Rahul Kumar Sharma
Sarvpal Singh

MULTI-OBJECTIVE-OPTIMIZATION APPROACH FOR OPTIMAL TASK SCHEDULING THROUGH IN DELAY SENSITIVE CLOUD ENVIRONMENT

Abstract *This work introduces a dynamic load-balancing algorithm called CHHO (Cuckoo Harris Hawks multiobjective optimization) applied to task scheduling in cloud environment. CHHO is a new hybrid method that combines Cuckoo Search Optimization (CSO) and Harris Hawks Optimization (HHO). This combination uses the strengths of both algorithms to address the complex issues of cloud task scheduling. Specifically, CHHO uses Cuckoo Search Optimization to widen the search area of Harris Hawks Optimization, aiming to improve factors such as cost, response time, and resource use. The CHHO algorithm improves system performance by increasing VM throughput, effectively distributing workloads across VMs and maintaining a balance among task priorities through dynamic adjustments in task waiting times. To test the performance of CHHO, the algorithm is implemented in the CloudSim environment. It is compared with existing load-balancing algorithms on various performance measures. Our simulation results clearly show that CHHO performs better than existing algorithms, providing a strong and efficient solution for load balancing in cloud computing. Introducing CHHO offers a significant advancement in the field, providing a dynamic and adaptable approach that improves cloud task scheduling and enhances the overall efficiency and effectiveness of cloud computing systems.*

Keywords CHHO, CHO, HHO, load balancing, makespan

Citation Computer Science 27(1) 2026: 91–117

1. Introduction

Cloud computing has brought significant changes to information technology by changing how computing resources are provided and used. Its key features–flexibility, scalability, and cost-efficiency–have created a major shift, changing how people and organizations access and manage their computing needs. However, this transformation comes with challenges, especially the complex task of scheduling in cloud environments.

Cloud task scheduling is a complex problem that involves managing many factors to ensure the efficient use of resources and continuous satisfaction of user needs. Cloud systems must quickly adapt to changing user demands, optimizing resource use while minimizing costs. Effective task scheduling is crucial because imbalanced workloads or underused resources can lead to significant issues. Underused systems can waste power and resources, affecting the environmental and economic sustainability of cloud operations. Overloaded systems can suffer from performance problems, machine failures, and unhappy users [4, 9].

Task-load balancing on virtual machines (VMs) is a key aspect of cloud task scheduling, crucial for resource optimization, response-time management, and cost efficiency [18]. Cloud computing is essential for modern internet-based knowledge sharing, with demand for cloud services continuously growing. This demand highlights the need for efficient load balancing to keep servers running smoothly [21]. The main challenge is to distribute workloads across servers efficiently, aiming for minimal power consumption and maximum resource use.

Attaining efficient load balancing in cloud computing presents a significant challenge, necessitating the meticulous management of virtual machines (VMs) within data centers and VMs functioning as the primary processing units in cloud infrastructures, oversee resource allocation during task execution [27,31]. These interconnected VMs are designed to process tasks promptly, minimizing delays attributable to resource availability [7, 14, 15]. Given the concurrent competition for VMs' resources through multiple tasks, the necessity for robust task scheduling is evident. Effective task scheduling requires not only the equitable distribution of tasks across VMs but also the prevention of overburdening any single VM. This strategy not only ensures a balanced load but also substantially enhances the response time for task execution [16, 24]. To address these challenges, researchers have proposed various heuristic and meta-heuristic approaches tailored to both homogeneous and heterogeneous cloud environments [32]:

Resource discovery: identifying appropriate resources.

Workload migration: seamlessly transferring tasks to available resources [26].

Approaches to load balancing can be broadly categorized into static and dynamic methods [10]. Static load balancing is optimal in environments where task loads exhibit minimal variation across VMs. However, it may prove inadequate in scenarios characterized by random fluctuations in task loads. Conversely, dynamic

load balancing thrives in environments with dynamically changing loads, necessitating the continuous assessment of load data and real-time adjustments [18]. In an era marked by rapid network expansion and evolving runtime resource demands, dynamic load balancing emerges as a critical requirement, particularly within a heterogeneous resource environment.

In the field of information technology, cloud computing has become a disruptive force that is changing how computer resources are delivered and used. Its flexibility, scalability, and cost efficiency have catalyzed a seismic shift in how individuals, organizations, and enterprises access and manage their computational needs. With this paradigm shift, however, comes an array of challenges that demand innovative solutions. Chief among these challenges is the optimization of task scheduling within cloud infrastructures.

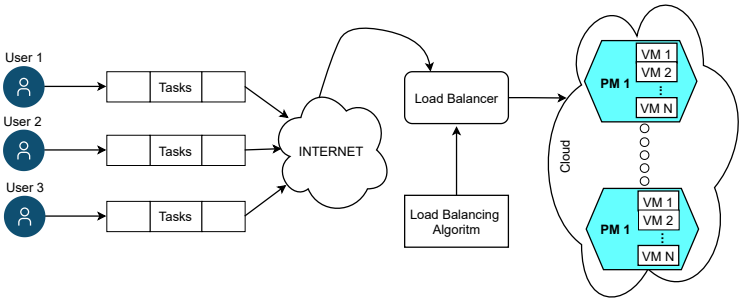

Figure 1. Task load-balance procedure in cloud computing

The optimization of cloud task scheduling is a complex operation that demands the coordination of several factors to guarantee the optimal allocation of computing resources and satisfactory service quality for users. Cloud systems operate in a dynamic environment, where the architecture must adapt to the ebb and flow of user demands while optimizing resource utilization and minimizing costs [6]. The criticality of task scheduling cannot be overstated, as the repercussions of imbalanced loads or underutilized resources extend beyond suboptimal performance.

Under-loaded systems may incur excess power consumption and waste, while overloaded systems can lead to performance degradation, machine failures, and frustrated users. Consequently, task load balancing on virtual machines (VMs) becomes a pivotal component of cloud task scheduling, poised at the intersection of resource optimization, response time management, and cost efficiency. This research paper delves into the intricacies of task scheduling in cloud computing and proposes a dynamic load-balancing algorithm, CHHO (Cuckoo Harris Hawks multi-objective optimization). The underlying premise of CHHO lies in the amalgamation of two distinctive optimization strategies: Cuckoo Search and Harris Hawks optimization [8]. The result

is a hybrid algorithm that harnesses the respective strengths of each, offering a unified and versatile approach to the multifaceted challenges of cloud task scheduling.

1.1. Key contributions

The paper presents several significant contributions. Firstly, it introduces a novel and highly efficient hybrid optimization algorithm designed to optimize load balancing within cloud networks. This innovative algorithm, CHHO (Cuckoo multi-objective Harris Hawks Optimization), tackles the dynamic load-balancing challenges inherent to cloud networks. Unlike conventional methods, CHHO leverages the power of Cuckoo optimization, replacing the MRFO procedure [29]. This facilitates the exploration space, while HHO remains employed to enhance response quality.

Notably, the developed CHHO algorithm proves to be highly effective in resolving critical constraints associated with cloud task scheduling, including resource utilization, response time, and cost. By integrating these advances, the algorithm provides a comprehensive solution for the complex and dynamic task-scheduling challenges faced in cloud networks. The newly proposed CHHO (Cuckoo Harris Hawks multi-objective Optimization) algorithm represents a groundbreaking advancement in cloud load balancing. Through a smooth integration of the Cuckoo optimization approach with the HHO algorithm, CHHO greatly improves load balancing performance metrics and convergence rates in cloud systems.

This innovative system excels in efficiently balancing workloads across virtual machines (VMs), thereby enhancing the operational efficiency of cloud networks. CHHO's unique capabilities and optimization strategies contribute to a more harmonious distribution of tasks, ultimately resulting in improved system performance and user satisfaction.

