Journal of Cloud Computing

RESEARCH Open Access

Dynamic scheduling strategies for cloud-based load balancing in parallel and distributed systems

Nasser S. Albalawi[1*]

Abstract

Actual load balancing in parallel and distributed systems ruins a serious task due to workloads' dynamic nature and resource availability. Existing scheduling procedures continually fail to regulate real-time alterations, leading to suboptimal performance and resource underutilization. Our study validates dynamic and effective load distribution by combining novel systems and optimization techniques to handle these issues. We utilize a comprehensive dynamic scheduling approach in this work to provide efficient load balancing in distributed and parallel systems. In this example, we start by using Round-Robin Allocation with Sunflower Whale Optimization (RRA-SWO) to perform an allocation procedure. The allocation step is followed by the Hybrid Ant Genetic Algorithm (HAGA), which is used to schedule tasks in parallel. The Least Response Time (LRT) technique for the Load Monitoring procedures will be developed once the job scheduling is complete. The Harmony Search Algorithm with Linear Regression (LR-HSA) is then used to do Distributed Computing-based Load Prediction and Adjustment. Alongside ongoing observation, this is carried out. Finally, we use the Least Recently Used (LRU) technique to do dynamic load balancing. Performance evaluations are using CloudSim and NetBeans 12.3, metrics like Packet Delivery Ratio at 98 (%), Average Response Time at 65 (s), Task Success Rate at 95 (%), Memory Utilization Rate at 80 (%), and Throughput at 97 (%) are all analyzed to validate our strategy.

Keywords Dynamic scheduling strategies, Load balancing, Parallel and distributed systemsf, Least Response Time (LRT), Round-Robin Allocation with Sunflower Whale Optimization (RRA-SWO), Least Recently Used (LRU)

Introduction

Stability virtual machine hosting (VMHs) can be loaded via migration of virtual machines (VMs) from one overcrowded virtual machine host (VMH) to more VMHs. In an attempt to evenly disperse the burden among all VMHs, this migration is being carried out [1]. The three work steps in this technique are detection, decision, and action. Finding any server farm imbalances is the aim of the detecting step. During the decision-making step, decisions are taken regarding which VMs to migrate and which VMH to receive them [2]. It is necessary to suspend, switch between VMHs, and then resume the selected virtual machines. Only the necessary virtual machines for transfer are moved when the migration suspends and resumes, leaving the remaining VMs in the VMHs operational. Because of this, VMH workloads are dynamic and may grow dramatically as a result of VM migration. An effective load-balancing technique is consequently required for the management of VMHs, although it is difficult. Systems for distributed and

*Correspondence:
Nasser S. Albalawi
Nasser.albalawi@nbu.edu.sa
[1]Department of Computer Sciences, Northern Border University, Rafha, Saudi Arabia

parallel computing are popular and appropriate choices for use in online applications that require sophisticated calculations [3, 4]. Multiple resources for processing are working on an operation in a distributed or parallel fashion to accelerate task execution [5]. Grid networks, cloud environments, and cluster networks are some of the most widely used distributed processing structures [6]. Cloud processing networks, in which all users always have access to high-performance computers (HPCs), are regarded as one of the most significant and well-liked distributed processing models [7]. Cloud resource management is the process of allocating handling, storing, and energy sources to a collection of apps to meet the needs of customers and cloud service providers [8]. Users of the cloud seek out dependable high-quality services at fair prices, as well as security and privacy. On the other hand, cloud providers aim to maximize resource use, minimize energy consumption, and improve resource availability [9, 10]. The act of figuring out when a task in a workflow should begin while taking into account all of the dependencies in the workflow and the job expiration dates is known as cloud task scheduling. For separate jobs, the task scheduling procedure is typically not difficult. Nevertheless, a workflow's interdependence might make scheduling difficult. The tasks should be scheduled and prioritized, and then the proper resources should be assigned to them. The right processing resource is needed for each task based on how long it will take to complete [11, 12]. Inadequate resource provisioning decreases resource utilization efficiency in addition to lengthening workflow execution times, increasing costs, and consuming more energy. On the other side, neglecting the load balancing of the resources might drastically reduce their efficiency. Thus, cloud service providers have significant issues regarding the timely and effective scheduling of tasks and the provisioning of resources. Scientific operations that are complex call for strong processing capabilities. Their scheduling method is considerably more complex than most other types of processes because of their task dependencies [13, 14]. When it comes to online workflows, this intricacy may further grow. A distributed computing system is made up of several networked computers that can communicate with one another. To assist users in solving a complex computational issue, the distributed structure can manage the utilization of resources. Utilizing a distributed computing system has several benefits over traditional centralized computing, such as lower costs, increased performance, and increased dependability. Because of this, the distributed computing system has been widely used in many different fields, including software, hardware, power generation, and electronic sensor systems. A distributed computing system's quality can be assessed in several ways, such as dependability, affordability, and efficiency [15, 16]. Generally speaking, the distributed hardware or software system of the shared memory is the parallel paradigm. As a result, sharing or distributing parallel techniques is feasible. Because the algorithm must handle particular problems such as memory scalability, job splitting, and load balancing for parallel computation, its successful design is difficult in parallel processing [17, 18]. Other difficulties include minimizing the costs associated with synchronization and communications, identifying effective techniques for data breakdown and layout, and minimizing I/O overhead. A distributed scheme is a collection of separate computing elements that together provide a dependable system for its users [19, 20]. We combine heuristic and meta-heuristic methods to solve the problems of resource allocation, job scheduling, and dynamic load balancing in cloud and distributed systems. Sunflower Whale Optimization, through dynamic environmental adaptation, optimizes resource allocation with a balance between exploration and exploitation. GA is the perfect algorithm for scheduling tasks in parallel across processors because it preserves variety and offers a global search capacity. The Hybrid Ant Genetic Algorithm (HAGA), which incorporates Ant Colony Optimization as a pheromone-based learning technique, is exploited to find the best possible scheduling routes and improve solutions' quality. Real-time task monitoring and prioritizing using Least Response Time can help avoid bottlenecks and achieve efficient load balancing. The linear regression (LR) will forecast the workload patterns, which can be used to predict resource changes. The Harmony Search Algorithm (HSA) will change resources proactively by optimizing through memory. Together, these techniques provide a strong and flexible way to maximize efficiency and resource use in changing settings.

