Journal of Theoretical and Applied Information Technology
15st July 2026. Vol.104. No.13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AN INTELLIGENT HYBRID METAHEURISTIC FRAMEWORK FOR ADAPTIVE LOAD BALANCING USING GREY WOLF OPTIMIZATION AND CENTROID OPPOSITION-BASED LEARNING

P. RAJESH[1], K. MAHARAJAN[2], M. JAYALAKSHMI[3]

[1]Research Scholar, Department of Computer Science and Engineering, Kalasalingam Academy of Research and Education, Krishnankoil, Tamil Nadu, India
[2]Associate Professor, Department of Computer Science and Engineering, Kalasalingam Academy of Research and Education, Krishnankoil, Tamil Nadu, India
[3]Professor, Department of Computer Science and Engineering – Cyber Security, Ramco Institute of Technology, Tamil Nadu, India

E-mail: [1]rajeshmephd30@gmail.com, [2]maharajank@gmail.com, [3]jayalacsmi@gmail.com

ABSTRACT

The problem of task scheduling into the VMs of different capacities is one of the most important problems in the cloud computing domain. Bad scheduling leads to long makespan, unutilized servers, and violation of Service Level Agreements (SLAs). In cloud computing scheduling problem, Grey Wolf Optimization (GWO) is widely used but it suffers from premature convergence in high dimensional problems. In dynamic cloud environments, GWO gets trapped in local optima, which is one of its weaknesses that needs to be addressed. A novel algorithm called Centroid Opposition-Based Learning (COBL) is introduced, where opposite solutions are obtained by using the centroid of the current population in search space as an opposite agent rather than by taking the limits of the search space as an opposite agent. Using COBL keeps the swarm more diverse and avoids premature convergence and improves the exploitation of the solutions without compromising on the quality of results. To overcome the premature convergence and local optima trap problem of GWO, we propose Centroid Opposition-Based Learning Grey Wolf Optimization algorithm (COBL-GWO). We implement the algorithm in CloudSim and compare COBL-GWO with Improved GWO (IGWO), PSO, CPSO, GA and standard GWO algorithm for 10–50 tasks on 5 and 10 VMs. On 5 VMs it lowered the average makespan to 458.4 s against 573.6 s for GA, a 20.1% reduction, and raised average resource utilization to 0.854 against 0.422; on 10 VMs the makespan gap widened to 25.3%. But, as we report, there is a price: COBL-GWO takes about 38% more execution time and requires more memory than GA due to the opposition step that is performed during each iteration. Thus, COBL-GWO may be more suitable for the use cases where resource balancing and good resource utilization take precedence over scheduling overhead.

Keywords: *Cloud Computing, Load Balancing, Task Scheduling, Grey Wolf Optimization, Centroid Opposition-Based Learning, Metaheuristic Optimization, Resource Utilization; Energy-Efficient Computing, Sustainable Digital Infrastructure.*

1. INTRODUCTION

Cloud computing has evolved the way of procurement and utilization of IT resources by organizations. It refers to an on demand provision and access to computing resources (including compute resources, storage resources, network resources and platform services) in a pooled manner [1]. As the usage of cloud continues to grow, one of the operational challenges that clouds must resolve is the scheduling of tasks; that is, given a set of tasks, and a set of available machines with varied capabilities, where the machines should the tasks be scheduled. Scheduling of tasks is also termed as the Load Balancing. It is the heart of cost, reliability and user perceived latency for a cloud provider. If the scheduling is done effectively, then none of the virtual machines will be over-burdened, nor will it have any underutilized virtual machine. Otherwise, it can lead to poor user satisfaction in terms of poor performance, energy inefficiency due to

underutilization, and the risk of violating Service Level Agreements (SLA) [2].

Conventional scheduling strategies, such as Round Robin, Least Connections, and Min-Min are simple and straightforward, however, they have the limitation that they work under the assumption of the system is static and not dynamic [3]. In practical real-life cloud environments, virtual machines have different performance and capacity features; tasks have dynamic arrival rates and the load profile changes on a minute-to-minute basis. In the presence of such complexities, deterministic algorithms are not capable of adjusting to the dynamic conditions and thus their performance suffers. For the large and dynamic clouds, they offer poor throughput. That is why metaheuristic techniques, which are the search algorithms inspired from the evolutionary process, physical and biological phenomena [4], such as, Ant Colony Optimization, Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Genetic Algorithm (GA) are the preferred method of cloud task scheduling. They help in traversing over large and complicated search spaces and can jump from the local optima to better global optima.

Among the meta-heuristic techniques, Grey Wolf Optimization (GWO) [5] is found to be one of the highly promising and popular techniques. GWO is an optimization technique inspired by the social leadership hierarchy and hunting behavior of the Grey wolves. It maintains three best solutions and is known as alpha, beta and delta while the remaining solutions are treated as Omega. GWO is widely used because it has few parameters, and thus it is quite easy to control the balance between exploration and exploitation capabilities. However, the drawback with GWO is its premature convergence that leads it to get stuck in the local optima. It occurs frequently in the problems with high dimensions, nonlinearity, multimodality, and dynamic changes, as it is in case of cloud scheduling. Once the GWO has reached to the local optima, then the whole population converges to the best leaders found so far; thus, it reduces the population diversity and is unable to jump to find the better optimal solutions.

Opposition-based learning (OBL) is a very simple learning method that is able to enhance the performance of algorithms [6]. OBL involves evaluating the opposite candidate of a given random candidate for every iteration, then the one with better fitness value is picked for the next iteration. With the OBL method, both the estimation and its corresponding opposite are considered at a given moment; thus it can cover more areas of the search space and speed up convergence. In the literature, the concept of centroid opposition-based learning (COBL) [7] has been proposed. Unlike OBL, the opposite candidate is not determined with respect to the limits of the search space, but rather the opposite is generated based on the center of the current solutions in the population; thus, the opposite solution always stays in good direction of the search space as defined by the population. COBL can be embedded with any population-based optimization method like GA, PSO, and GWO without disturbing the fundamental mechanism of these optimization techniques. Here, it helps GWO to jump out of the premature convergence.

In this work, we propose a hybrid of GWO with centroid opposition-based learning to develop a novel method for cloud task scheduling called Centroid Opposition-Based Grey Wolf Optimization (COBL-GWO). The proposed COBL-GWO method is used to determine the optimal task scheduling in cloud environment. COBL-GWO is used to improve the population diversity at initialisation phase and search phase in order to enhance the exploration phase of GWO. The proposed scheduler is developed with the primary objective to minimize the makespan while achieving good resource utilization, and also the cost of scheduling in terms of execution time and memory cost should be within the acceptable range.

The fundamental problem addressed in this paper is the task-to-VM scheduling problem in heterogeneous cloud environments, where the scheduler must simultaneously minimize the makespan and maximize resource utilization, subject to the constraint that the computation of the schedule itself must remain within an acceptable time and memory budget. Existing methods either suffer from premature convergence (GWO-based algorithms) or achieve good convergence at the cost of high parameter complexity and computational overhead (hybrid and learning-based methods). No existing method resolves GWO's premature convergence with a single lightweight operator while simultaneously measuring and reporting the cost of doing so. The present work addresses this gap by proposing COBL-GWO and explicitly quantifying both the quality gains and their computational cost.