In this research paper we present a comprehensive structure that outlines our contributions and findings. Our work begins in the second section, where we delve into the existing body of knowledge and discuss related works, and provide context for our research.

In the subsequent section we lay out the problem statement, clearly articulating the challenges we seek to address.

To establish a solid foundation, the fourth section defines the architecture for improving load balancing in cloud computing.

Building on this, the fifth section formulates the problem and defines our objective functions.

The heart of our work lies in the sixth section, where we introduce our novel CHHO (Cuckoo Harris Hawks multi-objective Optimization)-based load-balancing algorithm, complete with flowcharts and pseudo code to elucidate its operation.

The seventh section elaborates on the experimental setup, utilizing the CloudSim tool.

Finally, in the eighth section, we conduct a thorough performance analysis and comparative study.

Our paper concludes in the ninth section, summarizing our findings and charting the potential future directions for this research.

2. Related works

In Thirumal, K., et al. (2023), an innovative approach known as turbulent water flow optimization (TWFO) was introduced to tackle the intricate challenges of short-term hydrothermal planning in hybrid power systems [30]. The primary objective of their research was to establish an optimal hourly generation schedule for hydrothermal systems, effectively addressing a multitude of complex constraints. The unique aspect of this approach lies in its inspiration from the whirlpool phenomenon formed by turbulent water flow in seas and oceans.

Kaur and Narang (2023) were introduced to address the complex multi-objective generation scheduling problem in integrated energy systems [17]. The technique they employed is the quantum-based Cuckoo Search algorithm (QCSA), enriched with mutation operators. Their research primarily focused on enhancing the scheduling of energy generation in systems that integrate various sources, such as combined heat and power (CHP), hydroelectric, thermal, and heat units. One of the notable advantages of this approach is its demonstrated effectiveness in improving generation scheduling, which contributes to optimizing the overall operation of integrated energy systems.

Mohammad Haris and Swaleha Zubair (2021) addressed the intricate challenge of load balancing in cloud computing [11]. Their technique, a hybrid optimization strategy, combines the Mantaray modified multi-objective Harris Hawks Optimization (MMHHO) with the Manta Ray forging optimization (MRFO) algorithm. The benefits of this hybrid approach are notable, as it results in enhanced system performance, improved virtual machine (VM) throughput, and efficient load balancing. Moreover, the algorithm ensures the sustained balance of task priorities within the cloud environment. However, it is essential to acknowledge a limitation in this method–the lack of consideration for parameters related to the dependent tasks.

In addition, Princess and Radhamani (2021) made a significant contribution to the field of load balancing in cloud computing with their innovative approach, Hybrid Harris Hawk optimization with pigeon-inspired optimization [3]. Their method exhibited an impressive efficiency rate of 97%, showcasing its potential for enhancing resource allocation in cloud environments. However, a notable limitation of this approach is its inability to provide a substantially improved level of load balance.

Jena et al. (2020) introduced a promising approach known as QMPSO, which combines an improved Q-Learning technique with a modified particle swarm optimization (PSO) method [13]. Their innovation seeks to address the intricacies of load balancing by decreasing task waiting times and enhancing critical performance metrics, including makespan, throughput, and energy consumption. This approach marks a noteworthy contribution to the field, as it addresses key challenges in load distribution within cloud environments.

Table 1
Comparative analysis of existing techniques

Citation	Technique	Advantages	Disadvantages
Thirumal, K., et al., 2023 [30]	Turbulent Water Flow Optimization (TWFO)	Optimal hourly generation schedule for hydrothermal systems, addressing complex constraints	Overlooks aspects like cost reduction
Kaur, A., Narang, N., 2023 [17]	Mutation operator system with Cuckoo Search Approach	Improved generation scheduling for multi-objective integrated energy systems	Not explicitly mentioned
Haris, M., and Zubair, S., 2021 [11]	Hybrid optimization combining HHO with MRFO	Enhanced system performance, improved VM throughput, and load balancing, and sustained task priority balance.	Not address parameters like dependent tasks.
Princess and Radhamani, 2021 [3]	Hybrid HHO with Pigeon-inspired optimization	Achieves 97% efficiency	Falls short in addressing substantial load imbalance
Jena et al., 2020 [13]	QMPSO, blending improved Q-learning with MPSO	Minimizes the amount of time that tasks must wait, and improves makespan, throughput, and load-balancing energy efficiency.	Neglects consideration of dependent tasks
Negi et al., 2021 [23]	Priority-based Load Balancing (PLB)	Includes tasks starving, resource mapping, several queuing models, reduced makespan, equivalent implementation, and optimal performance.	Lacks provisions for dynamic resource rescheduling
Adaikalaraj, 2021 [1]	Quasi Oppositional Dragonfly Algorithm	Demonstrates optimal performance	Lacks mention of specific drawbacks
Ebadifard et al., 2020 [5]	Dynamic Load Balancing Method	Enhances the average makespan	Overlooks aspects like cost reduction
Semmoud et al., 2020 [25]	AST Distributed Load Balancing Method	Considers task priority levels	Does not address dependent tasks
Junaid et al., 2020a, Junaid et al., 2020b [14] [**?**]	DFTF combining SVM and modified CSO	Presents a multifaceted approach to improving time, cost efficiency, and overall cloud environment performance	Disregards elements such as task migrations, priority-based development, and timeline constraints.
Siddiqui and Darbari., M.,2020 [28]	QPSL with First in first out (FIFO) Discipline method	Gives consumers with severe loads optimal service availability	Does not incorporate weighted priority
Li and Wu, 2019 [19]	Ant Colony Optimization for Load Balancing	Maximizes reliability, scalability, load balancing, and minimizes migrations and response time	Experiences higher response times, reduced reliability, and lower quality of service
Hou and Zhao, 2018 [12]	Fusion Strategy based on Task Scheduling	Maximizes resource utilization	Diminishes Quality of Service (QoS)
Narale and Butey, 2018 [22]	Load Balancing with Throttled Approach	Reduces data center processing time and costs but experiences increased cost, time, and response times with a rise in VMs	———
Milan et al., 2019 [20]	Nature-Inspired Metaheuristic Methods	Maximizes error tolerance, minimizes overhead, and optimizes resource utilization	Yields maximal makespan and response time

Table 1
(Continued...)

Citation	Technique	Advantages	Disadvantages
Adhikari et al., 2019 [2]	Task Deployment via Metaheuristic-Bat Approach	Mitigates load imbalance, reduces response times, and optimizes resource allocation	Exhibits lower scalability, tailored for homogeneous environments, and caters to unique tasks

Negi et al. [23] presented an innovative approach known as priority-based load balancing (PLB) for optimizing load distribution in cloud computing environments. Their method incorporated various essential components, including resource mapping, task starvation prevention, multi-queuing models, truncated makespan management, and the pursuit of an ideal performance balance. While PLB offers a holistic approach to addressing critical aspects of load balancing, it does have a notable limitation. Specifically, it does not encompass the dynamic rescheduling of resources to adapt to evolving situations within cloud environments. This aspect of adaptability is essential in cloud systems characterized by ever changing workloads and resource demands. Despite this limitation, Negi et al.'s approach is a notable contribution to the pursuit of efficient and effective load balancing in the cloud.

Adaikalaraj [1] introduced a noteworthy contribution to the field of load balancing in cloud computing through the quasi oppositional dragonfly algorithm. The proposed approach demonstrated optimal performance, yet the specific advantages and disadvantages were not explicitly mentioned in the source. Nonetheless, Adaikalaraj's work highlights the potential of novel optimization techniques to enhance the efficiency and effectiveness of load balancing in cloud environments. While further details regarding the strengths and limitations of the quasi oppositional dragonfly algorithm are not provided, the mere mention of optimal performance suggests its viability as a contender in the ongoing quest to optimize cloud resource allocation.