Motivation & objectives

The following difficulties have been recognized in existing study, each needful advanced keys to improve the efficiency of load balancing approaches:

- ➢ *Optimality and VM-Failures*: The existing approach confirms the balancing of load but absences established optimality and a robust technique for management of VM failures, foremost to it take more time to process and degrade.
- ➢ *Dynamic Task Constraints*: The existing method wants alteration to lever tasks with dynamic limitations on mechanism requirements.
- ➢ *Workload Limit Consideration*: The existing procedure does not consider the load limit for each working node, affecting overall scheme performance.
- ➢ *Limitation in Multi-Cloud Scheduling*: Challenges in existing methods include task scheduling in multi-

cloud situations and the need for effectual planning of tasks on VMs.
- ➢ *Large-Scale Simulations*: Existing algorithms lack essential components for large-scale simulation and the capability to handle multiple organizations for managing servers within information center networks.
- ➢ *Rescheduling and Migrations*: existing methods face challenges in rescheduling tasks and handling frequent migrations, which affects system performance.

The primary goals of this study are to better understand how effective resource management ensures optimal performance, dependability, and scalability in modern computing. Distributing work and resources among several nodes can be difficult. Dynamic scheduling techniques solve this problem.

- ➢ To utilize a novel approach for effective resource allocation.
- ➢ To employ the effective approach to distribute tasks efficiently across multiple processors.
- ➢ To obtain the innovative algorithm to collect and analyze real-time load data.
- ➢ To apply the novel algorithm for proactive load adjustment.
- ➢ To implement the Threshold-based Load Balancing with Priority-based Task Rescheduling using an effective method.

Research contributions

The objective of this research is to create and assess dynamic scheduling plans that improve distributed and parallel systems load balancing. The integration of several algorithms and techniques to maximize resource usage, minimize makespan, and guarantee system resilience is included in the scope. Below are some of this research's main contributions,

- ➢ We propose a new integration of Round-Robin Allocation and Sunflower Whale Optimization (RRA-SWO) to balance exploration and exploitation effectively, maximizing the use of resources and minimizing task turnaround time.
- ➢ The combination of Ant Colony Optimization (ACO) and Genetic Algorithms (GA) in the Hybrid Ant Genetic Algorithm (HAGA) allows for effective parallel task scheduling with minimal bottlenecks and processor optimization.
- ➢ The Least Response Time (LRT) method is employed for real-time load monitoring to maintain balanced load distribution among virtual machines and decrease system latency.
- ➢ We utilize the Harmony Search Algorithm with Linear Regression (LR-HSA) to forecast future workload fluctuations, allowing proactive load adjustment to improve system robustness.
- ➢ A Threshold-based Load Balancing strategy along with the Least Recently Used (LRU) methodology is used for efficient dynamic task rescheduling so that there is negligible downtime and maximum resource usage.

These contributions together lead to substantial enhancements in major performance metrics, including improved Packet Delivery Ratio, enhanced Task Success Rate, minimized Average Response Time, improved Memory Utilization, and enhanced Throughput. Large-scale simulation with CloudSim verifies the efficacy of our approach.

Research organizations

The following parts comprise the remaining portion of this document: Section II illustrates the literature study of the earlier research that is more pertinent to our work. Section III presents the main problem statements addressed in the previous literature. Section IV presents the research methodology for the proposed work, which consists of a pseudocode, a mathematical representation, and a protocol. In Section V, the experimental results are presented along with a comparison of the recommended and ongoing works. Section VI offers a conclusion to the suggested study as well as future work plans for this research.

Literature survey

This section deals with the survey of literature on Dynamic Scheduling Strategies for Load Balancing in Parallel and Distributed Systems, Authors in [21], investigate the obtainable powerful arrangement method called "Deep Reinforcement Learning with Parallel Particle Swarm Optimization (DRLPPSO)" to rapidly and precisely resolve the load balancing issues and its many limits. Also, a cross method that syndicates machine learning and swarm optimization has been obtainable to improve the thoughts of resource management and resource allocation. By this method, real-time analytics on the complicated and active cloud network will be believable. Authors in [22] provide in this research decreases energy consumption and makespan through upholding system balance, in addition to enhancing resource utilization. It originally distillates on initiating a load balancing procedure to consistently allocate the loads among VMs following the task's compatibility with VMs. Then, the JAYA procedure is run to regulate the ideal mapping of errands onto VMs. Also, the efficiency of this technique strength be used to progress a scheduling procedure based on green cloud computing. The

article [23] suggests Swarm Intelligence (SI) as a cloud computing load-balancing solution. The load balancing convergence time with global optimization is not taken into consideration by any of the alternatives that have been studied in the literature, including genetic algorithms, ACO, PSO, BAT, GWO, and many more. Grey Wolf Optimization (GWO) and Particle Swarm Optimization (PSO) are the two methods that are highlighted in this study. In this work, a combined GWO-PSO strategy that leverages the advantages of global optimization and quick convergence is suggested.

