > 1$, the algorithm emphasizes exploration by referencing a randomly selected solution. Conversely, when $\vec{A} < 1$, the emphasis shifts toward exploitation, refining solutions around the current optimum.

Simultaneously, the algorithm incorporates a probabilistic decision model based on a random number p to alternate between encircling and spiral movements for solution updates. The update mechanism is bifurcated into two cases, GWO-style refinement and WOA-inspired spiral exploitation.

When $p < 0.5$, the solution update follows a leader-guided search strategy, combining the difference between the best-known task allocation and the current candidate, and the elite average position. When $p \geq 0.5$, the algorithm adopts a spiral search path influenced by the Whale Optimization concept. This update models the tightening bubble-net motion that localises optimal solutions through logarithmic spirals. The position is recalculated by incorporating the proximity to the best-known solution and a decaying spiral motion.

V. Results

To assess the efficacy of the proposed HGWWO in optimizing task scheduling and load balancing in cloud environments, comprehensive simulations were conducted using CloudSim 3.0.3 on a Windows 7 machine equipped with an Intel Core i7 processor, 8 GB RAM, and a 3.4 GHz CPU. The experimental setup is described in Table II.

TABLE II. Simulation Configuration and System Specifications

Component	Quantity	Attributes	Specification
Hosts	20	CPU cores	6
		Bandwidth	15 GB/s
		RAM	16.0 GB
		Hypervisor	Xen
		Storage volume	4.0 TB
		Computational power	177,730 MIPS
Virtual machines	10-100	Hypervisor	Xen
		Processing elements	1
		Disk image size	10 GB
		Bandwidth	1 GB/s
		RAM	0.5 GB
		Processing capability	9726 MIPS
Data center	1	Storage allocation cost (per unit)	0.001
		Memory allocation cost (per unit)	0.05
		Base usage cost	3.0 units
		Virtualization platform	Xen
		Operating system	Linux
		Architecture type	x86

The performance of HGWWO was compared against five prominent algorithms, ACO, GWO, WOA, ABC, and HHO, on a large scale of tasks and VM configurations. The comparison was made based on multi-objective measures such as makespan, energy usage, resource usage, migration rate, imbalance factor, throughput, and idle time.

Two principal scenarios were experimented with, varying the tasks (100 to 2000) while the number of VMs (100) remains fixed, and fixing the tasks (1000) while varying the number of VMs (10 to 100). The fitness function, defined in Eq. (13), integrates load deviation, energy consumption, and task cost. Optimal weight values $\lambda_1 = 1$, $\lambda_2 = 0.5$, and $\lambda_3 = 0.5$ were obtained through trial-and-error and are detailed in Table III.

TABLE III. IMPACT OF FITNESS WEIGHTS ON LOAD BALANCING METRICS

Weights			**Simulation outcomes**			
λ_1	λ_2	λ_3	**Energy usage (kJ)**	**Load variance**	**Throughput (req/ms)**	**Makespan (ms)**
1	0.9	0.9	305.08	0.355	9.05	6570.31
		0.8	294.62	0.349	8.51	6352.19
		0.7	290.29	0.338	8.29	9105.88
		0.6	283.84	0.374	8.05	8995.02
		0.5	279.43	0.361	7.88	8906.53
	0.8	0.9	245.66	0.351	8.59	9389.11
		0.8	240.38	0.342	8.40	9313.76
		0.7	224.17	0.318	8.17	9109.31
		0.6	221.02	0.285	7.94	8976.19
		0.5	211.27	0.253	7.77	8911.42
	0.7	0.9	291.18	0.331	8.42	9102.35
		0.8	282.65	0.296	8.19	9085.72
		0.7	283.45	0.304	7.76	9106.79
		0.6	276.38	0.294	7.60	8987.46
		0.5	266.18	0.232	7.55	8519.17
	0.6	0.9	261.09	0.275	8.18	9015.41
		0.8	256.21	0.256	8.01	8722.15
		0.7	251.60	0.247	7.79	8613.91
		0.6	246.52	0.219	7.25	8482.31
		0.5	242.96	0.203	7.01	8314.20
	0.5	0.9	235.15	0.255	7.29	8818.73
		0.8	229.23	0.161	7.05	8416.29
		0.7	217.26	0.094	6.64	8093.19
		0.6	209.41	0.079	5.73	7914.46
		0.5	**170.77**	**0.063**	**5.39**	**7751.62**
	0.4	0.9	251.63	0.351	7.74	8386.94
		0.8	242.10	0.285	7.26	8352.28
		0.7	221.48	0.244	6.93	8109.12
		0.6	204.19	0.206	6.75	8059.26
		0.5	183.69	0.186	6.41	7903.13