Ebadifard et al. [5] delved into the realm of load balancing in cloud computing by introducing a dynamic method that showcased an improvement in average makespan. Their approach aimed to enhance the efficiency of cloud operations, particularly in terms of task execution. However, it is essential to note that their method did not encompass considerations for factors like cost reduction, a critical aspect in managing cloud resources efficiently.

Semmoud et al. [25] presented a notable contribution to the field of load balancing in cloud computing through their adaptive starvation threshold-based distributed load-balancing algorithm. This innovative approach places an emphasis on considering the priority levels of tasks, ensuring that critical tasks receive the attention they require. While the algorithm effectively addresses priority-based task management, it does have a limitation–it does not account for dependent tasks.

Siddiqui and Darbari., M., [28] presented an FIFO queue discipline and QPSL queuing model with M/M/k Queue. This technique was created to give users better

service availability, especially during periods of high traffic, proving a dedication to ensuring that cloud service users have a flawless experience. While this approach excels in improving service availability, one notable limitation is its lack of consideration for weighted priority in task management. However, one important aspect that their approach did not incorporate is the consideration of virtual network functions. While their method prioritizes makespan and execution time, it might not be the ideal choice in scenarios where virtual network functions play a crucial role in cloud operations.

Li and Wu [19] introduced an innovative approach to load balancing through ant colony optimization (ACO) tailored for load balancing based on SWIM. Their work aimed to achieve maximal reliability, scalability, and load balancing while minimizing the count of migrations and response times.

Hou and Zhao [12] presented a fusion-strategy-based task-scheduling approach, which seeks to maximize resource utilization in cloud environments. While the approach, indeed, accomplishes the goal of maximum resource utilization, it does come at a tradeoff, potentially leading to reduced quality of service (QoS).

Narale and Butey (2018) [22] introduced a load balancing strategy utilizing the throttled approach. Their method aimed at reducing data-center processing time and associated costs.

Milan et al. [20] presented a way to solve load-balancing problems in cloud systems that use meta-heuristic techniques inspired by nature. By using this approach they aimed to reduce overhead, increase scalability, improve fault tolerance, and optimize resource usage.

Adhikari et al. [2] presented the task deployment approach, a fascinating load balancing method based on the meta-heuristic bat approach. Their approach focuses on reducing load imbalance in particular, with the goal of achieving faster reaction times and efficient resource usage.

3. Problem statement

The issue of load balancing emerges as a critical problem that demands meticulous attention. The arrival of tasks in cloud computing occurs randomly, often resulting in resource overload scenarios. In these instances, the utilization of CPU resources becomes sporadic, leading to inefficient resource allocation, in which some resources are heavily burdened while others remain underutilized. Traditional load-balancing techniques typically rely on allocating the entire load based on the number of virtual machines, or CPU utilization, across the complete network. While such approaches may achieve a degree of balance, they frequently overlook the crucial aspects of resource capabilities and user requirements. Existing research in load balancing within cloud computing has primarily focused on improving the quality of load distribution, often neglecting factors related to scalability and response time. These algorithms face various challenges, including potential system bottlenecks and the associated high hardware costs. However, they often fall short in considering the inherent abilities of resources to execute specific tasks in tandem with dynamic user demands.

To address these complex issues, an effective multi-objective hybrid optimization method has been designed in the cloud computing context with the goal of correcting load balancing. This novel approach seeks to optimize resource allocation while considering not only load distribution but also the intrinsic capabilities of resources and the diverse requirements of cloud users. By doing so, it endeavors to overcome the limitations of existing load-balancing strategies, ensuring both superior performance and enhanced scalability while also mitigating the implications of potential system bottlenecks and cost-intensive hardware.

4. Load-balancing methods in cloud computing

Load balancing in cloud computing represents a pivotal element in optimizing the performance and resource utilization of cloud infrastructures. As cloud environments continue to evolve and scale, the efficient distribution of computational workloads becomes increasingly critical. Load balancing refers to the equitable allocation of tasks across virtual machines (VMs) or servers to ensure that no single resource is overwhelmed while others remain underutilized. The primary goal is to deliver consistent, responsive services to users, optimize resource usage, and prevent system bottlenecks.

Cloud computing offers the promise of scalability, flexibility, and cost-efficiency, making it a compelling choice for businesses and organizations of all sizes. However, the dynamic nature of cloud systems, with fluctuating user demands and workloads, necessitates sophisticated load-balancing mechanisms. These mechanisms consider various factors, such as the capabilities of individual resources, task priorities, and performance objectives, to ensure an equitable and efficient distribution of tasks. Efficient load balancing not only improves the overall performance and response time of cloud services but also has a direct impact on cost savings. By ensuring that resources are utilized optimally, cloud providers can minimize operational expenses and energy consumption. The challenge lies in developing load-balancing algorithms that can adapt to the dynamic nature of cloud computing while simultaneously considering resource capabilities and user requirements.

In recent years, researchers have explored innovative approaches, including hybrid optimization algorithms, multi-objective techniques, and machine learning, to address the complexities of load balancing in the cloud. These approaches aim to achieve superior performance, scalability, and efficient resource utilization. As cloud computing continues to play a pivotal role in modern computing, the pursuit of effective load balancing remains a driving force behind enhanced service quality, cost reduction, and resource optimization in this ever-evolving technological landscape.

5. Load-balancing optimization methods

The CHHO algorithm introduced here combines the CSO and HHO algorithms, addressing certain drawbacks inherent in the basic HHO and CSO approaches. Both

the HHO and CSO algorithms have exhibited limitations, notably in terms of their convergence rates and computational costs.

Furthermore, load balancing in cloud computing via the HHO algorithm typically leads to faster response times. To mitigate these issues, the proposed CHHO method combines the strengths of the HHO and CSO algorithms. The CHHO algorithm, as presented, successfully achieves an efficient average load distribution while simultaneously improving critical performance metrics, including optimized resource utilization and task response times.

5.1. Harris-Hawks optimization method

An approach for optimization motivated by nature, the algorithm is based on how hawks hunt. It belongs to the class of metaheuristic algorithms used to solve complex optimization problems. HHO mimics the hunting strategy of hawks, where hawks collaborate to capture prey effectively. In the context of optimization, hawks represent potential solutions or candidate solutions, and their collaboration aims to find the best solution to a given problem.

The HHO algorithm can be mathematically described as follows:

1. Initialization: Create a starter population of hawk solutions. Within the search space of the issue, every solution is represented as a vector of decision variables.
2. Objective function: Define an objective function that quantifies the quality of a solution. The objective function is problem-specific and reflects what needs to be optimized (maximized or minimized).
3. Hunting phase: During this phase, hawks employ different strategies to explore and exploit the search space. The key equations include:
   Capture equation: HHO uses a capture equation to decide which hawks (solutions) are more likely to capture the prey (optimal solution):

$$X_{t+1} = X_t + r_1 * (X_t - X_p) - r_2 * X_{rand} \tag{1}$$

   Here, X_{t+1} is the updated solution, X_t is the current solution, X_p is the prey (best solution), X_{rand} is a randomly selected solution, andr_1 and r_2 are random coefficients.
   Exploration equation: HHO also employs an exploration equation to encourage exploration of the search space:

$$X_{expl} = X_{lb} + rand * (X_{ub} - X_{lb}) \tag{2}$$

   Here, X_{expl} is the exploration solution, X_{lb} and X_{ub} represent the lower and upper bounds of the search space, and (rand) is a random number between 0 and 1.
4. Update phase: Evaluate the objective function for each hawk solution and update the population by selecting the solutions that lead to better objective function values.

5. Termination criterion: Continue the hunting and update phases iteratively until a stopping criterion is met. This criterion can be a maximum number of iterations, a convergence threshold, or other problem-specific conditions.
6. Result: The best solution found in the final population is considered the optimal solution to the given optimization problem.