Authors in [24] demonstrate This research provides an innovative technique of dynamic load balancing among VMs utilizing the hybridizing of "modified particle swarm optimization (MPSO)" and an improved Q-learning process, known as QMPSO. The hybridizing process is employed to regulate the MPSO speed by combining the best action produced by the improved Q-learning. By reducing task waiting times, boosting virtual machine throughput, and maintaining task priorities through workload balance among VMs, hybridization will increase the machine's performance. This is where dynamic load balancing amongst the dependent jobs will take place. Authors in [25] provide in this study, that they primarily focus on two methods for determining the optimal scheduling of the many interdependent tasks in a distributed, parallel system. To efficiently search the scheduling solution space for several dependent jobs, the first method proposes to use a hybrid technique. The idea is to utilize heuristic methods to first discover a timeline, and then use those as initial points in the search space to apply unsupervised machine learning techniques to find better answers. Every task graph in the second way is split up into multiple sub-task clusters, each of which is mapped onto a single resource. This tactic, in contrast to the first, tries to reduce resource idle times as much as feasible. One new feature of the planned methods is that they schedule the different dependent activities simultaneously instead of consecutively. Using the suggested scheduling strategies here allows for the optimization of additional scheduling parameters.

Authors in [26] present a fog scheduling method based on reinforcement learning to get over these problems. The results of the testing indicate that the proposed algorithm enhances load balancing and decreases response time when compared to the existing scheduling algorithms. Moreover, the proposed algorithm outperforms other approaches concerning the number of devices used. It is recommended that fuzzy logic be used in future research to determine the incentive. Three elements can be considered by the fuzzy system: resource efficiency, device load imbalance, and internal load imbalance. The output of the fuzzy system can then be returned as a reward to the learning agent. Synchronization between the fog and cloud levels may also be taken into account in subsequent studies. The mobility of IoT devices is another area that could be explored in further research. Based on a model of a typical vehicle multicast service in 5G-V2X, the authors of [27] present a collaborative multicast service authentication and data delivery method. To enable cars to safely receive multicast service data in point-to-multipoint mode, the plan calls for dividing large cars that are served by the same RAN together and relating them to the content provider so they may utilize the scattered credentials that the 5G network at home securely distributes to access a multicast service. Due to the high failure probability of aggregation verification in the group-based multicast access to the services verification process of the proposed solution, the RAN would have to transmit all signatures again, resulting in overhead costs related to computation and transmission. They'll keep searching for the best means of reducing this limitation.

The author in [28] presents a new dynamic load-balancing method that uses a deep learning model that combines recurrent neural networks (RNNs) and convolutional neural networks (CNNs) to determine load levels for every virtual machine (VM). The technique optimizes stress distribution and work scheduling to improve cloud performance. To divide virtual machines (VMs) into overloaded and underloaded clusters, the suggested architecture uses a dynamic clustering method based on calculated loads. It combines a complex Hybrid Lyrebird Falcon Optimization (HLFO) technique with Reinforcement Learning (RL) to increase clustering efficiency. HLFO improves load balancing efficiency by combining the Lyrebird Optimization Algorithm (LOA) and Falcon Optimization Algorithm (FOA). A load-balancing framework for resource scheduling deployment in cloud-based healthcare contexts is introduced in this article [29]. The schedule resources use reinforcement learning algorithms like GA, SARSA, and Q-learning. The best way to handle the load in cloud-based healthcare settings is predicted by these algorithms.

To handle dynamic resource allocation (DRA) and load balancing (LB) activities in a cloud environment, they offer a reinforcement learning (RL) approach [30]. This approach achieves high scalability and a significant performance improvement. This idea is a dynamic load-balancing method based on the reinforcement learning algorithm Q-learning. This approach takes advantage of Q-learning to continuously learn the best resource distribution strategy based on user preferences, resource accessibility, and workload. They provide a reward system that accounts for economic concerns and performance characteristics like reaction time and resource use. This method achieves optimum bilateral transposed convolution filtering, excellent quality of service,

throughput, scalability, and short response time by using a multi-objective PSO algorithm with Pareto dominance [31]. Moreover, ineffective VM placement across PMs during VM migration causes service-level agreement breaches using current methodologies. A feedback controller-equipped Double Deep Q proximal model has been suggested as a solution to these problems. The decision model's twofold weight for offline and online updating ensures a seamless service level agreement with the cloud. Additionally, both decentralized and centralized controller algorithms have coordination problems and a single point of failure in complex scenarios including process instruction mixing. The article [32] uses a Genetic Algorithm (GA) to provide a new load-balancing method that incorporates task migration. Minimizing the makespan for a certain collection of jobs while achieving appropriate load distribution within the cloud architecture is the goal. With the help of task migration's adaptability and evolutionary algorithms' innate optimization powers, the proposed method shows the dynamic response to the ever-changing needs of cloud computing. The author in [33] provides a three-step method for putting the Johnson Sequencing algorithm into effect for cloud computing work scheduling. First, they examine the connections between jobs to create a precedence graph. Then, by assigning jobs to servers, the precedence graph is converted into a two-machine Johnson Sequencing issue. To minimize the makespan, they finally use the Dynamic Heuristic Johnson Sequencing approach to figure out the optimal work order on each server.

Authors in [28], proposed a dynamic load-balancing method that incorporates deep learning models, namely Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), to compute load values for every virtual machine (VM). The methodology utilizes Reinforcement Learning (RL) with a Hybrid Lyrebird Falcon Optimization (HLFO) algorithm to increase clustering efficiency and maximize task scheduling. The integration of multiple algorithms needs complex implementation strategies, which can be challenging to implement in real-world applications. This systematic review [34], offers a detailed examination of different load balancing and task scheduling techniques in cloud computing, emphasizing the significance of dynamic resource allocation to ensure optimal performance. However, it does not suggest new methodologies but synthesizes existing knowledge. An Enhanced Dynamic Load Balancing (EDLB) algorithm that aims to maximize task scheduling and resource allocation in cloud environments [35]. In contrast to benchmark algorithms that involve static VM selection or post-hoc cloudlet relocation, the EDLB algorithm identifies optimal cloudlet placement dynamically in real-time. Our solution actively assigns cloudlets to VMs according to prevailing system states and Service Level Agreement (SLA) deadlines, thus preemption SLA violations in advance. Moreover, if a VM fails to fulfill the cloudlet's deadline, the algorithm leads the cloudlet to an alternate data center and rearranges CPU resources between VMs for the best possible allocation. The performance of the algorithm on large and more intricate cloud settings is not examined. With a higher number of cloudlets and VMs, keeping load balancing efficient and execution time low might prove to be challenging. In [36], proposes a novel technique for dynamic load balancing in cloud settings called Meta-RHDC, or Meta Reinforcement Learning Driven Hybrid Lyrebird Falcon Optimization for Dynamic Load Balancing in Cloud Computing. Convolutional and recurrent neural networks are used by the Meta-RHDC model to assess virtual machine loads and dynamically divide them into classes that are overloaded and underloaded. Compared to other current algorithms like Load Optimization Algorithm (LOA), Reinforcement Learning (RL), and Falcon Optimization Algorithm (FOA), Meta-RHDC enhances scheduling tasks and load balancing using reinforcement learning in conjunction with advanced optimization techniques. The model does not necessarily deal with the optimization of energy consumption. Though power management is seen to be stable in simulation, real-world deployment in huge cloud infrastructures might expose underlying inefficiencies and higher energy utilization.