Since, in our experimental environment, the tasks are offline and non-preemptive, the tasks are independent, the target system is heterogeneous, and we have considered only a single data center, we have limited the scope of this work to offline non-preemptive independent task scheduling on heterogeneous VMs in a single data center. In the future, we would like to extend our experimental work to consider online task arrivals, tasks with dependencies, multiple clouds, and real-life deployment for a practical evaluation of COBL-GWO.

A preliminary version of this research appeared at the 9th International Conference on Inventive Systems and Control (ICISC 2025) [33], in which we outlined the combination of grey wolf optimizer (GWO) with chaotic opposition-based learning (COBL) and provided initial makespan comparisons. We greatly expand on this earlier study in the present paper: It adds the full mathematical formulation of the scheduling model, a much larger comparison (six algorithms across two VM configurations and five task loads), four performance metrics rather than one, an explicit treatment of the cost side of the trade-off, a threats-to-validity analysis, and a discussion of where the method should and should not be used. No data from the conference paper is reused without being re-run and re-reported here.

The work is organized around three research questions:

1. **RQ1.** Does centroid opposition-based learning measurably reduce the makespan and improve the resource utilization of GWO-based cloud scheduling, and by how much relative to GWO, IGWO, CPSO, PSO, and GA?
2. **RQ2.** What does the diversity injected by COBL cost in execution time and memory, and is that cost stable as the task load and the number of VMs grow?
3. **RQ3.** For which classes of cloud workload is COBL-GWO the right choice, and for which would a cheaper scheduler be preferable?

The specific contributions are:

- A task-to-VM scheduling model for heterogeneous clouds that combines expected execution time, transfer time, and VM-state constraints into a single fitness function for makespan and utilization.
- COBL-GWO, which uses centroid opposition both to initialize the population and to refresh it during the GWO search, so diversity is maintained without weakening exploitation.
- A controlled experiment in CloudSim for workloads of 10–50 tasks and 5/10 VMs, comparing COBL-GWO against five baseline algorithms across four metrics — the most comprehensive evaluation reported for this problem class.
- A practical cost–benefit analysis that quantifies the gain in makespan and utilization against the additional computational and memory demands, and that specifies the workload classes for which the trade-off is advantageous.

We introduce a controlled experiment in CloudSim for workloads of 10–50 tasks and 5/10 VMs, comparing GWO–COBL against five baseline algorithms across four metrics: makespan, resource utilization, task execution time, and memory consumption, including a discussion of when GWO–COBL becomes costlier than these baselines. We translate the experimental outcomes into a practical cost–benefit analysis, quantifying the gain in makespan and utilization against the additional computational and memory demands, and outlining the class of workloads for which the tradeoff is advantageous.

The remainder of this paper is organized as follows: Section 2 surveys existing metaheuristic-based load balancers and pinpoints the specific gap addressed in this work. Section 3 presents the formal problem definition. Section 4 describes the GWO and COBL algorithms as well as their integration in COBL–GWO. Section 5 details the simulation environment, experimental protocol, and findings. Section 6 summarizes our key contributions and discusses the strengths, limitations, and validity threats of this study. Section 7 concludes the paper, while Section 8 points toward future research directions.

2. RELATED WORK

Because load balancing determines how efficiently a cloud system utilizes its available resources, this area has drawn many metaheuristic and hybrid approaches. We group the most recent proposals here based on the limitation that inspired our work, and then clarify the gap that we address.

2.1 Comparative And Single-Method Studies

Using make span, response time, and resource utilization as evaluation criteria, Zhou et al. [2] compared several metaheuristic load balancers and reported that PSO performed well across all three indicators. Their research is useful as a baseline comparison, but since they simply compare existing heuristics rather than introducing a better algorithm, the fundamental convergence challenge is not directly addressed. Murshedi et al. [20] employed a resource allocation matrix based on the ant colony algorithm that is resource friendly but lacks sufficient optimization depth. Similarly, Du and Wang [28] presented an adaptive dynamic load balancer to improve throughput without employing a hybrid metaheuristic. Both studies indicate that a standalone metaheuristic requires further tuning to effectively address dynamic workloads in cloud environments.

2.2 Hybrid Metaheuristics

Many recent approaches combine two or more metaheuristics in an attempt to increase efficiency. Kumar and Rajesh [3] developed an adaptive algorithm combining the global elite optimization (GEO) with comprehensive learning-based particle swarm optimization (CMBO) and named it MCGEO for multi-objective load balancing that exhibits better convergence behavior with significantly greater computational complexity (which could hinder the scalability of this algorithm). Sumathi et al. [4] developed HGAPSO (a fusion of genetic algorithm, PSO, and harmony search optimizer) and demonstrated lower response time and makespan compared with the other algorithms, yet the combination of three metaheuristics requires more parameter tuning and becomes fragile in complex scenarios. Al Reshan et al. [6] presented PSO-GWO for load balancing, noting that the combination of both metaheuristics converges faster, though the resulting hybrid method can converge prematurely in highly dynamic workloads (an issue we also target in this research). In a recent study, Geeta and Kamakshi Prasad [9] proposed DCCO (CMBO- DXO hybrid) to obtain better resource utilization; however, their approach lacks the adaptability required to effectively respond to dynamic workloads. Similar approaches are adopted by several other authors: Assudani and Balakrishnan [14] in the design of IPSO-GA, Sumathi et al. [13] in HHO-ACO, Gabhane et al. [23] in ACO-Tabu search, Lilhore et al. [24] in WWO-ACO, and Elsakaan and Amroun [31] in a hybrid of clustering-based approach, genetic algorithm, and round robin. These approaches show that while the hybrid approaches can improve on certain performance measures, the additional cost, parameter tuning complexity and potential problems regarding scalability need to be addressed. Overall, there is still room to improve the convergence in highly dynamic workloads.

2.3 Learning-Driven And Fuzzy Schedulers

Several recent load balancing mechanisms incorporate some form of learning to improve efficiency in different ways. Krishna and Vali [5] developed the Meta-RHDC (Metaheuristic Reinforcement-based Hierarchical Distributed Cloud) algorithm that utilizes RL techniques with deep learning models for scalable, dynamic workload allocation; however, the training overhead can be high in terms of resources and training time. Lokesh and Baseer [7] introduced a modified dragonfly algorithm and the bidirectional long short-term memory neural network to predict resource availability; their proposed hybrid algorithm achieved high throughput but suffers from the resource-intensive and time-consuming training requirements of deep learning models and additional complexity in the underlying system design. In a recent work, Ghafir et al. [11] combined double deep Q-learning algorithms with a deep generative adversarial network to create a feedback controller and a fault-tolerant load balancer. Though the fault-tolerant approach is promising, the overall complexity and overhead are high. Long et al. [10] used a PSO-based load balancing technique to achieve better performance, but the design complexity was increased by the inclusion of fuzzy decision-making. While these learning-based approaches are capable of improving various performance metrics, their adoption and practicality are constrained by the requirement of having both training data and training time available.