Fig. 2 and Fig. 3 show that HGWWO has the lowest energy consumption among all rival algorithms. This results from its innovative adaptive hybrid architecture, which inhibits unnecessary task migration and accurately assigns workloads to idle VMs. It maintains efficient CPU and memory usage while minimizing excessive idle energy consumption by integrating GWO's leadership-based global exploration mechanism with WOA's local spiral exploitation.

Fig. 4 and Fig. 5 indicate that HGWWO beats its competitors in make-span reduction, a key contributor to system responsiveness and QoS. The reduction is a result of spreading out global exploration and switching to local optimization inherent in HGWWO, which facilitates faster convergence on optimum task allocations. The effect is a more deterministic response time with less latency, resulting in increased user service availability. The load standard deviation between the VMs (as depicted in Fig. 6) quantifies the uniformity in the distribution of resources. HGWWO maintains the lowest deviation, notably after t=3700 ms, reflecting stable and efficient load balancing. HGWWO's leader-based update approach, in which alpha, beta, and delta wolves collectively affect each candidate's position, and spiral movement adjusts the load in finer-grained steps, minimizes oscillations between overloaded and under-loaded nodes.

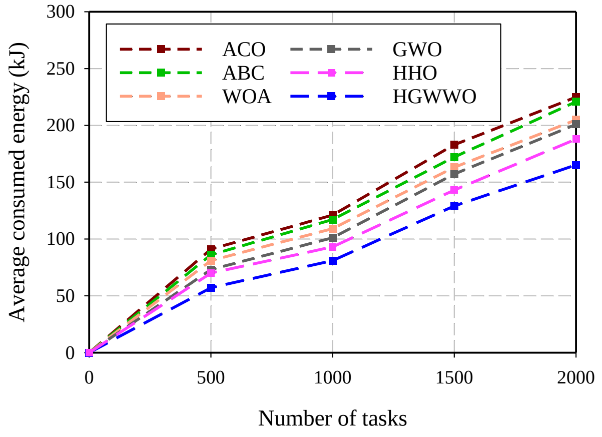

Fig. 2. Energy consumption comparison with increasing task volume at constant VM count.

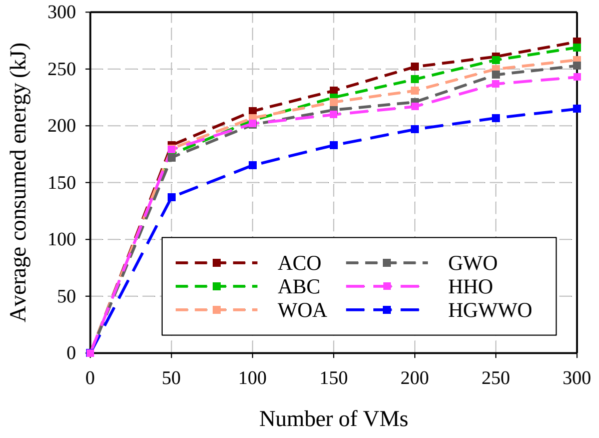

Fig. 3. Energy consumption comparison with increasing VM count at constant task volume.

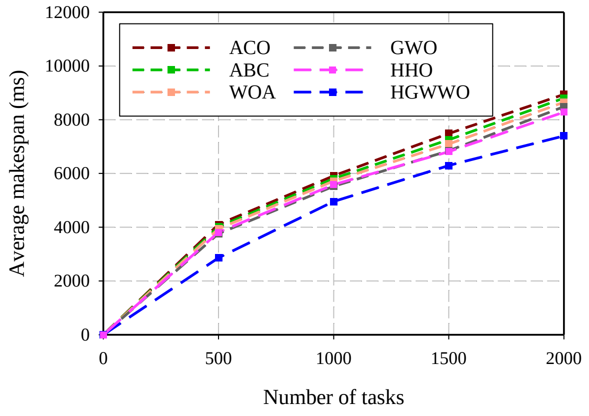

Fig. 4. Makespan comparison with increasing task volume at constant VM count.