The article [37], an optimization technique for energy-efficient scheduling and two-tiered coordinated load balancing. First, they build an energy-saving scheduling algorithm (ESSA) based on an energy efficiency model based on Service Level Agreements (SLA) to lower the energy consumption of Flink clusters during task execution. The impacts of two SLA performance measures, such as node response time and throughput, on node energy consumption are taken into account by this ESSA method. It also takes into account the variations in energy efficiency across nodes in heterogeneous clusters. Secondly, an Energy-Aware Two-Tier Coordinated Load Balancing algorithm (TTCLB-EA) is proposed to address the load imbalance issue that may be brought on by Flink's default scheduling policy. This algorithm optimizes the cluster load at both the intra-node and inter-node levels by prioritizing energy efficiency in tasks. This article [38], proposes a cost-efficient load balancing algorithm (LBA-CE) and a cost-efficient task scheduling algorithm (CETSA) for Flink to optimize load balancing and lower job execution costs. First, a load-balancing model based on Flink and a cost-effective model is built. Next, the cost-efficient model is used to improve the basic mechanism of Flink task scheduling, and the enhanced task scheduler is put into use. Moreover, the load balancing model incorporates the idea of node adaptability into cost-effective scheduling, guaranteeing that the

cluster load is as evenly distributed as feasible while lowering expenses in a heterogeneous cluster. This study [39], suggests a cost-effective scheduling method for the Storm framework (CE-Storm) that satisfies deadline constraints while lowering costs. Initially, a new cost-efficient model based on the Storm framework is constructed, which includes the costs of energy, communication, and resource utilization. A cost-efficient scheduling method that incorporates a communication detection module and a resource monitoring module is then created based on the cost model. To reduce the cluster's overall cost, the cluster's nodes are ranked based on cost-effective information, and the nodes with the highest priority are given jobs first. Additionally, this technique enhances the Storm cluster's cost efficiency by lowering the cost of communication between nodes.

Our proposed methodology is differentiated through the combination of the Sunflower Whale Optimization (SWO) algorithm for task allocation and the Hybrid Ant Genetic Algorithm (HAGA) for scheduling. This is in an attempt to strike an optimal balance between exploration and exploitation in addressing computational complexity in the prevailing hybrid models. Moreover, by tapping into the distinctive advantages of SWO and HAGA, your methodology attempts to optimize adaptability in dynamic workloads while being computationally efficient.

Problem statement

This section focuses on the unique issues that current works frequently encounter. The suggested remedy is also provided. Several of the specific problem statements that already exist include,

Background of existing problems: Authors in [27] to determine Utilizing the suggested fair task distribution scheme, the "Load Balancer (LBer)" serves as a central server in this scenario, distributing incoming jobs among the virtual machines in a fair and balanced manner based on their processing capabilities and present condition. They have contrasted the plan with newer algorithms that achieve load balancing through the use of the Honey bee foraging strategy and particle swarm optimization. The problems employed from these approaches are,

- ➢ Optimization is a topic that requires more research. Though they haven't yet demonstrated optimality, work has been shown to ensure load balancing and yield good outcomes for a range of metrics under various distribution circumstances. They also need to incorporate a VM availability plan. Currently, the fair distribution strategy is used to distribute VM tasks to the other VMs when it is not operating effectively. Nevertheless, these circumstances result in longer overall completion times, which lengthen makespans and lower performance overall.

In this research [40], an expanded version of the queue-based scheduling approach known as IQSLB which also includes load balancing—is developed for handling emergency scenarios. The suggested method reshuffles the jobs in the queue based on the placement value determined by comparing the current state of the VM in the cluster. When a task cannot be adopted by the virtual machine for execution and must be reshuffled with another task in a different queue, the extended IQSLB handles the deadlock issue. The issues employed in this work are,

- ➢ In this instance, the suggested method can be adjusted to manage assignments with dynamic limitations on the specifications of the machines required for their completion.

This load-balancing problem will reduce the network system's overall performance. Therefore, to address the aforementioned issues, the BCSV scheduling method has been optional in this research. The fundamental concept of BCSV is to optimize task dispatch performance utilizing the "Smallest Suffrage Value (SSV), Largest Suffrage Value (LSV), and Criteria Suffrage Value (CSV)" as scheduling factors [41]. Some of the major problems employed in this work are,

- ➢ This is a result of the suggested algorithm failing to take each operational node's higher load limit into account. To enhance the algorithm going forward, they will take into account the workload limit for every working node. They anticipate strengthening the enhanced method to provide efficient task allocation to appropriate working nodes.

The authors of [42], a "resource-aware dynamic load balancing" procedure called DRALBA has been introduced in this study. Run-time adjustments are also made to the VMs' load and computation share. The enhancement of task response time and resource usage is the foremost objective of the DRALBA technique. The suggested method determines the computing share of each VM by using a set of tasks. Next, it chooses a VM whose length is fewer than the computation part of the chosen VM to assign larger size tasks. The suggested scheduler modifies each virtual machine's load at each scheduling choice. The problems existed in this approach are,

- ■ The scheduling of tasks in a multi-cloud context is a challenge. In our upcoming study, we will provide an effective solution to task scheduling. Future research also intends to suggest a resource- and task-aware scheduling method for productive task mapping on virtual machines in CDCs.