2.4 Energy- And Qos-Aware Swarm Methods

The third research direction in recent years focuses on balancing computational efficiency, energy consumption, and QoS. Singhal et al. [8], [15] adopted Rock Hyrax-based algorithm for energy-efficient and QoS-aware load balancing, but their experiments are limited in terms of workload size and validation in a real-time environment. Hayyolalam and Özkasap [19] combined chaos theory with a Black Widow Optimization algorithm (CBWO) for dynamic load balancing and showed very promising results in terms of makespan and energy consumption, but their chaotic-based dynamics can lead to unpredictable behavior in

different conditions.Jesi et al. (2024) [16] (Mayfly, MFO-LB), Rajashekar et al. (2023) [17] (SCEHO-IPSO), Zhanuzak et al. (2024) [18] (EDLB), Priya et al. (2025) [12] (PTBO), Ramya and Ayothi (2023) [21] (HDWOA-LBM), Kaplan and Babalik (2025) [22] (GA-PSO under DDoS), Geetha et al. (2024) [29] (IC&BA), Boroumand et al. (2025) [30] (TSO-MCR), Neelakantan and Yadav (2023) [25] (HKHW-DBNM), Nivitha et al. (2025) [26] (CBBM-WARMS), and Choudhary and Rajak (2024) [27] (modified Min-Min) each report improvements on one or two metrics. Common criticisms raised across the group are slow convergence in certain problem areas, reliance on correct predictions, sensitivity to parameters, and only being tested using simulations. Table 1 gives a summary of this comparison.

Table 1: Comparative analysis of recent metaheuristic load-balancing approaches

Ref.	Method	Key contribution	Merits	Limitation addressed here
[2]	PSO (comparative)	Benchmarks metaheuristics on QoS	Strong on makespan, RT	No new method; convergence untouched
[3]	MCGEO (GEO+CMBO)	Multi-objective balancing	High throughput	High computational cost
[4]	HGAPSO (GA+PSO+HHO)	Triple hybrid scheduler	Lower RT and makespan	Fragile parameter tuning
[5]	Meta-RHDC (RL+DL)	RL-driven dynamic balancing	Scalable, good utilization	Heavy training cost
[6]	GWO-PSO	Faster global convergence	Reduced response time	Premature convergence remains
[7]	Dragonfly+ABi-LSTM	Predictive balancing	High throughput	DL complexity
[9]	DCCO (CMBO+DXO)	Server-load reduction	Good utilization	Weak adaptivity
[11]	PSO+DDQ+GAN	SLA-aware scheduling	Fault tolerant	Very high cost
[16]	MFO-LB (Mayfly)	Swarm balancing	Lower RT, makespan	Slow convergence
[17]	SCEHO-IPSO	Better exploration	Avoids local optima	Added complexity
[19]	CBWO (Chaos+BWO)	Energy-efficient hybrid	Strong gains	Possible instability
[24]	WWO-ACO	Hybrid scheduling	Lower cost/energy	High compute cost
[31]	Cluster+GA+RR	Multi-layer scheduler	Scalable	High overhead

2.5 Research Gap

Three problems persist across this literature. First, the strongest hybrids and learning-based schedulers are expensive, which limits their use where scheduling must be cheap and frequent. Second, premature convergence and local optima are still reported for GWO-based and several swarm methods, so the diversity problem is not solved. Third, almost every study is validated only in simulation, and few quantify the cost of the gains they claim – the compute and memory price is usually left out. COBL-GWO is aimed squarely at the second problem: it adds population diversity through centroid opposition to delay convergence, while keeping the method light enough to run in a standard CloudSim setup. We also take the third problem seriously by reporting execution time and memory alongside makespan and utilization, including the configurations where our method is the more expensive option.

3. PROBLEM FORMULATION

The scheduler must map each incoming task to exactly one VM so that the overall finishing time is short and the machines are kept busy without overloading any of them. Consider a data center with x tasks $T = \{T_1, T_2, \ldots, T_x\}$ and y heterogeneous virtual machines $VM = \{VM_1, VM_2, \ldots, VM_y\}$, with $x > y$ so that machines genuinely share load. Each task has a length in million instructions (MI); each VM has a processing speed in million instructions per second (MIPS). The expected execution time of task i on VM j follows from its length and that machine's speed:

$$EET_{i,j} = \frac{L_i}{S_j} \quad (1)$$

When several tasks share a VM, the combined file size depends on the individual sizes and their dependency factor:

$$T_{\{total\}} = T_{\{individual\}} \times T_{\{dependency\}} \quad (2)$$

Moving a task between machines incurs a transfer that depends on the link speed $TT_s(i, j)$. The transfer time and the resulting communication time are:

$$TT_{\{t\}(i,j)} = \frac{T_{\{total\}}}{TT_{\{s\}(i,j)}} \quad (3)$$

$$CT_{i,j=\frac{T_{total}}{TT_t(i,j)}} \quad (4)$$

The total time a task spends on a VM is the sum of its execution, transfer, and communication times, gated by a binary decision variable $DV_{i,j}$ that is 1 when task i is assigned to VM j and 0 otherwise:

$$ET_i = (EET_{i,j} + TT_s + CT_{i,j}).DV_{i,j} \quad (5)$$

Makespan is the time the last task finishes, i.e. the maximum completion time over all tasks. A smaller makespan means more tasks clear the data center per unit time, so it tracks throughput:

$$MS = \max\{ET_i\}, \quad i = 1 \ldots . x \quad (6)$$

Resource utilization measures how much of the available machine time is actually used. It is the total work done divided by the makespan times the number of machines, so a value near 1 means little idle capacity:

$$R_u = \frac{\sum_i ET_i}{MS * y} \quad (7)$$

The scheduler minimizes a single fitness function that rewards a short makespan and high utilization and penalizes waiting time WT, with weights w_1, w_2, w_3 set so the makespan term dominates:

$$F = w_i MS + w_2(1 - R_u) + w_3 WT \quad (8)$$

3.1 VM Availability And State

A VM is labelled by how loaded it is. Its load VML_j is the aggregate demand of the tasks assigned to it relative to its capacity, where capacity is set by available processing speed, memory, CPU, and bandwidth. A machine is underloaded when its usage is below 25% of capacity, overloaded above 80%, and balanced in between. When a machine crosses into the overloaded band, tasks are migrated from it to an underloaded machine until the system returns to a balanced state. Tasks are admitted in order of arrival, and among ready tasks the shortest is placed first, which keeps short jobs from queuing behind long ones.