As evident from Fig. 7, HGWWO migrates fewer tasks than its competitors. This is the result of its strategy for allocation based on fitness, which predicts upcoming changes in load ahead of time and assigns tasks to the appropriate VMs in advance, resulting in fewer runtime reassignments and reduced overhead. Fig. 7 shows that HGWWO achieves higher throughput for most conditions, particularly under heavy load conditions. The locations' proximity-based update policies, along with spiral exploitation policies in the algorithm, enable timely job completion with constant service continuity, thus enhancing the cloud system's ability to service user demands.

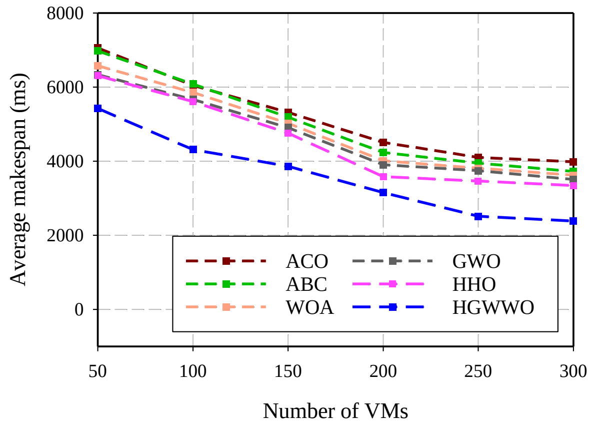

Fig. 5. Makespan comparison with increasing VM count at constant task volume.

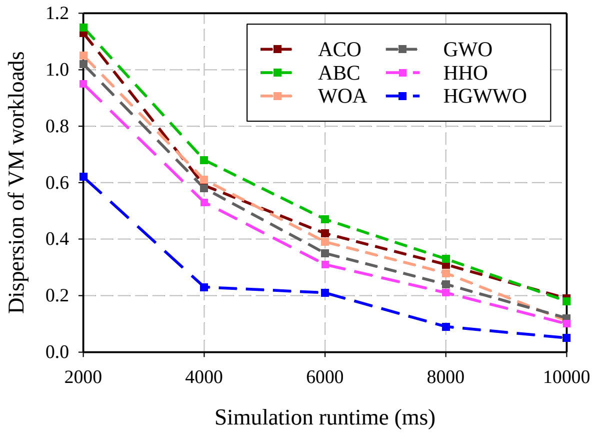

Fig. 6. Temporal variation of load imbalance levels.

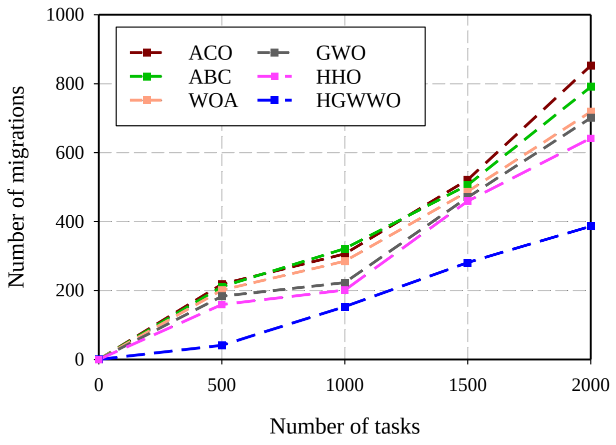

Fig. 7. Task migration volume comparison.

VI. Discussion

Experimental findings reveal that our proposed HGWWO algorithm produces significant advancements in dynamic load balancing for cloud computing infrastructures. The hybridization of GWO and WOA exhibits a higher convergence trend, a reduced value for makespan, reduced energy consumption, and an equalized task allotment between virtual machines. These advantages result from several primary design attributes of the proposed HGWWO algorithm.

Firstly, incorporating GWO's hierarchical structure for leadership with WOA's exploitation strategy based on a spiral allows HGWWO to obtain a more stable tradeoff between local exploration and global exploitation, thereby reducing the risk of early convergence, which is prone to single standalone meta-heuristics such as GWO, WOA, and ABC. Secondly, adaptive behavior is achieved by incorporating auto-tuning algorithmic parameters into a variable workload distribution based on its state. This is needed in heterogeneous clouds due to sudden changes in task intensity alongside available resources.