Authors in [43] proposed in this study, that they create "two centralized dynamic parallel flow scheduling algorithms, CDPFS and CDPFSMP, for single-path and multi-path, respectively", and augment the "BCube topology with a central master computer" to reduce collisions and enhance bandwidth utilization. By examining the data regarding the condition of the global network, they concentrate on determining the least crowded channel for every flow. Moreover, they assign those pathways to every flow concurrently.

- One avenue for further research could be "to run large-scale simulations of DCNs with central controllers" on a physical machine cluster. Using numerous controllers to handle servers in groups of every level within a single DCN is an alternative approach. Additionally, they suggest analyzing recursively specified structures like DCell and MDCube. In the end, they suggest extending our methods to handle those various topologies of data centers.

Authors in [44] are to introduce an efficient "dynamic load balancing technique (EDLB)" that uses modified particle swarm optimization and convolutional neural networks. It consists of three main modules: the "CNN-based classifier (CBC)", the "fog resource monitor (FRM)", and the "optimized dynamic scheduler (ODS)". The primary goal of EDLB is to use a dynamic real-time scheduling method to achieve LB in an FC context.

- On the other hand, requires stopping the work and saving its existing state. There will also be many migrations and a lengthy rescheduling process if there are no open servers available to host this operation, which will negatively affect system performance. In the future, they hope to address this issue appropriately to provide customers with higher-quality experiences (QoE) and services (QoS).

Research solutions

To tackle the above-mentioned issues using the dynamic resource allocation techniques in parallel and distributed computing using the RRA-SWO offers a workable solution to optimization and VM accessibility difficulties. By using SWO metaheuristic competencies for dynamic resource distribution and regularly allocating tasks, this hybrid strategy exploits resource use. Utilizing VMs for storage reduces total completion instances, improves scheme performance resilience against VM errors, and adapts to varying workload situations to exploit efficiency across computing environments. Using HAGA allocates tasks across mainframes as professionally as likely, assuring effective load balancing and avoiding task refusal because of processor overload. The Least Response Time (LRT) system can be employed to include a load monitoring system and solve the problem of task limit attention for all nodes in distributed computing. To guarantee that tasks are professionally dispersed across nodes within their workload limits, this scheme gathers real-time load data from nodes, examines it centrally, and recruits load balancing. By including historical information and predictive schemes such as LR-HSA, this technique forecasts upcoming workload variations diagonally over multiple clouds. It proactively controls the allocation of resources based on predicted workload, enhancing performance and avoiding overload. This method guarantees effective task mapping onto VMs in Cloud Data Centers (CDCs), consistent resource utilization vigorously, and enlightens the system scalability and receptiveness to differing demands of workload. Includes real-time rearrangement of tasks and resources to uphold optimal across a distributed computing setting via Threshold-based Load balancing. Contributively Fault Tolerant Load Balancing with Priority-based Task Rescheduling, mainly using processes such as Least Recently Used (LRU). Table 1. Represents the notation of symbols.

Proposed work

In this study efficient resource management is serious in current computation to confirm optimal performance and reliability. This research summarizes dynamic scheduling approaches for balancing the load in parallel and distributed schemes, leveraging middleware to coordinate procedures and enable seamless processing and data transfer. Figure 1 represents the overall architecture of this research. The five foremost procedures involved are.

- ➢ Resource Allocation
- ➢ Parallel Task Scheduling
- ➢ Load Monitoring
- ➢ Distributed Computing-based Load Prediction and Adjustment
- ➢ Dynamic Load Balancing

Resource allocation

First, we set up a cloud-sim scenario with 51 virtual machines, 5 data centers, 3 brokers, 2 cloud service providers, and 100 simultaneous users. In a distributed computing system, resource allocation refers to the dynamic distribution of resources, including CPU, memory, and storage, according to availability and demand. Our system uses virtual machines (VMs) for distributed storage, so all nodes can access data more quickly. To guarantee equitable distribution, the Round-Robin Allocation algorithm cyclically allocates resources to tasks. The Sunflower Whale Optimization (SWO) procedure and the hunting performance of whales. In contrast to more

Table 1 Notation of symbols

Symbol	Description
HAGA	Hybrid Ant Genetic Algorithm — combines ACO and GA for parallel task scheduling.
ACO	Ant Colony Optimization — pheromone-based optimization technique.
GA	Genetic Algorithm — optimization using crossover and mutation techniques.
RRA-SWO	Round-Robin Allocation with Sunflower Whale Optimization — for resource allocation.
SWO	Sunflower Whale Optimization — balances exploration and exploitation.
LR-HSA	Harmony Search Algorithm with Linear Regression — predicts workload and adjusts resources.
LRT	Least Response Time — monitors real-time system loads for load balancing
LRU	Least Response Time — monitors real-time system loads for load balancing.
P	Population size — number of search agents, ants, or chromosomes.
Pc	Crossover probability — chance of crossover during reproduction.
Pm	Mutation probability — chance of mutation for variation.
HMS	Harmony Memory Size — the number of solution vectors in memory.
HMCR	Harmony Memory Considering Rate — the probability of choosing from memory.
t	Task runtime or response time.
m	Total number of tasks.
C_i	Computational capacity of task i.
σ	Skewness — used in resource allocation fitness function.
α , β	Coefficients for workload prediction in linear regression.
τ	Pheromone level — indicates the load on a virtual machine.
A	Exploration-exploitation balance — a linear reduction from 2.0 to 0.
I	Maximum number of iterations.
N	Number of search agents.
w	Set of virtual machines for task allocation