The HHO algorithm leverages the cooperative behavior of hawks in hunting to efficiently explore the solution space, strike a balance between exploration and exploitation, and ultimately converge toward the optimal solution. Numerous domains, such as engineering, finance, and machine learning, have used it to solve a wide range of optimization issues. Its adaptability and effectiveness make it a valuable tool for solving complex optimization challenges.

5.2. Basic Cuckoo Search Optimization Method

The Cuckoo Search (CS) Algorithm is a nature-inspired optimization technique that draws inspiration from the brood parasitism behavior of cuckoo birds. It is applied especially in the realm of metaheuristic algorithms to tackle complicated optimization issues. CS is designed to locate the optimal solution within the search space of an optimization problem. Here, we will delve into the CS Algorithm and present four mathematical equations that describe its key components.

1. Initialization: In CS, a population of nests is created to represent potential solutions to the optimization problem. Each nest is a point in the search space, denoted as X_i, where i ranges from 1 to the number of nests.
2. Objective function: The objective function, denoted as f(X), quantifies the quality of a solution. The aim is to either maximize or minimize this objective function, depending on the optimization problem.
3. Levy flight: One of the defining features of CS is the use of Levy flights for exploration. The Levy flight step size (Si) follows a Levy distribution and is calculated using the following equation:

$$Si = \lambda \cdot \frac{u}{|v|^{\frac{1}{3}}} \tag{3}$$

where λ is a scaling factor and u and v are random variables following a standard normal distribution.

4. Nest position update: The position of each nest is updated iteratively. The new position X_i^{new} of a nest is determined as follows:

$$X_i^{new} = X_i + S_i \cdot LevyFlight() \tag{4}$$

The $LevyFlight()$ function generates the Levy flight step.

5. Nest selection and replacement: After updating the positions, a selection mechanism is applied to determine whether the new nest is better than the existing one. If the new nest is superior it replaces the old one.

6. Abandonment and replacement: The CS algorithm also includes a mechanism in which some nests are randomly abandoned with a probability Pa. These abandoned nests are replaced with new random nests, ensuring diversity in the population.
7. Termination criterion: Until a termination requirement is satisfied, the CS algorithm iterates through the update, selection, and replacement stages. Achieving an acceptable solution quality or limiting the number of iterations is an example of common criteria.

The Cuckoo Search Algorithm excels in balancing exploration and exploitation of the solution space. Its use of Levy flights promotes effective exploration, while the selection and replacement mechanism enhance exploitation of promising solutions.

5.3. Proposed Cuckoo Harris Hawks multi-objective optimization (CHHO) algorithm

Adapting the Cuckoo Search (CS) Algorithm from single-objective optimization to multi-objective optimization (MOO) involves some modifications. Here's an outline of how CS can be extended for MOO with mathematical equations at each step:

Step 1: Initialization

Similar to single-objective CS, initialize a population of nests (Xi) in the search space, where i ranges from 1 to the number of nests.

Step 2: Objective functions

In MOO, you have multiple objective functions to optimize. We will denote them as $f1(X), f2(X), ..., fk(X)$, where c is the number of objectives.

Step 3: Levy flight for exploration

In multi-objective CS you can use Levy flights to explore the solution space. The Levy flight step size (si) follows a Levy distribution, which can be calculated as:

$$Si = \lambda \cdot \frac{u}{|v|^{(1/3)^u}} \tag{5}$$

Here, λ is a scaling factor, and u and v are random variables following a standard normal distribution.

Step 4: Position update for multi-objective

The position of each nest is updated iteratively. The new position (X_i^{new}) of a nest is determined as follows:

$$X_i^{new} = X_i + S_i \cdot LevyFlight() \tag{6}$$

Step 5: Pareto-based selection

You need to maintain a set of non-dominated solutions, often referred to as the Pareto front. The selection mechanism compares each new solution with the existing solutions in terms of Pareto dominance. Let's denote the set of non-dominated solutions as P.

Step 6: Abandonment and replacement

Similar to single-objective CS, you can include a mechanism to abandon some nests with a probability pa to maintain diversity in the population. Abandoned nests are replaced with new random nests.

Step 7: Termination criterion

Update, selection, and replacement stages are iterated through by the algorithm until a termination requirement is satisfied. One common set of requirements is to attain a suitable Pareto front or a maximum number of repetitions.

Step 8: Pareto front evaluation

Once the algorithm terminates you have a set of non-dominated solutions in P, representing the Pareto front. These solutions are optimal in the sense that no other solution in the front is better in all objectives simultaneously.

By using above the Cuckoo Search (CS) Algorithm for multi-objective optimization (MOO), the best energy efficiency and response-time-point find out for the algorithm, and this output works as an input for next part of the algorithm, which is related to HHO.

Hunting phase: During this phase, hawks employ different strategies to explore and exploit the search space. The key equations include:

A. Capture equation: HHO uses a capture equation to decide which hawks (solutions) are more likely to capture the prey (optimal solution):

$$X_{t+1} = X_t + r_1 * (X_t - X_p) - r_2 * X_{rand} \tag{7}$$

Here, X_{t+1} is the updated solution, X_t is the current solution, X_p is the prey (best solution), X_{rand} is a randomly selected solution, and r_1 and r_2 are random coefficients.

Exploration equation: HHO also employs an exploration equation to encourage exploration of the search space:

$$X_{expl} = X_{lb} + rand * (X_{ub} - X_{lb}) \tag{8}$$

Here, X_{expl} is the exploration solution, X_{lb} and X_{ub} represent the lower and upper bounds of the search space, and (rand) is a random number between 0 and 1.

B. Update phase: Evaluate the objective function for each hawk solution and update the population by selecting the solutions that lead to better objective function values.

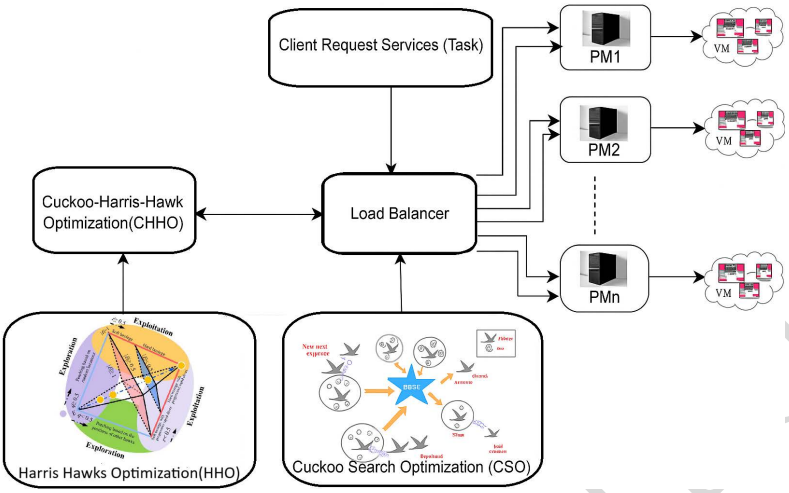

Figure 2. Proposed Cuckoo Harris Hawks multiobjective Optimization (CHHO) Algorithm

C. Termination criterion: Iteratively carry out the searching and updating stages until a halting requirement is satisfied. A maximum number of iterations, a convergence threshold, or other circumstances unique to the issue might be used as these criteria.

The presented algorithm identifies the under-loaded container that meets the criteria for energy efficiency (E) and response time (r) under the challenging "progressive dives" condition, designating it as the optimal under-loaded container. After that, a migration procedure is carried out to move work from containers that are overcrowded to the containers that have been determined to be under-loaded. Until all jobs are evenly allocated and load-balanced within the containers, this iterative procedure is carried out repeatedly.