4. PROPOSED METHODOLOGY

4.1 Grey Wolf Optimization

GWO models the leadership and hunting of grey wolves [32]. The pack is split into alpha (α), beta (β), delta (δ), and omega (ω) wolves. The three fittest solutions are the α, β, and δ; the remaining ω wolves reposition relative to them each iteration. Encircling the prey is modelled by

$$\vec{D} = \left|\vec{C}.\overrightarrow{X_p}(t) - \vec{X}(t)\right| \quad (9)$$

$$\vec{X}(t+1) = \vec{X}p(t) - \vec{A}.\vec{D} \quad (10)$$

where t is the current iteration, X_p is the prey (best) position, X is a wolf's position, and A and C are coefficient vectors:

$$\vec{A} = 2\vec{a}.\vec{r}_1 - \vec{a}, \quad \vec{C} = 2\vec{r}_2 \quad (11)$$

Here a decreases linearly from 2 to 0 over the run, and r_1, r_2 are random vectors in [0, 1]. Because α, β, and δ hold the best estimates of where the prey is, each wolf measures its distance to all three and updates toward their average:

$$D_\alpha = |C_1 X_\alpha - X|, D_\beta = |C_2 X_\beta - X|, D_\delta = |C_3 X_\delta - X| \quad (12)$$

$$X_1 = X_\alpha - A_1 D_\alpha, X_2 = X_\beta - A_2 D_\beta, X_3 = X_\delta - A_3 D_\delta \quad (13)$$

$$X(t+1) = \frac{X_1 + X_2 + X_3}{3} \quad (14)$$

As a decays, the search shifts from exploration to exploitation. The flaw is that once the pack clusters around the current α–β–δ it loses the spread needed to escape a local optimum, which is the behaviour COBL is meant to counter.

Algorithm 1: GWO

1. Initialize a random population of task-to-VM assignments.
2. Evaluate fitness (Eq. 8) for every wolf.
3. Record the best three solutions as Xα, Xβ, Xδ.
4. For each wolf: update a, A, C; compute Dα, Dβ, Dδ; update position by Eq. 14; re-evaluate fitness.
5. Update Xα, Xβ, Xδ if improved; repeat until convergence.
6. Return the best assignment.

4.2 Centroid Opposition-Based Learning

Opposition-based learning was introduced to machine learning in the mid-2000s and has since been folded into many metaheuristics [33-survey]. For a value x in [a, b], its opposite is

$$\check{x} = a + b - x \quad (15)$$

Plain OBL reflects through the bounds of the search space, which ignores where the population actually sits. COBL instead reflects through the population centroid. For N candidates $X = \{X_1, \ldots, X_n\}$ in D dimensions, the centroid of dimension j and the centroid-opposite of candidate i are

$$M_j = \frac{1}{N}\sum_{i=1}^{N} X_{i,j} \quad (16)$$

$$\check{X}_i = 2M - X_i \quad (17)$$

For each candidate the algorithm keeps whichever of X_i and $\check{X}_i$ has the better fitness:

$$X_i \leftarrow \arg\min\left\{F(X_i), F\left(\check{X}_i\right)\right\} \quad (18)$$

Reflecting through the centroid keeps the opposite points near the active region of the search rather than at the far corners of the domain, so they are more often useful and less often wasted. This distinguishes COBL from standard OBL: boundary-based reflection (Eq. 15) generates candidates that may lie far from the current population, wasting fitness evaluations. Centroid-based reflection (Eq. 17) ensures the opposite candidate lies on the opposite side of where the

population currently resides, keeping it within a relevant search region. This property is what allows COBL to maintain diversity without compromising the exploitation quality of GWO. COBL has not previously been applied to cloud task scheduling in this form. Figure 1 shows the idea: a candidate is mirrored through the population centroid to produce its opposite, and the fitter of the two is retained.

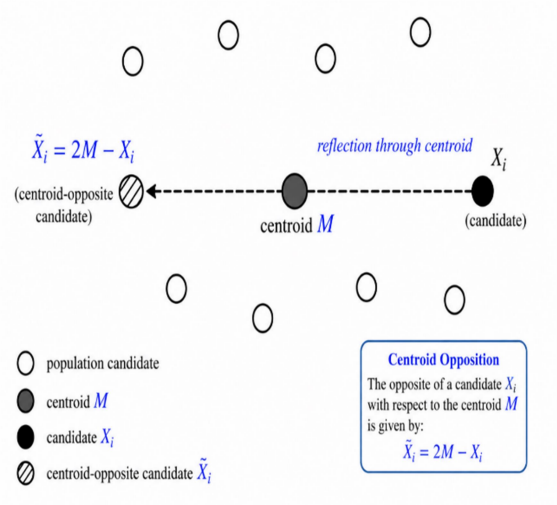

Figure 1: Centroid opposition: a candidate X_i is reflected through the population centroid M to its opposite $\tilde{X}_i = 2M - X_i$; the fitter of the pair survives.

Algorithm 2: COBL

1. Initialize the population of candidate solutions.
2. Compute the population centroid (Eq. 16).
3. Generate the centroid-opposite of each candidate (Eq. 17).
4. Evaluate fitness of both the original and the opposite solution.
5. Keep the fitter of the two for each candidate (Eq. 18).
6. Return the refreshed population.

4.3 The COBL-GWO scheduler

COBL-GWO runs GWO as the main search and uses COBL twice: once to build a diverse initial population, and again inside the loop to refresh diversity before the α–β–δ update. A candidate solution is a complete task-to-VM mapping; its fitness is Eq. 8. At each iteration the wolves move toward the average of the three leaders (Eq. 14), then COBL generates centroid-opposite mappings and the fitter of each pair survives. The opposition step represents the only addition to standard GWO, and this is also what contributes to the extra per-iteration cost we give in Section 5 -- every opposite candidate has to be made and evaluated. The reasoning behind this decision is this: we accept higher per-iteration costs in return for a population that does not prematurely collapse, since what we care about on the scheduling instances in question is the quality of the placement, rather than the time a single optimization process will take. Figure 2 shows load balancing framework which shows the flow of tasks through a scheduler into a virtual machine pool with state monitoring; Figure 3 gives the algorithmic flow, and Table 2 summarizes the parameters of the simulation.

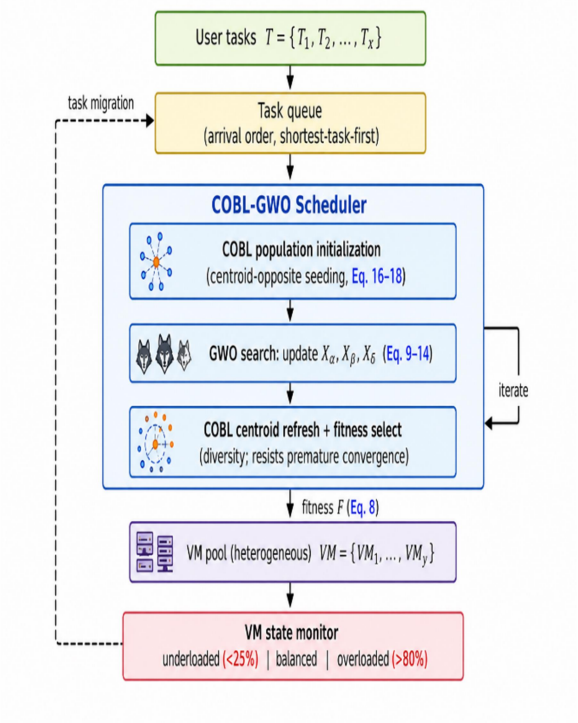

Figure 2 : COBL-GWO load balancing framework. Tasks are queued, scheduled by the COBL-GWO engine, and placed on heterogeneous VMs; a state monitor triggers migration when a machine becomes overloaded.