Additionally, the multi-objective optimization model employed in this work contributes to these enhancements. As it optimizes makespan, energy consumption, and load deviation simultaneously, HGWWO handles the practical tradeoffs that cloud service providers face. This end-to-end optimization results in more efficient resource utilization and a higher level of QoS compared to alternative algorithms.

However, our proposed model also has several weaknesses that need to be addressed. One of these principles is that, due to its multi-objective optimization with a hybrid structure, the algorithm's computational complexity is higher than that of simpler, single-heuristic schemes. Although this tradeoff allows for superior performance, it may not be suitable for extremely large-scale implementations or applications that depend highly on latency and for which speed of execution is a concern. In addition, validation was conducted using synthetic workloads within the CloudSim simulation platform. Nonetheless, a widely adopted benchmarking platform, validation within realistic real-world infrastructures (e.g., OpenStack, Kubernetes) would be more convincing in practical terms.

It also has the limitation that it requires manual setting of algorithm parameters, including weight coefficients and control factors. Although they were tuned experimentally, an adaptive or self-tuning system would be suitable for further enhancing the robustness of HGWWO when environmental conditions vary. Lastly, security, fault tolerance, or energy-aware geographic distribution are not explicitly addressed by the current model, but become significant in specific scenarios of cloud deployment.

VII. Conclusion

This study introduced the HGWWO algorithm for the rigorous problem of dynamic load balancing and task scheduling in cloud computing. HGWWO was compared with five algorithms, ACO, GWO, WOA, HHO, and ABC, along with significant performance metrics such as makespan, energy usage, the standard deviation, throughput, task migration, the degree of imbalance, and idle time, based on comprehensive simulations under different workloads and VM settings using CloudSim 3.0.3, and further validated using a physical cloud testbed. Experimental results showed that HGWWO outperforms all baseline algorithms in balanced energy-aware and resource allocation. Notably, it achieved a smaller makespan, less energy consumption, and fewer task migrations while maintaining high throughput with even load distribution.

Compared with other meta-heuristics and hybrid solutions, our proposed HGWWO algorithm possesses several practical merits. It converges faster due to leveraging GWO's initial global exploration and WOA's terminal local exploitation. This adaptive switching also minimizes task migration and improves throughput, particularly for heterogeneous or load-intensive environments. It also distinguishes itself from deep-learning-based models, which require extensive training datasets and produce significant overheads. Our proposed HGWWO is thus lightweight and usable even in resource-constrained cloud nodes, including real-time scheduling. Its merits qualify it as a viable contender for cloud production use, particularly where task patterns change dynamically, and deterministic scheduling is infeasible.

Although the suggested HGWWO algorithm demonstrates promising behavior for fluctuating and varying cloud conditions, several weaknesses must be considered. First, even though HGWWO converges faster than standalone GWO or WOA, its hybridization also incurs more computational overhead, which may be noticeable for ultra-large scales. Secondly, the algorithm's behavior is now evaluated under synthetic workloads and simulated environments. Realistic deployment environments with non-deterministic workload spikes and hardware faults may require further fine-tuning or complementing with learning-based adaptive modules. This limitation presents opportunities for enhancing HGWWO with real-time or predictive feedback mechanisms in future work.

Based on the positive outcome by HGWWO, future work will focus on expanding the algorithm to distributed and federated cloud environments such as multi-cloud computing, edge computing, and fog computing. They outline more demanding scenarios, such as latency-aware task assignment, limited resource availability, and asynchronous message passing, which can be tailored for handling by HGWWO. Combining workload prediction with deep reinforcement learning will also enable proactive resource allotment and improve scalability in real-time. Further enhancements would include adaptive parameter self-tuning methodologies for reducing manual weight modifications in the fitness function. Others would include security-aware policy-making for scheduling purposes, along with carbon-aware optimization for eco-friendly cloud control. Long-term verification will consist of practical deployment and benchmarking on running cloud infrastructures (e.g., OpenStack) for demonstrating robustness under realistic constraint scenarios.

ACKNOWLEDGEMENT

This work was supported by the project of Science and Technology Program of Henan Province (252102320051).

REFERENCES

[1] T. Khan, W. Tian, G. Zhou, S. Ilager, M. Gong, and R. Buyya, "Machine learning (ML)-centric resource management in cloud computing: A review and future directions," Journal of Network and Computer Applications, vol. 204, p. 103405, 2022.