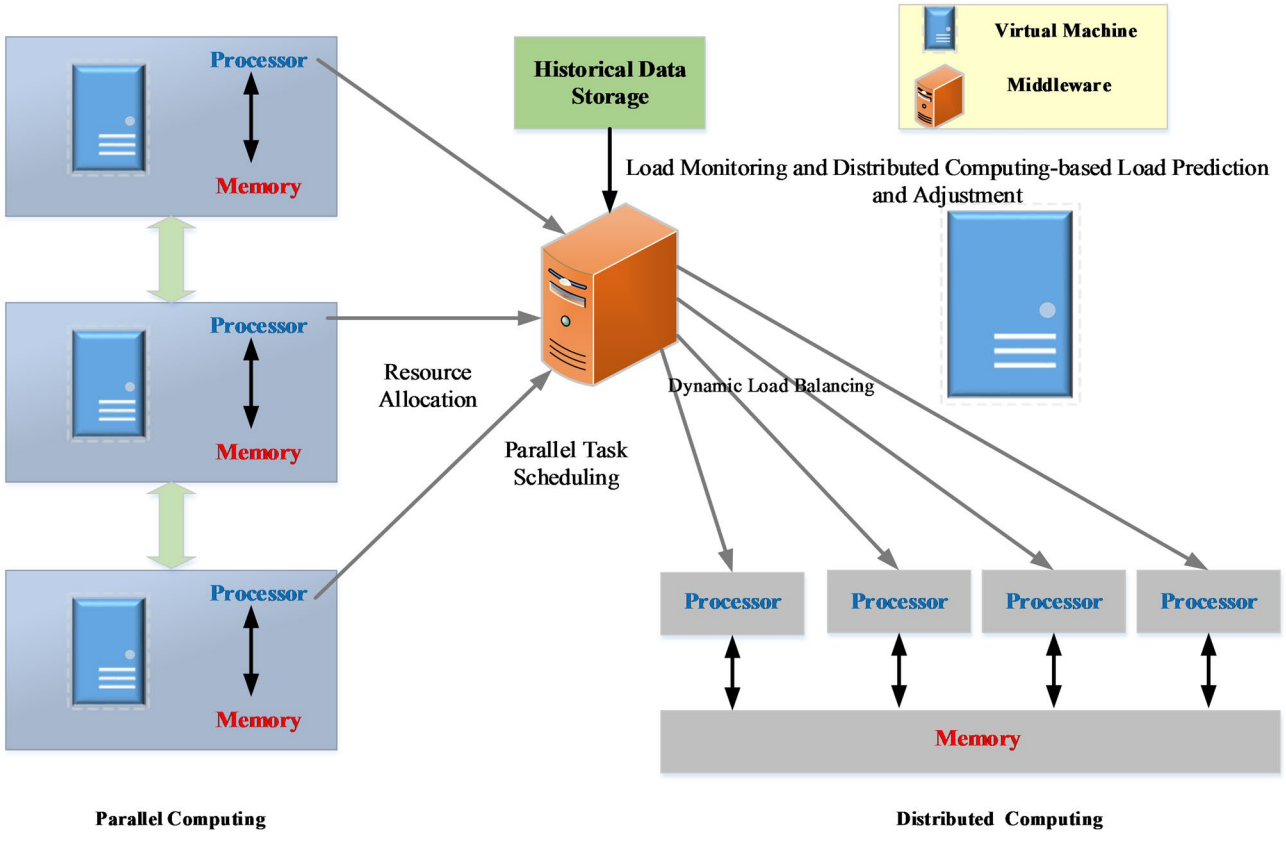

Fig. 1 Overall structure of this study

well-known procedures like particle swarm optimization, it is still comparatively new. Round-Robin Allocation with Sunflower Whale Optimization (RRA-SWO). Combining Sunflower Whale Optimization (SWO) and Round-Robin Allocation (RRA) is designed to improve task allocation efficiency by bridging the adaptability and responsiveness of SWO with the simplicity and fairness of Round-Robin. Round-Robin guarantees fair, cyclic allocation of resources to prevent task starvation, while SWO adapts dynamic resource allocation based on the simulated intelligent hunting nature of whales and the sunflower pollination technique. This hybridization satisfies exploration and exploitation and enables rapid convergence to optimal solutions with responsiveness to workload changes. In comparison to current state-of-the-art hybrid metaheuristics, RRA-SWO is capable of efficiently reducing resource imbalance, minimizing task execution time (makespan), and sustaining scalability under dynamic and large-scale workloads. The interaction among deterministic scheduling and adaptive optimization yields enhanced resource utilization, lowered system latency, and superior overall performance in distributed and parallel computing environments. Assistance from this hybridizing can contain better resource usage, earlier convergence to ideal keys, or more flexibility to surroundings involving dynamic allocation of resources.

a. *Round Robin allocation*

As of right now, CPU task allocation both make extensive use of the Round Robin allocation algorithm. It is specifically designed for operating system sharing and is a preemptive scheduling solution. When the time quantum runs out, the CPU moves on to a new task between the processes. The CPU scheduler selects the first process from the ready queue. After one time quantum, a timer is set to break, releasing the procedure. Two things that happen in the round-robin are listed below.

- ❖ The CPU willfully releases the process if its burst time is shorter than the one-time quantum.
- ❖ Processes that have CPU burst times longer than the time quantum will be pushed to the back of the ready queue.

Flow of Round Robin allocation in Fig. 2. Through the use of a round-robin allocation method, we have attempted to reduce VM performance metrics such as turnaround time and waiting time.

Pseudocode for RRA
→Put each step in order according to when it will burst
→Determine the average of each burst duration
→Assume that the regular burst time equals the quantum number
→Tasks are fewer than quantum facts. Execute these tasks in ascending order.
→Greater than or equal to the quantum number. Place in increasing sequence and execute.
→Execute each process until its time limit expires.

b. *Sunflower Whale Optimization (SWO)*

In virtual machines (VMs), resource allocation is carried out by the assessment of many criteria, including CPU, RAM, MIPS, and skewness. The suggested SFWOA is used to assign the tasks in VM based on the lowest possible cost. SFO and WOA are integrated into the proposed SFWOA. Using metrics like CPU consumption, resource utilization, RAM, MIPS, skewness, and, the suggested SFWOA allocates resources according to fitness values.