Table 2. Simulation parameters.

Entity	Parameter	Value
Task	Task range	10 – 50
Task	Length (MI)	1000 – 6000
Task	File size	500
VM	VM range	5-10
VM	Speed (MIPS)	225 – 300
VM	Memory	256 – 512
VM	CPU	1 – 5
VM	Bandwidth	1000
VM	VMM	XEN

Journal of Theoretical and Applied Information Technology
15st July 2026. Vol.104. No.13

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Algorithm 3: COBL-GWO for load balancing

1. Compute EET, transfer time, and communication time for all task–VM pairs (Eqs. 1–4).
2. Compute current utilization, load, and capacity for each VM.
3. Check load-balancing feasibility and VM availability.
4. Initialize a random population of task-to-VM mappings.
5. Generate the centroid-opposite of each mapping and keep the fitter initial solution.
6. Evaluate fitness (Eq. 8); record Xα, Xβ, Xδ.
7. For each candidate: update a, A, C; compute Dα, Dβ, Dδ; update position (Eq. 14); apply COBL to refresh the population; keep the fitter of original and opposite; re-evaluate fitness; update Xα, Xβ, Xδ.
8. Repeat until convergence.
9. Return the balanced assignment with all tasks placed.

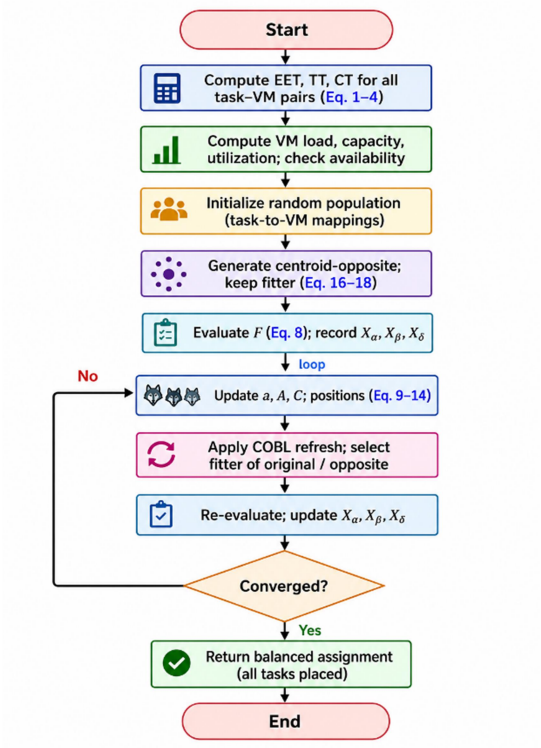

Figure 3: Flow of the COBL-GWO scheduler, with the centroid-opposition steps (shaded) added to the standard GWO loop.

5. EXPERIMENTAL SETUP AND RESULTS

COBL-GWO was implemented in CloudSim and compared against IGWO [34], GWO [35], CPSO [36], PSO [37], [38], and GA [39]. Experiments ran on an Intel Core i7-1235U at 2.30 GHz with 16 GB RAM under 64-bit Windows 11. We used two VM configurations (5 and 10) and five task loads (10, 20, 30, 40, 50), with heterogeneous VMs so that the scheduler is tested across different processing capacities. Each configuration was evaluated on four metrics: makespan, resource utilization, execution time, and memory utilization. Makespan and utilization measure scheduling quality; execution time and memory measure what that quality costs.

5.1 Makespan

COBL-GWO produced the shortest makespan at every task load and in both VM configurations. On 5 VMs its makespan rose from 207 s at 10 tasks to 710 s at 50 tasks, staying below GA throughout (362 s to 798 s). Averaged over the five loads, COBL-GWO finished in 458.4 s against GA's 573.6 s, a 20.1% reduction, and it also beat IGWO, GWO, CPSO, and PSO. For the case of 10 VMs, the order is the same but the margin is larger, with COBL-GWO averaging 293.6 s and GA 393 s -- a 25.3 % reduction in optimization time. This is because having more VMs lets diversity provided by COBL more room to find better placements, which explains why the advantage grows with the number of VMs. Results on 5 VMs are provided in Table 3 and Figure 4 report the 5-VM results.

Table 3: Makespan (s) for 5 VMs

Tasks	COBL-GWO	IGWO	GWO	CPSO	PSO	GA
10	207	271	282	294	308	362
20	306	332	345	406	422	442
30	471	482	512	530	553	566
40	598	618	630	663	691	700
50	710	721	737	754	773	798

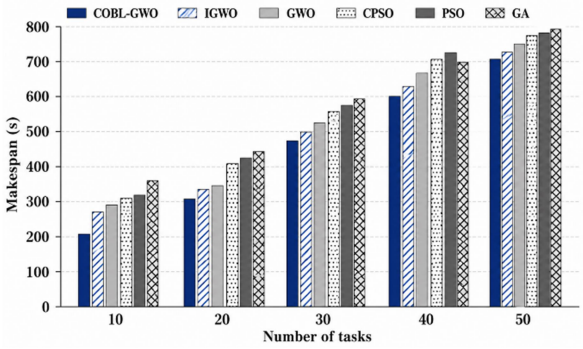

Figure 4: Makespan against task load for 5 VMs; COBL-GWO is lowest at every load

5.2 Resource Utilization

Utilization shows the same results. For 5 VMs COBL-GWO has an average utilization of 0.854, which is nearly twice the amount of GA's at 0.422. On 10 VMs the gap increases again with COBL-GWO having an average utilization of 1.12 (at the load of roughly 50 tasks) while GA's is 0.518, which means COBL-GWO continues to allocate useful work even as the tasks increase.

Table 4: Resource utilization for 5 VMs

Tasks	COBL-GWO	IGWO	GWO	CPSO	PSO	GA
10	0.74	0.57	0.47	0.41	0.39	0.32
20	0.8	0.63	0.53	0.45	0.39	0.21
30	0.87	0.74	0.62	0.54	0.47	0.38
40	0.9	0.86	0.78	0.7	0.64	0.59
50	0.96	0.88	0.8	0.75	0.7	0.61

The higher resource utilization comes as the direct consequence of finding better placements, meaning less capacity wasted and less slots left idle. Results on 5 VMs are in Table 4 and Figure 5 show the 5-VM results.

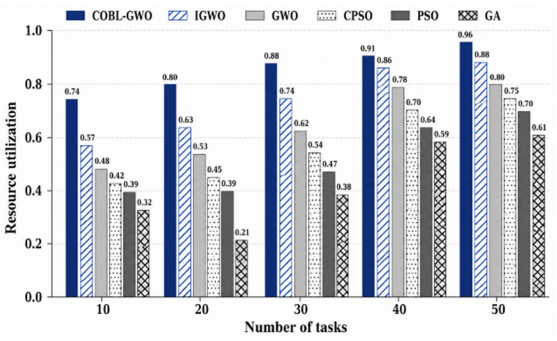

Figure 5: Resource utilization against task load for 5 VMs.