5.4. Proposed CHHO algorithm

Step 1: Initialization

1. Initialize the system by setting up virtual machines (VMs), hosts, containers, and tasks.

Step 2: Job scheduling

2. Schedule tasks across VMs using a random number scheduling algorithm.

Step 3: Container classification

3. For each VM:
   Calculate the remaining capacity of the VM.
   Classify the containers into three categories:

i. Overloaded containers
ii. Under-loaded containers
iii. Balanced containers

Step 4: Apply CHHO to optimize under-loaded containers

4. Apply the CHHO (Chaotic Harris Hawks Optimization) algorithm to determine the optimal under-loaded container for load balancing.

Step 5: Initialization of CHHO parameters

I. Set the population size as the number of under-loaded containers.
II. Set the iteration limit to 70.
III. Estimate the initial energy of the prey (remaining resources of the under-loaded containers).
IV. Define the goal function considering of the following criteria:
   (a) Minimize cost
   (b) Maximize resource utilization
   (c) Reduce task-completion time
V. Define the fitness function to optimize:
   (a) Minimal cost
   (b) Maximal resource utilization
   (c) Minimal response time
VI. Use Harris Hawks Optimization (HHO) to determine optimal energy (E) and resource utilization rate (r):
   (a) Estimate the number of under-loaded containers needed to balance the over-loaded containers.
   (b) Update the search space using the best values for E and r.
VII. Update the position of the prey using a hard besiege strategy with incremental dives to further refine the selection.
VIII. Identify the under-loaded container that best satisfies the optimal values for *???* in the besiege process.
IX. Check for convergence after each iteration and continue for up to 70 iterations, E and r.

Step 6: Task migration

6. Migrate tasks from overloaded containers to the optimal under-loaded containers determined by the CHHO algorithm.

Variables (N, T_max, E_opt, etc.) are clearly defined. The algorithm aims to efficiently allocate tasks to virtual machines (VMs) within cloud containers, ensuring optimal resource utilization, reduced response times, and minimized operational costs.

In Step 1 the algorithm begins with the initialization of key elements, including virtual machines (VMs), hosts, tasks, and containers, setting the foundation for task scheduling and load balancing.

Step 2 involves the random scheduling of tasks onto the available virtual machines, a fundamental step in the load-balancing process. This step ensures the initial allocation of tasks to VMs in a randomized manner.

After that, Step 3 addresses container management, dividing them based on the remaining capacity of VMs into three categories: under-load, overload, and balanced containers. This step prepares the system for effective load distribution.

Step 4 introduces the CHHO algorithm, which identifies the best-under-loaded container among the available options. The goal of the CHHO optimization approach is to balance work distribution while taking cost, reaction time, resource utilization, and other aspects into account.

Then, Step 5 delves into the initialization of CHHO's parameters, including defining the number of populations (representing the under-loaded containers) and setting an iteration limit of 70. This iteration limit ensures that the optimization process is performed up to a specific number of times, allowing for a comprehensive search.

Further, in Step 5, the prey's starting energy, which represents the under-loaded container's remaining energy, is calculated using the algorithm. The objective function, which entails minimizing reaction time, maximizing resource utilization, and minimizing cost, is evaluated. These objectives are critical for optimizing cloud resource usage. In the end, the CHHO algorithm leverages the Cuckoo Search Optimization (CSO) approach to select the best values for energy (E) and response time (r), storing these values for further use. It estimates the required number of under-loaded containers, ensuring balance with the overload containers. The algorithm updates the search space based on the best values of E and r, allowing for more efficient optimization.

Step 6 finalizes the process with task migration, transferring tasks from overloaded containers to under-loaded ones, further improving resource distribution and enhancing the overall system efficiency.

Throughout these steps, the algorithm continuously checks for convergence, ensuring that the optimization process does not exceed the specified limit of 70 iterations. This comprehensive approach to load balancing in cloud computing helps mitigate issues related to resource under-utilization, high response times, and increased operational costs, ultimately improving quality of service and efficiency within cloud environments.

6. CloudSim software and test configuration

The assessment of the load-balancing algorithm's performance was conducted through a simulation-based analysis. For this purpose, CloudSim, a specialized simulation software designed for cloud computing environments, was utilized. CloudSim offers

a valuable platform for developers to model, simulate, and test cloud computing services and applications accurately. In this study, the cloud computing experiments were executed using the CloudSim 5.0 simulator. The simulation runs on a machine equipped with an Intel Core i5 processor, 4 GB of RAM, a 3.30 GHz CPU, and the Windows operating system. Detailed information about the simulation environment is provided in Table 2.

Table 2
CloudSim Software and Test Configuration

Type.	Number	*Parameters*	Value
Host.	4	*Cores*	6
.		*RAM*	16GB
.		*Storage*	4 TB
.		*MIPS*	117.73
.		*Bandwidth*	15 GB/s
Data Centre.	1	*Arch*	x86
.		*VMMonitor*	Xen
.		*CostPerMemory*	0.05
.		*Costperstorage*	0.001
.		*OS*	Linux
.		*Cost*	3
VM.	300	*Processorspeed*	9726 MIPS
.		*VMmonitor*	Xen
.		*Bandwidth*	1 GB/s
.		*NumberofPEs*	1
.		*Memory*	0.5 GB

7. Performance analysis and comparison

Comparing the recently implemented dynamic load balancing technique (CHHO) with existing approaches allowed for an evaluation to be conducted in order to determine its efficacy. Initially, a variety of measures were used to evaluate the created system's performance, including the number of migrated jobs, throughput, energy consumption, degree of imbalance, standard deviation, makespan, and power of processing of related virtual machines. The effectiveness of the proposed system was then compared to other methods in a comparative analysis, including MPSO, MMPSO, QMPSO, and Q-Learning. Furthermore, analysis of the effectiveness of the created method was evaluated to standalone HHO, CSO, and MRFO, taking into account the differences in the number of jobs and virtual machines.

Table 3 and Figure 3 present energy utilization (in kJ) for different algorithms with varying numbers of virtual machines (VMs).

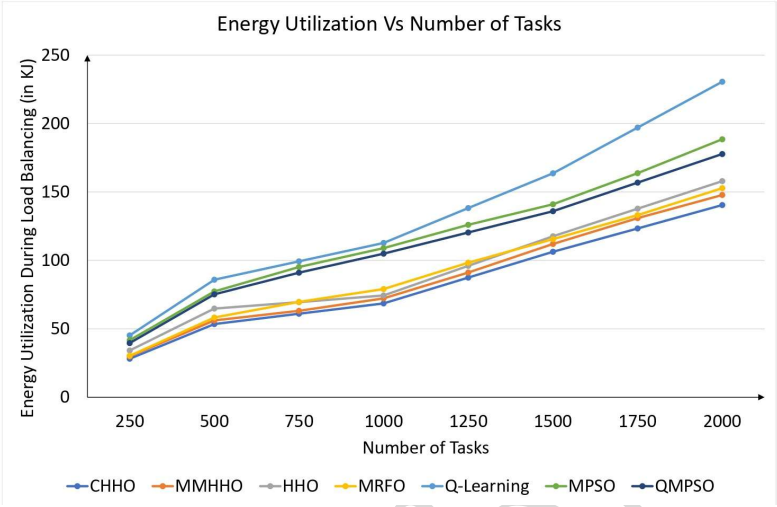

Figure 3. Energy utilization with the number of tasks during load balancing

Table 3
Energy utilization (kJ) for different algorithms and number of VMs

Energy Utilization(kJ)					
Algorithm	Number of VMs				
	100	150	200	250	300
CHHO (Proposed)	143.5	161.3	178.4	184.3	194.2
MMHHO	145.4	165.525	185.65	193.935	202.22
HHO	150.5	183.45	201.4	211.22	222.04
MRFO	155.4	171.95	193.5	201.6	209.7
Q-Learning	174.3	198.85	213.4	230.76	243.12
MPSO	164.88	187.89	204.9	214.8	230.7
QMPSO	150.56	174.13	194.7	204.1	213.5

The proposed CHHO algorithm demonstrates superior performance, consistently outperforming MMHHO, MPSO, MRFO, HHO, QMPSO, and Q-Learning across all VM configurations. Notably, CHHO achieves lower energy consumption, showcasing its efficiency in resource allocation for cloud computing environments, and in Table 4 and Figure 4 it illustrates energy utilization (in kJ) for various algorithms across different task quantities. The CHHO algorithm, proposed in this research, consistently outperforms MMHHO, HHO, MRFO, Q-Learning, MPSO, and QMPSO. Particularly notable is CHHO's efficiency in minimizing energy consumption, demonstrating its effectiveness in task allocation for enhanced performance in cloud computing scenarios with varying task workloads.