- ➢ *Encoding the solution*: The ideal allocation of resource approach in the cloud is defined by the representation of the solution vector. Figure 3 defines the solution vector founded on the values corresponding to each task. Using the suggested optimization algorithm, the task with the lowest value is assigned to the VM. The resource allocation approach is implemented in the cloud by cross-checking each task's worth with that of other users. The number of tasks assigned in the virtual machine is taken into consideration as seven when the solution vector is provided as [1*7].

- ➢ *Assessment of fitness*: To find the best possible solution, the fitness function is calculated. The optimal solution is acknowledged to be the fitness purpose with the lowest value. However, the fitness value is calculated using the following function:

$$f = \sum_{m=1}^{q} B_m + \sum_{n=1}^{p} F_n + (1 - A_n) + G_n \quad (1)$$

In this case, B_m represents the task's runtime, F_n represents the virtual machine's cost, A_n represents the task's resource consumption, G_n represents the skewness, and q stands for the total number of tasks. In this case, A_n and G_n are shown as,

$$A_n = \frac{L_n^v \times Q_n^v \times U_n^v}{L_n^t \times Q_n^t \times U_n^t} \times \frac{W_u}{W_x} \quad (2)$$

$$G_n = \left(\frac{A_n}{A} - 1\right)^2 \quad (3)$$

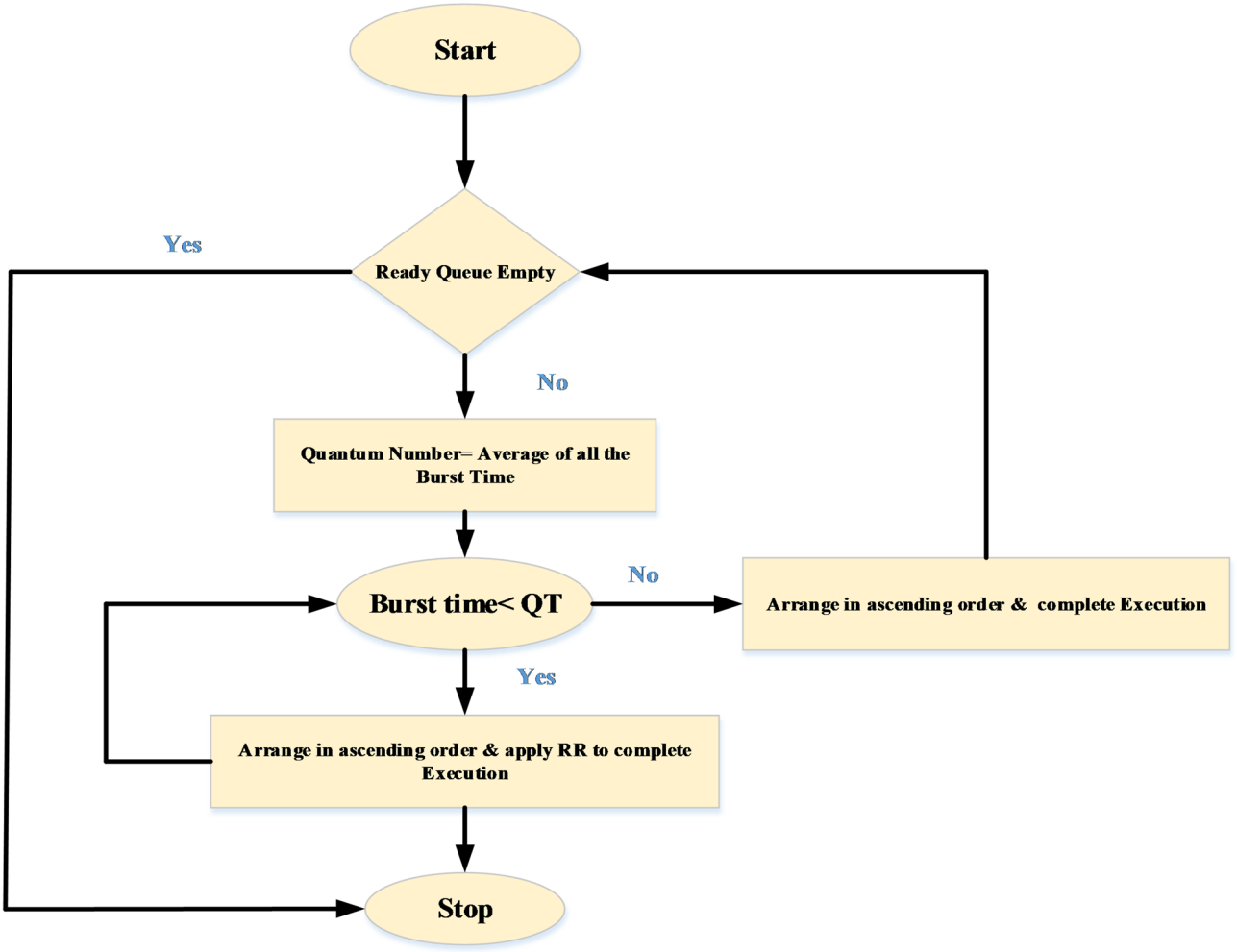

Fig. 2 Flow of Round Robin Allocation

In this case, L_n^t designates the total CPU obtainable in the nth VM, Q_n^t indicates memory obtainable in the nth VM, U_n^v indicates the MIPS utilized by the nth VM, and A specifies the average allocation of resources. Time slot use is shown by W_u, while the maximum total slot is indicated by W_x.