5.3 Execution Time – The Cost Side

Here the ranking inverts, and we report it openly. GA is consistently the fastest to run and COBL-GWO the slowest, because the opposition step doubles the number of candidates evaluated per iteration. On 5 VMs, COBL-GWO averaged 178.1 s against GA's 128.5 s – about 38% more. For 10 VMs the averages are 280.8 s for COBL-GWO and 215.6 s for GA. This is the cost of diversity in exchange for better placement; a more complete search will take more time. It is not free, and for latency-critical scheduling it may not be acceptable. Table 5 and Figure 6 give the 5-VM detail.

Table 5: Execution time (s) for 5 VMs

Tasks	COBL-GWO	IGWO	GWO	CPSO	PSO	GA
10	49	45	40	37	35	31
20	46	43	41	38	36	33
30	52	47	45	41	38	35
40	55	49	44	42	39	37
50	59	56	51	48	43	39

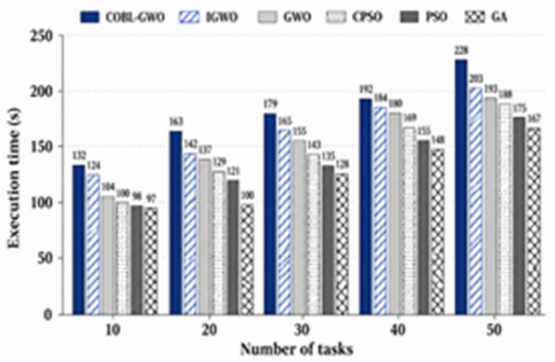

Figure 6: Execution time against task load for 5 VMs; COBL-GWO is highest.

5.4 Memory Utilization

Memory follows execution time, for the same reason: opposition-based search keeps more candidate solutions alive. On 5 VMs COBL-GWO used about 52.2 MB on average against GA's 35 MB; on 10 VMs the figures were 66.4 MB and 42 MB. The overhead, though not too large in absolute terms, is significant; it depends on the amount of work the population carries out. Results on 5 VMs are in Table 6 and Figure 7.

Table 6: Memory utilization (MB) for 5 VM

Tasks	COBL-GWO	IGWO	GWO	CPSO	PSO	GA
10	130.73	122.93	103.87	99.08	97.87	96.74
20	163.45	142.65	137.45	130.04	120.98	101.02
30	178.82	165.76	154.87	143.98	135.76	128.09
40	192.76	185.82	179.78	167.89	154.65	147.98
50	228.64	203.46	193.48	187.87	173.52	168.65

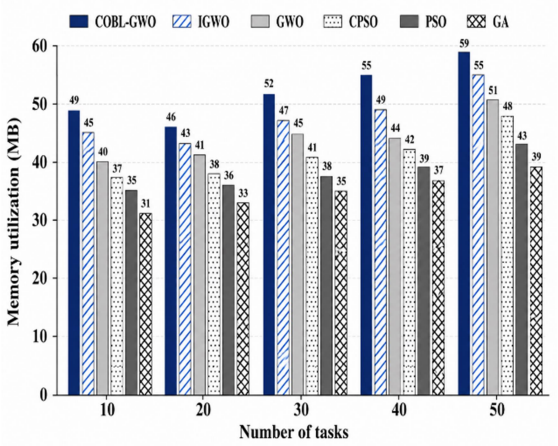

Figure 7. Memory utilization against task load for 5 VMs.

5.5 Summary Of The Trade-Off

The two halves of the picture are evident. COBL-GWO clearly wins on the metrics that matter in terms of quality of scheduling – makespan and utilization – the advantage becoming bigger as task load and VM count increase.

Table 7: Averaged COBL-GWO vs GA across both VM configurations

Metric	COBL-GWO	GA	Gain / cost
Makespan, 5 VM (s)	458.4	573.6	20.1% lower
Makespan, 10 VM (s)	293.6	393.0	25.3% lower
Utilization, 5 VM	0.854	0.422	~2× higher
Utilization, 10 VM	1.12	0.518	~2× higher
Exec. time, 5 VM (s)	178.1	128.5	38.6% higher
Memory, 5 VM (MB)	52.2	35.0	49% higher

It loses on the metrics that describe the cost of computing the schedule – execution time and memory – by a stable, predictable margin. Table 7 and Figure 8 places the averaged makespan and utilization for COBL-GWO and GA side by side across both VM counts. Whether COBL-GWO is the right tool depends entirely on which side of this trade matters more for a given deployment, a question we return to in Section 6.

6. DISCUSSION

6.1 What We Add Over Prior Work

Compared to studies in Section 2, COBL-GWO has three strengths. It tackles premature convergence at its root by reflecting candidates across the population centroid (this is where GWO-PSO [6] and several hybrids of swarm-based algorithms report convergence problems). It takes its diversity from a single low-cost operator, rather than layering algorithms in the way of HGAPSO [4] or DCCO [9] (making parameter-tuning easier). And, most uncommon in this literature, it measures the cost of its gains (most studies simply quote the metric they improve on, whereas COBL-GWO reports makespan and utilization against execution time and memory, so a researcher can judge the full trade-off). In terms of a metric they share, COBL-GWO is competitive with the stronger hybrids – reducing GA's makespan by 20–25% – but uses simpler mechanics.

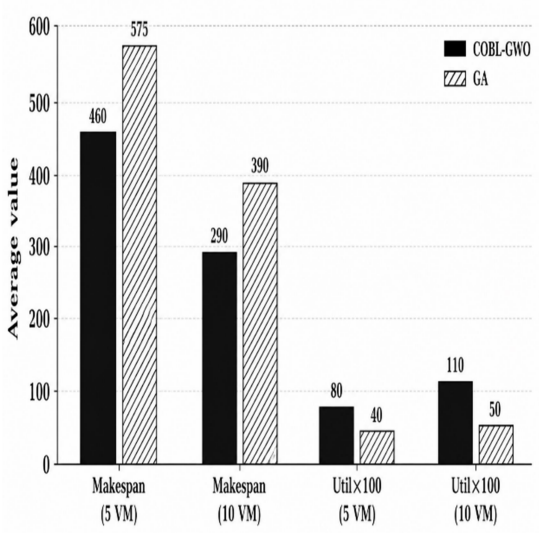

Figure 8: Averaged makespan and utilization (×100) for COBL-GWO and GA across 5 and 10 VMs.

6.2 Strengths And Weaknesses

Against the goals in Section 1, the strengths are obvious. COBL-GWO clearly addresses RQ1: it has a 20.1% (5 VM) or 25.3% (10 VM) reduction of makespan compared to GA, and a higher score on all bases. It also roughly doubles utilization. The larger the scale, the bigger the advantage, the desired direction for cloud scheduling. It is easy to implement, with one additional operator added to GWO. The weaknesses, too, are clear, addressing RQ2. Compared to GA, the opposition step costs about 38% extra in time and 50% extra in memory; this is an unavoidable cost for gaining the diversity. The work is limited to CloudSim and is not run in a real-world data centre; these are simulation results, not real measurements. We have only run

independent tasks that cannot be preempted, so we have not investigated the scheduling of dependent workloads and dynamic job arrival. While cloud scheduling research in general typically focuses on reducing energy consumption, it turns out that COBL-GWO consumes more resources per schedule run (in time and memory), so any energy savings would be at a high datacentre-level, from improving resource utilisation, rather than COBL-GWO being a computationally and energetically cheaper algorithm. We acknowledge this rather than suggest the research results would show any direct energy savings.