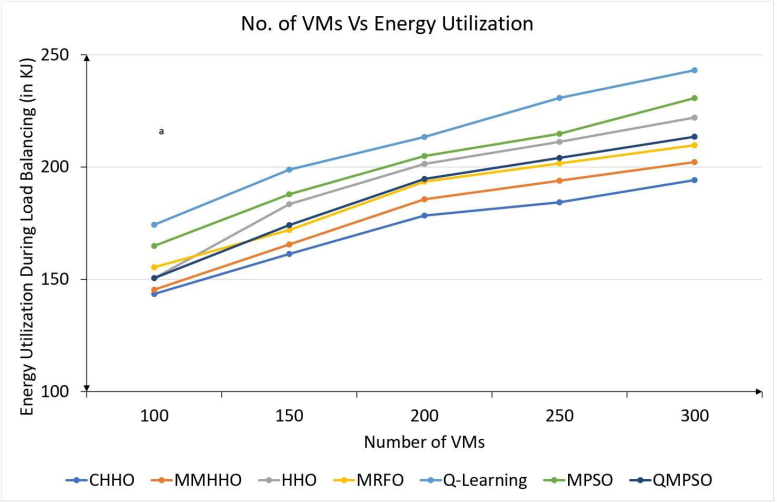

Figure 4. Energy utilization with the number of virtual machines during load balancing

Table 4
Energy Utilization with the Number of Tasks During Load Balancing

Energy Utilization (kJ)				
Algorithm	Number of VMs			
	500	1000	1500	2000
CHHO (Proposed)	53.44	68.54	106.34	140.45
MMHHO	56.09	72.21	111.98	147.78
HHO	64.76	74.33	117.65	157.9
MRFO	58.21	79.09	115.5	152.76
Q-Learning	85.87	112.78	163.65	230.56
MPSO	77.26	108.98	140.98	188.54
QMPSO	75.21	104.89	135.93	177.75

Table 5 and Figure 5 display makespan values (in ms) for diverse algorithms across varying numbers of virtual machines. The proposed CHHO algorithm consistently achieves the lowest makespan, indicating superior task-completion efficiency. In contrast, MMHHO, HHO, MRFO, Q-Learning, MPSO, and QMPSO exhibit higher makespan values. CHHO stands out for its effectiveness in minimizing task completion time and optimizing resource utilization, and Table 6 and Figure 6 present makespan values (in ms) for different algorithms across various task quantities. The proposed CHHO algorithm consistently achieves lower makespan, indicating enhanced task completion efficiency. Notably, CHHO outperforms MMHHO, HHO, MRFO,

Q-Learning, MPSO, and QMPSO across all task configurations, emphasizing its effectiveness in minimizing task completion time and optimizing resource utilization in diverse computing scenarios.

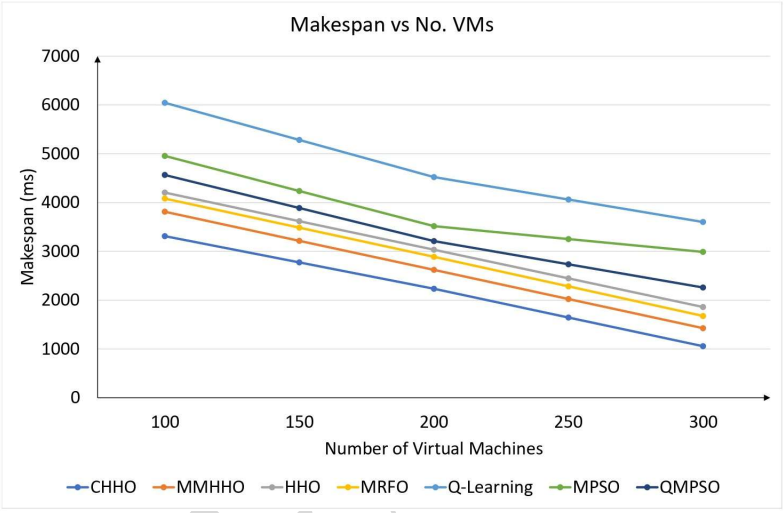

Figure 5. Makespan with the number of virtual machines during load balancing

Table 5
Makespan with the number of virtual machines during load balancing

Makespan (ms)				
Algorithm				
	500	1000	1500	20000
CHHO (proposed)	1350	2680	4450	6245
MMHHO	1545	2970	4705	6610
HHO	2021	3440	5280	7114
MRFO	1812	3216	5021	6911
Q-Learning	3443	5521	6859	8379
MPSO	3248	5169	6704	8059
QMPSO	3022	4878	6488	7809

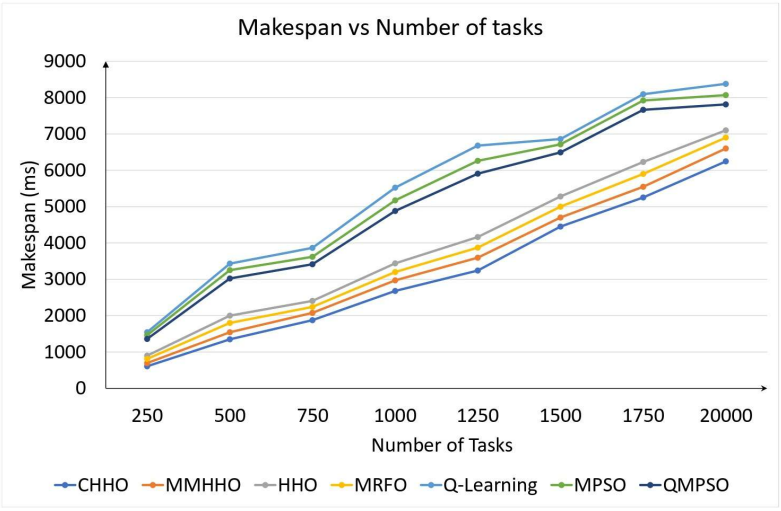

Figure 6. Makespan with the number of tasks during load balancing

Table 6
Makespan with the number of tasks during load balancing

Makespan (ms)					
Algorithm	Number of VMs				
	100	150	200	250	300
CHHO (proposed)	3312	2773	2234	1645	1056
MMHHO	3812	3216	2620	2023	1426
HHO	4204	3619	3034	2446	1859
MRFO	4082	3485	2888	2282	1676
Q-Learning	6045	5284	4523	4062	3601
MPSO	4956	4236	3517	3252	2988
QMPSO	4565	3888	3211	2735	2259

Table 7 and Figure 7 depict migration counts for different algorithms across varying task quantities. The proposed CHHO algorithm consistently demonstrates the lowest migration numbers, indicating its efficiency in task placement and reduced resource relocation. In contrast, MMHHO, HHO, MRFO, Q-Learning, MPSO, and QMPSO exhibit higher migration rates, emphasizing CHHO's effectiveness in minimizing task migration for improved stability and performance in cloud computing environments.