[2] 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," Concurrency and Computation: Practice and Experience, vol. 34, no. 5, p. e6698, 2022, doi: https://doi.org/10.1002/cpe.6698.

[3] J. K. Samriya, S. Kumar, M. Kumar, H. Wu, and S. S. Gill, "Machine learning based network intrusion detection optimization for cloud computing environments," IEEE Transactions on Consumer Electronics, 2024.

[4] N. Devi et al., "A systematic literature review for load balancing and task scheduling techniques in cloud computing," Artificial Intelligence Review, vol. 57, no. 10, p. 276, 2024.

[5] B. Pourghebleh and V. Hayyolalam, "A comprehensive and systematic review of the load balancing mechanisms in the Internet of Things," Cluster Computing, vol. 23, no. 2, pp. 641-661, 2020.

[6] D. A. Shafiq, N. Z. Jhanjhi, A. Abdullah, and M. A. Alzain, "A load balancing algorithm for the data centres to optimize cloud computing applications," Ieee Access, vol. 9, pp. 41731-41744, 2021.

[7] A. R. Khan, "Dynamic load balancing in cloud computing: optimized RL-based clustering with multi-objective optimized task scheduling," Processes, vol. 12, no. 3, p. 519, 2024.

[8] M. B. Bagherabad, E. Rivandi, and M. J. Mehr, "Machine Learning for Analyzing Effects of Various Factors on Business Economic," Authorea Preprints, 2025, doi: https://doi.org/10.36227/techrxiv.174429010.09842200/v1.

[9] M. Ahmadi et al., "Optimal allocation of EVs parking lots and DG in micro grid using two - stage GA - PSO," The Journal of Engineering, vol. 2023, no. 2, p. e12237, 2023, doi: https://doi.org/10.1049/tje2.12237.

[10] S. M. Sanjari, A. M. Shibli, M. Mia, M. Gupta, and M. M. A. Pritom, "SmishViz: Towards A Graph-based Visualization System for Monitoring and Characterizing Ongoing Smishing Threats," in Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy, 2024, pp. 257-268, doi: https://doi.org/10.1145/3714393.3726499.

[11] 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 Computing, vol. 24, no. 3, pp. 2673-2696, 2021, doi: https://doi.org/10.1007/s10586-021-03294-4.

[12] S. N. Makhadmeh et al., "Recent advances in Grey Wolf Optimizer, its versions and applications," Ieee Access, vol. 12, pp. 22991-23028, 2023.

[13] S. Mirjalili and A. Lewis, "The whale optimization algorithm," Advances in engineering software, vol. 95, pp. 51-67, 2016.

[14] A. Muteeh, M. Sardaraz, and M. Tahir, "MrLBA: multi-resource load balancing algorithm for cloud computing using ant colony optimization," Cluster Computing, vol. 24, no. 4, pp. 3135-3145, 2021.

[15] S. Sefati, M. Mousavinasab, and R. Zareh Farkhady, "Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: performance evaluation," The Journal of Supercomputing, vol. 78, no. 1, pp. 18-42, 2022.

[16] K. Ramya and S. Ayothi, "Hybrid dingo and whale optimization algorithm - based optimal load balancing for cloud computing environment," Transactions on Emerging Telecommunications Technologies, vol. 34, no. 5, p. e4760, 2023.

[17] P. Geetha, S. Vivekanandan, R. Yogitha, and M. Jeyalakshmi, "Optimal load balancing in cloud: Introduction to hybrid optimization algorithm," Expert Systems with Applications, vol. 237, p. 121450, 2024.

[18] F. A. Emara, A. A. Gad-Elrab, A. Sobhi, A. S. Alsharkawy, M. E. Embabi, and M. Abd El-Baky, "Multi-objective task scheduling algorithm for load balancing in cloud computing based on improved Harris hawks optimization," The Journal of Supercomputing, vol. 81, no. 6, pp. 1-38, 2025.

[19] M. Haris and S. Zubair, "Battle Royale deep reinforcement learning algorithm for effective load balancing in cloud computing," Cluster Computing, vol. 28, no. 1, p. 19, 2025.

[20] S. Tabagchi Milan, N. Jafari Navimipour, H. Lohi Bavil, and S. Yalcin, "A QoS-based technique for load balancing in green cloud computing using an artificial bee colony algorithm," Journal of Experimental & Theoretical Artificial Intelligence, vol. 37, no. 2, pp. 307-342, 2025.