6.3 Threats To Validity

We use the usual internal/external/construct approach because it maps neatly onto what can go wrong in scheduling research; internal validity deals with the fairness of the comparison itself; external validity with whether the result is generalisable to other test cases; construct validity asks whether all four metrics are appropriate for measuring scheduling quality and cost. These are treated below. Internal validity. The six algorithms ran within the same CloudSim environment, on the same hardware, configured with the same task and VM parameters (Table 2). Differences come from the algorithm rather than the simulation environment. Metaheuristics are stochastic, which means that one instance of a run may suffer from noise; the consistent ordering over five different task loads and two VM-counts means that we do not believe the COBL-GWO superiority is due to a lucky seed, there is a gap as they reported variance across multiple runs.

External validity: Our results come from simulation runs with up to 50 tasks and 10 virtual machines. In the real-world cloud, clusters are much larger and noisier, and we thus expect absolute values to vary from those that we will observe in a live cloud. We, therefore, present these findings in the current paper as relative comparisons, in simulated controlled conditions, with deployment to a live cluster in the cloud reserved for future work, instead of making any claims about it.

Construct validity: It is fair to accept that Makespan and utilization are suitable metrics of scheduling performance and execution time and memory metrics to represent the cost of the algorithm, and hence all of the four together represent our objective measure of the trade-off of quality vs. cost. We do not consider response time to bursty job arrivals and SLA violation rates as part of our study metrics, which would have further strengthened our measures and are listed in our future work.

6.4 Practical Implications And SDG Alignment

COBL-GWO is recommended for: (1) Batch and analytics workloads — where a schedule is computed once and amortised over a long run, making the 38% overhead negligible relative to the placement quality gain. (2) Scientific and rendering pipelines — where makespan is the primary metric and the workload is known in advance. (3) Capacity planning — where the schedule is computed infrequently but governs resource provisioning over extended periods.

COBL-GWO is not recommended for: (1) Per-request, low-latency scheduling — where the GA is the better choice due to its 38% lower scheduling overhead. (2) Real-time or online task arrival scenarios — until the adaptive COBL trigger mechanism described in Section 8 is validated. In addition, by increasing resource utilization — approximately doubling it relative to GA — COBL-GWO reduces the number of idle but powered-on machines needed to serve a given workload. This provides a basis for alignment with SDG 9 (resilient, resource-efficient, sustainable digital infrastructure) and secondary alignment with SDG 12 (efficient use of computing resources, reduced idle capacity).

7. CONCLUSION

COBL-GWO couples Grey Wolf Optimization with centroid opposition based learning to attack one single weakness which is generally recognized to hamper the ability of the GWO optimizer to be applied for the problem of cloud scheduling task placement, namely, the problem of the algorithm getting trapped in premature convergence. The new contribution is in where the opposition learning is carried out such that the opposition of candidates in relation to the centroid of the population in turn maintains the diversity of the current population, thereby keeping it in the relevant parts of the search landscape for the active exploitation by GWO, and without needing the overhead of an additional stacked algorithm, it only introduces one additional operator and is easy to deploy for any GWO application in any search problem space. The benefit from this improvement is quantified as well.

Our COBL-GWO implementation on top of CloudSim compared to both, genetic algorithm and four other baselines, shows an improvement in average makespan in our simulations by 20.1% for 5 VMs, 25.3% for 10 VMs with the relative benefit widening for larger instances, and it roughly doubled the utilization of the resources. We do not shy away from showing the costs too, which are similarly quantified at about 38% higher execution time cost and about 50% memory cost as a consequence of the more thorough search that the new algorithm is doing. The consequence then is for a practical use of our system to suggest a more targeted application set in terms of workloads that will require this scheduler, where quality of task placement is more important than the cost of a single instance of scheduling for the workloads that is batch and analytics and capacity planning workloads, and to exclude per request workloads that require low latency in scheduling. This is a well-defined boundary set by both sides of the tradeoff that we have quantified for our new method, and that is our final contribution.

8. FUTURE RESEARCH DIRECTIONS

The constraints noted in Section 6 motivate each of these proposed research paths, meaning that this proposed research agenda arises from and corresponds with the study's aims rather than being presented as a "wish list".

- Report the variance obtained between multiple experimental runs, and perform significance testing, in order to remedy the threat to internal validity created by single-run comparisons.
- Evaluate the proposed COBL-GWO on a real or containerized compute cluster, rather than in CloudSim, to determine whether the reduction in makespan and the increase in utilization are observed in live systems.
- Explore how the proposed COBL-GWO can be adapted and applied to additional classes of cloud-workflows including dependent workflow (i.e. DAGs) and online task arrival, which were excluded in this study's scope.
- Perform a detailed and more granular analysis of data center power consumption, in order to determine the degree to which the proposed COBL-GWO actually contributes to the UN Sustainable Development Goal 7:
- Apply the COBL-GWO's opposition mechanism in an adaptive way (e.g. only when population diversity decreases below a certain threshold) in order to close the gap in execution time (the opposition mechanism's overhead) with the GA while maintaining the improvement in convergence rate.
- Apply a light-weight cloud resource utilization predictor in tandem with the proposed COBL-GWO in order to make more informed scheduling decisions based on resource utilization prediction rather than reacting to the current state.

REFERENCES:

[1] A. Jyoti, A. Yadav, D. Kumar, P. Mamoria, and V. Yadav, “Classification & technological analysis of load balancing in cloud computing,” Recent Patents on Engineering, vol. 20, no. 1, p. E18722121358630, 2026.

[2] J. Zhou et al., “Comparative analysis of metaheuristic load balancing algorithms for efficient load balancing in cloud computing,” Journal of Cloud Computing, vol. 12, no. 1, pp. 1–21, 2023.

[3] K. V. Kumar and A. Rajesh, “Multi-objective load balancing in cloud computing: a meta-heuristic approach,” Cybernetics and Systems, vol. 54, no. 8, pp. 1466–1493, 2023.

[4] M. Sumathi, S. Raja, A. Jense, A. P. S. Nelsting, and V. Swetha, “Optimal load balancing in cloud computing using hybrid meta-heuristic algorithms,” Journal of High Speed Networks, vol. 32, no. 1, pp. 3–17, 2026.

[5] M. S. R. Krishna and D. K. Vali, “Meta-RHDC: meta reinforcement learning driven hybrid lyrebird falcon optimization for dynamic load balancing in cloud computing,” IEEE Access, vol. 13, pp. 36550–36574, 2025.

[6] M. S. Al Reshan et al., “A fast converging and globally optimized approach for load balancing in cloud computing,” IEEE Access, vol. 11, pp. 11390–11404, 2023.