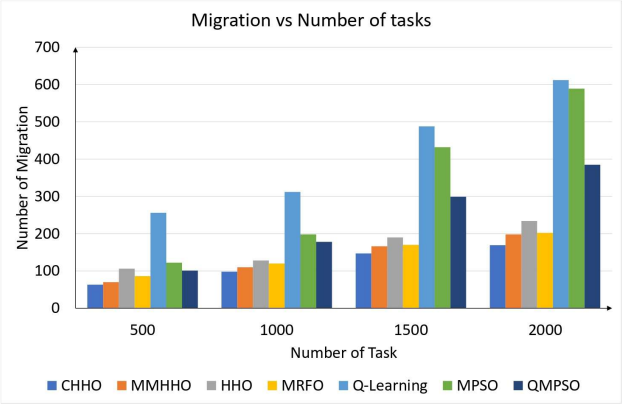

Figure 7. Migration with the number of tasks during load balancing

Table 7
Migration with the number of tasks during load balancing

Migration				
Algorithm	Number of Tasks			
	500	1000	1500	2000
CHHO (proposed)	63	98	147	169
MMHHO	70	110	166	198
HHO	106	128	190	234
MRFO	86	120	170	202
Q-Learning	256	312	488	612
MPSO	122	198	432	589
QMPSO	101	178	299	385

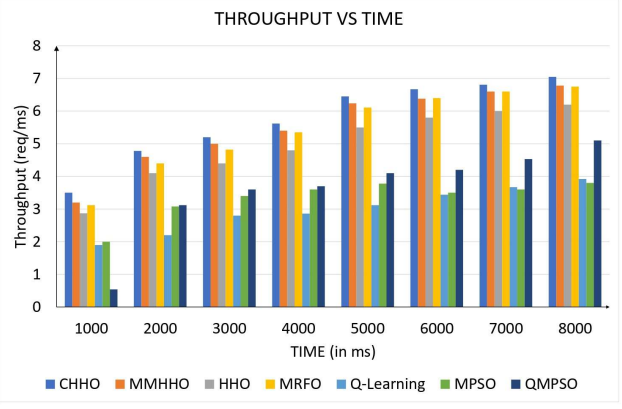

Figure 8. Throughput with respect to time during load balancing

The table displays throughput values (req/ms) for various algorithms over different time intervals. The proposed CHHO algorithm consistently outperforms MMHHO, HHO, MRFO, Q-Learning, MPSO, and QMPSO in terms of throughput. Notably, CHHO exhibits increasing throughput with time, showcasing its effectiveness in handling requests. QMPSO demonstrates a peculiar trend, with an initial dip and subsequent improvement. The results emphasize CHHO's superior performance and potential for efficient request processing in cloud-computing environments over varying time intervals.

Table 8
Throughput with respect to time during load balancing

Throughput (req/ms)								
Algorithm	Time (in ms)							
	1000	2000	3000	4000	5000	6000	7000	8000
CHHO (Proposed)	3.5	4.78	5.2	5.62	6.45	6.67	6.81	7.05
MMHHO	3.2	4.6	5	5.4	6.24	6.38	6.6	6.78
HHO	2.87	4.1	4.4	4.8	5.5	5.8	6	6.2
MRFO	3.12	4.4	4.82	5.35	6.11	6.4	6.6	6.75
Q-Learning	1.9	2.2	2.8	2.86	3.12	3.44	3.67	3.92
MPSO	2.001	3.08	3.4	3.6	3.78	3.5	3.6	3.8
QMPSO	0.54	3.12	3.6	3.7	4.1	4.2	4.53	5.1

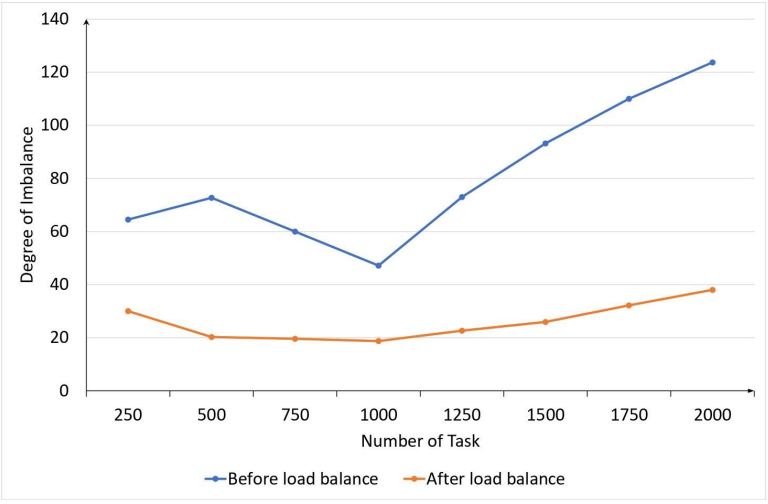

Figure 9. Measurement of the suggested scheme's degree of imbalance both after and before the applied load-balancing approach

Table 9 illustrates the degree of imbalance for the proposed CHHO algorithm across various task quantities. Initially, before load balancing, the system displays uneven task distribution. However, after load balancing, CHHO significantly reduces the degree of imbalance, achieving a more equitable distribution. This underscores CHHO's effectiveness in optimizing resource allocation, ensuring a balanced workload, and improved performance in cloud computing environments.

Table 9
Measurement of the suggested scheme's degree of imbalance both after and before the applied load-balancing approach

Degree of Imbalance								
Algorithm	Number of Tasks							
	250	500	750	1000	1250	1500	1750	2000
Before load balance	64.5	72.7	60	47.15	73	93.15	110	123.71
After load balance	30	20.26	19.6	18.73	22.67	25.92	32.17	38.02

8. Conclusion and future possibilities

The load-balancing problem in cloud networks was addressed by developing CHHO, an improved hybrid optimization technique. Following CHHO's implementation in CloudSim, a number of critical metrics were used to evaluate its performance, including degree of imbalance, throughput, energy usage, makespan, and task migration. The CHHO algorithm leverages the exploration capabilities of Cuckoo Search Optimization and the exploitation strengths of the Harris Hawks optimization. By integrating the random walk and Lävy flight mechanisms of Cuckoo Search, CHHO enhances its ability to explore diverse solutions, avoiding local optima. Simultaneously, the cooperative hunting strategy of Harris Hawks Optimization ensures effective exploitation, refining the search for the optimal solution. This combination results in a robust approach to task scheduling and balancing load efficiently in cloud environments. Achieving balance in task scheduling within cloud computing infrastructure significantly enhances system efficiency and resource allocation. The CHHO-based load-balancing approach improves throughput, reduces makespan, and optimizes energy utilization, effectively reducing task waiting times. Simulation results indicate its superiority over existing methods, with potential enhancements through the incorporation of factors such as bandwidth, dynamic context, and dependent tasks with preemption. Future work aims to validate the algorithm's efficacy in real-world cloud computing scenarios.

References

[1] Adaikalaraj J.R., *et al.*: Load Balancing In Cloud Computing Environment Using Quasi Oppositional Dragonfly Algorithm, *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12(10), pp. 3256–3273, 2021.

[2] Adhikari M., Nandy S., Amgoth T.: Meta heuristic-based task deployment mechanism for load balancing in IaaS cloud, *Journal of Network and Computer Applications*, vol. 128, pp. 64–77, 2019. doi: 10.1016/j.jnca.2018.12.010.

[3] Annie Poornima Princess G., Radhamani A.: A hybrid meta-heuristic for optimal load balancing in cloud computing, *Journal of grid computing*, vol. 19(2), p. 21, 2021. doi: 10.1007/s10723-021-09560-4.

[4] Balaji K., Kiran P.S., Kumar M.S.: WITHDRAWN: An energy efficient load balancing on cloud computing using adaptive cat swarm optimization, 2021. doi: 10.1016/j.matpr.2020.11.106.