[7] G. Lokesh and K. Baseer, “A meta-heuristic approach-aided multi-objective strategy with optimal resource allocation via fault tolerant and priority-based scheduling for load balancing in cloud,” Wireless Networks, vol. 31, no. 3, pp. 2419–2441, 2025.

[8] S. Singhal et al., "Energy efficient load balancing algorithm for cloud computing using rock hyrax optimization," IEEE Access, vol. 12, pp. 48737–48749, 2024.
[9] K. Geeta and V. Kamakshi Prasad, "Multi-objective cloud load-balancing with hybrid optimization," International Journal of Computers and Applications, vol. 45, no. 10, pp. 611–625, 2023.
[10] G. Long, S. Wang, and C. Lv, "QoS-aware resource management in cloud computing based on fuzzy meta-heuristic method," Cluster Computing, vol. 28, no. 4, p. 276, 2025.
[11] S. Ghafir, M. A. Alam, F. Siddiqui, and S. Naaz, "Load balancing in cloud computing via intelligent PSO-based feedback controller," Sustainable Computing: Informatics and Systems, vol. 41, p. 100948, 2024.
[12] P. S. Priya, S. K. Medishetti, R. Kurupati, R. Brahmini, T. R. Gokul, and S. A. Baji, "PTBO: energy and makespan-aware scheduling in cloud computing using meta-heuristic algorithm," in 2025 6th Int. Conf. on Data Intelligence and Cognitive Informatics (ICDICI), 2025, IEEE, pp. 562–569.
[13] M. Sumathi, N. Vijayaraj, S. P. Raja, and M. Rajkamal, "HHO-ACO hybridized load balancing technique in cloud computing," International Journal of Information Technology, vol. 15, no. 3, pp. 1357–1365, 2023.
[14] P. J. Assudani and P. Balakrishnan, "An efficient approach for load balancing of VMs in cloud environment," Applied Nanoscience, vol. 13, no. 2, pp. 1313–1326, 2023.
[15] S. Singhal, N. Gupta, P. Berwal, Q. N. Naveed, A. Lasisi, and A. W. Wodajo, "Energy efficient resource allocation in cloud environment using metaheuristic algorithm," IEEE Access, vol. 11, pp. 126135–126146, 2023.
[16] M. Jesi, A. Appathurai, M. Narayanaperumal, and A. Kumar, "Load balancing in cloud computing via mayfly optimization algorithm," Revue Roumaine des Sciences Techniques – Série Électrotechnique et Énergétique, vol. 69, no. 1, pp. 79–84, 2024.
[17] K. J. Rajashekar, Channakrishnaraju, P. C. Gowda, and A. B. Jayachandra, "SCEHO-IPSO: a nature-inspired meta-heuristic optimization for task-scheduling policy in cloud computing," Applied Sciences, vol. 13, no. 19, p. 10850, 2023.
[18] R. Zhanuzak, M. A. Ala'Anzy, M. Othman, and A. Algarni, "Optimizing cloud computing performance with an enhanced dynamic load balancing algorithm for superior task allocation," IEEE Access, vol. 12, pp. 183117–183132, 2024.
[19] V. Hayyolalam and Ö. Özkasap, "CBWO: a novel multi-objective load balancing technique for cloud computing," Future Generation Computer Systems, vol. 164, p. 107561, 2025.
[20] T. A. Murshedi et al., "Optimizing cloud computing: a holistic strategy for efficient and sustainable operation through ant colony load balancing and resource optimization," International Journal of Intelligent Engineering & Systems, vol. 17, no. 5, 2024.
[21] 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.
[22] F. Kaplan and A. Babalik, "Performance analysis of cloud computing task scheduling using metaheuristic algorithms in DDoS and normal environments," Electronics, vol. 14, no. 10, p. 1988, 2025.
[23] J. P. Gabhane, S. Pathak, and N. M. Thakare, "A novel hybrid multi-resource load balancing approach using ant colony optimization with tabu search for cloud computing," Innovations in Systems and Software Engineering, vol. 19, no. 1, pp. 81–90, 2023.
[24] U. K. Lilhore et al., "A multi-objective approach to load balancing in cloud environments integrating ACO and WWO techniques," Scientific Reports, vol. 15, no. 1, p. 12036, 2025.
[25] P. Neelakantan and N. S. Yadav, "An optimized load balancing strategy for an enhancement of cloud computing environment," Wireless Personal Communications, vol. 131, no. 3, pp. 1745–1765, 2023.
[26] K. Nivitha, P. Pabitha, and R. Praveen, "CBBM-WARM: a workload-aware meta-heuristic for resource management in cloud computing," China Communications, vol. 22, no. 6, pp. 255–275, 2025.
[27] A. Choudhary and R. Rajak, "A novel strategy for deterministic workflow

scheduling with load balancing using modified min-min heuristic in cloud computing environment," Cluster Computing, vol. 27, no. 5, pp. 6985–7006, 2024.

[28] L. Du and Q. Wang, "Metaheuristic optimization for dynamic task scheduling in cloud computing environments," International Journal of Advanced Computer Science and Applications, vol. 15, no. 7, 2024.

[29] 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.

[30] A. Boroumand, M. Hosseini Shirvani, and H. Motameni, "A heuristic task scheduling algorithm in cloud computing environment: an overall cost minimization approach," Cluster Computing, vol. 28, no. 2, p. 137, 2025.

[31] N. Elsakaan and K. Amroun, "A novel multi-level hybrid load balancing and task scheduling algorithm for cloud computing environment," The Journal of Supercomputing, vol. 80, no. 9, pp. 13434–13474, 2024.

[32] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," Advances in Engineering Software, vol. 69, pp. 46–61, 2014.

[33] P. Rajesh and K. Maharajan, "An efficient load balancing scheme in cloud computing using grey wolf optimization with centroid opposition-based learning approaches," in 2025 9th Int. Conf. on Inventive Systems and Control (ICISC), 2025, IEEE, pp. 1811–1818.

[34] B. B. Aburinya, M. I. Daabo, and C. I. Nakpih, "A proposed enhanced dynamic grey wolf optimization algorithm for cloud load balancing," Journal of Electrical Systems and Information Technology, vol. 13, no. 1, p. 27, 2026.

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

[36] R. Vadivel and T. Sudalaimuthu, "Cauchy particle swarm optimization (CPSO) based migrations of tasks in a virtual machine," Wireless Personal Communications, vol. 127, no. 3, pp. 2229–2246, 2022.

[37] F. Ramezani, J. Lu, and F. K. Hussain, "Task-based system load balancing in cloud computing using particle swarm optimization," International Journal of Parallel Programming, vol. 42, no. 5, pp. 739–754, 2014.

[38] B. Mondal, "Optimization techniques for big data: a VM load balancing in cloud computing using particle swarm optimization," in Mathematical Modeling for Big Data Analytics, Elsevier, 2026, pp. 143–156.

[39] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam, "A genetic algorithm (GA) based load balancing strategy for cloud computing," Procedia Technology, vol. 10, pp. 340–347, 2013.