[5] Ebadifard F., Babamir S.M., Barani S.: A dynamic task scheduling algorithm improved by load balancing in cloud computing. In: *2020 6th International Conference on Web Research (ICWR)*, pp. 177–183, IEEE, 2020. doi: 10.1109/icwr49608.2020.9122287.

[6] Ejaz H., Awan M.A., Muzzammil H., Ullah M., Akhavan-Safar A., daSilva L., Tanveer A.: Strength improvement/optimization methods in adhesively bonded joints: A comprehensive review of past and present techniques, *Mechanics of Advanced Materials and Structures*, pp. 1–29, 2024. doi: 10.1080/15376494.2024.2400610.

[7] Farrag A.A.S., Mohamad S.A., El Sayed M.: Swarm Intelligent Algorithms for solving load balancing in cloud computing, *Egyptian Computer Science Journal*, vol. 43(1), pp. 45–57, 2019.

[8] Golchi M.M., Saraeian S., Heydari M.: A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: Performance evaluation, *Computer Networks*, vol. 162, p. 106860, 2019. doi: 10.1016/j.comnet.2019.106860.

[9] Haris M., Khan R.Z.: A systematic review on cloud computing, *International Journal of Computer Sciences and Engineering*, vol. 6(11), pp. 632–639, 2018. doi: 10.26438/ijcse/v6i11.632639.

[10] Haris M., Khan R.Z.: A systematic review on load balancing issues in cloud computing, *Sustainable Communication Networks and Application: ICSCN 2019*, pp. 297–303, 2020.

[11] Haris M., Zubair S.: Mantaray modified multi-objective Harris hawk optimization algorithm expedites optimal load balancing in cloud computing, *Journal of King Saud University-Computer and Information Sciences*, vol. 34(10), pp. 9696–9709, 2022. doi: 10.1016/j.jksuci.2021.12.003.

[12] Hou X., Zhao G.: Resource scheduling and load balancing fusion algorithm with deep learning based on cloud computing, *International Journal of Information Technology and Web Engineering (IJITWE)*, vol. 13(3), pp. 54–72, 2018. doi: 10.4018/978-1-7998-0414-7.ch058.

[13] Jena U.K., Das P.K., Kabat M.R.: Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment, *Journal of King Saud University-Computer and Information Sciences*, vol. 34(6), pp. 2332–2342, 2022. doi: 10.1016/j.jksuci.2020.01.012.

[14] Junaid M., Sohail A., Ahmed A., Baz A., Khan I.A., Alhakami H.: A hybrid model for load balancing in cloud using file type formatting, *IEEE Access*, vol. 8, pp. 118135–118155, 2020. doi: 10.1109/access.2020.3003825.

[15] Junaid M., Sohail A., Rais R.N.B., Ahmed A., Khalid O., Khan I.A., Hussain S.S., Ejaz N.: Modeling an optimized approach for load balancing in cloud, *IEEE access*, vol. 8, pp. 173208–173226, 2020. doi: 10.1109/access.2020.3024113.

[16] Kaur A., Kaur B.: Load balancing optimization based on hybrid Heuristic-Metaheuristic techniques in cloud environment, *Journal of King Saud University-Computer and Information Sciences*, vol. 34(3), pp. 813–824, 2022. doi: 10.1016/j.jksuci.2019.02.010.

[17] Kaur A., Narang N.: Multi-objective generation scheduling of integrated energy system using hybrid optimization technique, *Neural Computing and Applications*, vol. 36(3), pp. 1215–1236, 2024.

[18] Kumar P., Kumar R.: Improved Active Monitoring Load-Balancing Algorithm in Cloud Computing. In: *Proceedings of 2nd International Conference on Communication, Computing and Networking: ICCCN 2018, NITTTR Chandigarh, India*, pp. 1033–1040, Springer, 2019.

[19] Li G., Wu Z.: Ant colony optimization task scheduling algorithm for SWIM based on load balancing, *Future Internet*, vol. 11(4), p. 90, 2019. doi: 10.3390/fi11040090.

[20] Milan S.T., Rajabion L., Ranjbar H., Navimipour N.J.: Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments, *Computers & Operations Research*, vol. 110, pp. 159–187, 2019. doi: 10.1016/j.cor.2019.05.022.

[21] Mohanty S., Patra P.K., Ray M., Mohapatra S.: An approach for load balancing in cloud computing using JAYA algorithm, *International Journal of Information Technology and Web Engineering (IJITWE)*, vol. 14(1), pp. 27–41, 2019. doi: 10.4018/ijitwe.2019010102.

[22] Narale S.A., Butey P.: Throttled load balancing scheduling policy assist to reduce grand total cost and data center processing time in cloud environment using cloud analyst. In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pp. 1464–1467, IEEE, 2018. doi: 10.1109/icicct.2018.8473062.

[23] Negi S., Rauthan M.M.S., Vaisla K.S., Panwar N.: CMODLB: an efficient load balancing approach in cloud computing environment, *The Journal of Supercomputing*, vol. 77(8), pp. 8787–8839, 2021. doi: 10.1007/s11227-020-03601-7.

[24] Prassanna J., Venkataraman N.: Adaptive regressive holt–winters workload prediction and firefly optimized lottery scheduling for load balancing in cloud, *Wireless Networks*, vol. 27(8), pp. 5597–5615, 2021.

[25] Semmoud A., Hakem M., Benmammar B., Charr J.C.: Load balancing in cloud computing environments based on adaptive starvation threshold, *Concurrency and Computation: Practice and Experience*, vol. 32(11), p. e5652, 2020. doi: 10.1002/cpe.5652.

[26] Sethi N., Singh S., Singh G.: Improved mutation-based particle swarm optimization for load balancing in cloud data centers. In: *Harmony Search and Nature Inspired Optimization Algorithms: Theory and Applications, ICHSA 2018*, pp. 939–947, Springer, 2019.

[27] Shafiq D.A., Jhanjhi N.Z., Abdullah A., Alzain M.A.: A load balancing algorithm for the data centres to optimize cloud computing applications, *IEEE Access*, vol. 9, pp. 41731–41744, 2021. doi: 10.1109/access.2021.3065308.

[28] Siddiqui S., Darbari M., Yagyasen D.: An QPSL queuing model for load balancing in cloud computing, *International Journal of e-Collaboration (IJeC)*, vol. 16(3), pp. 33–48, 2020. doi: 10.4018/ijec.2020070103.

[29] Singh I., Dhillon S.K.: Optimization Techniques for Mechatronics: A Comprehensive Review and Future Directions, *Computational Intelligent Techniques in Mechatronics*, pp. 109–133, 2024. doi: 10.1002/9781394175437.ch4.

[30] Thirumal K., Sakthivel V., Sathya P.: Solution for short-term generation scheduling of cascaded hydrothermal system with turbulent water flow optimization, *Expert Systems with Applications*, vol. 213, p. 118967, 2023. doi: 10.1016/j.eswa.2022.118967.

[31] Tyagi N., Rana A., Kansal V.: Creating elasticity with enhanced weighted optimization load balancing algorithm in cloud computing. In: *2019 Amity International Conference on Artificial Intelligence (AICAI)*, pp. 600–604, IEEE, 2019. doi: 10.1109/aicai.2019.8701375.

[32] Xingjun L., Zhiwei S., Hongping C., Mohammed B.O.: A new fuzzy-based method for load balancing in the cloud-based Internet of things using a grey wolf optimization algorithm, *International Journal of Communication Systems*, vol. 33(8), p. e4370, 2020. doi: 10.1002/dac.4370.

Affiliations

Rahul Kumar Sharma
MMMMUT,Gorakhpur, rahulsharma9045@gmail.com

Sarvpal Singh
MMMMUT,Gorakhpur, spsitca@mmmut.ac.in

Received: 21.08.2024
Revised: 06.09.2024
Accepted: 28.08.2025