➢ *Sunflower Whale optimization*

The suggested optimization algorithm SFWOA is utilized to allocate resources in cloud computing. The properties of SFO [27] and WOA [40] are integrated to create the proposed SFWOA. In the suggested SFWOA, the resource allocation method is carried out by modifying the WOA update equation with the SFO update equation. The suggested approach is a meta-heuristic optimization method that draws inspiration from the bubble net hunting technique. Whales are magnificent animals that are regarded as the biggest mammals on the planet. Adult whales can reach weights of 180 feet and lengths of 30 m. Because whales must breathe from the ocean's surface, they are thought to be predators because they never sleep. Spindle cells, which are common brain cells seen in humans, are also present in whales. Human social actions, emotions, and judgment are all governed by spindle cells. Given that they have twice as many spindle cells as an adult human, whales are the smartest mammals. Whales' social behavior is what makes them the most fascinating animals. The largest species of baleen whale is known as the humpback whale, and they can live in groups or alone. The allocation of resources method in the cloud model is specifically implemented utilizing the hunting technique of humpback whales. The humpback whale's foraging strategy is known as the "bubble net model." The following is an explanation of the algorithmic phases in the proposed SFWOA:

- *Initialization of the population* Taking into account that there are z whales, M and that their population is initiated as M_τ $(\tau = 1,2,\ldots,z)$.

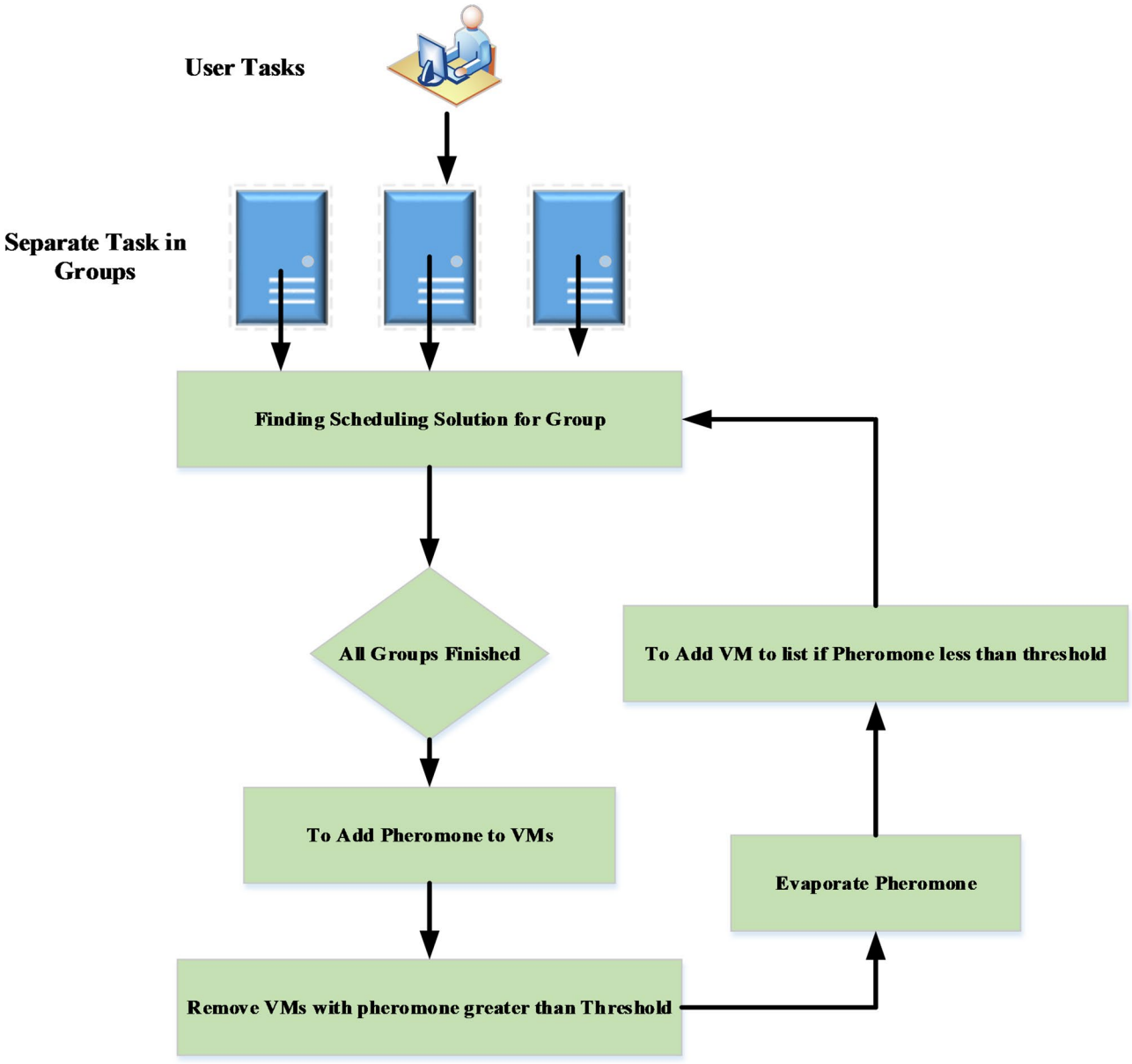

Fig. 3 HAGA for task scheduling

The location vector M^*, the coefficient matrices $\overrightarrow{X}$ and $\overrightarrow{Y}$, And the current iteration i are the variables specified in the suggested algorithm, accordingly.

All search agents' fitness values should be calculated: To find the search agent's best optimal solution, which is specified in Eq. (1), the fitness value is computed.

- *Encircling prey* Once they locate their target, the humpback whales circle them. The goal prey, which is located close to the ideal value, is specified as the current solution at first. When the optimal search agent has been identified, the remaining search agents adjust their position based on that analysis.

Thus, the following equation can be used to depict the encircling behavior of humpback whales:

$$\overrightarrow{\kappa} = |\overrightarrow{Y}.\overrightarrow{M}_i^* - \overrightarrow{M}_i| \tag{4}$$

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_i^* - \overrightarrow{X}.\overrightarrow{\kappa} \tag{5}$$

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_i^* - \overrightarrow{X}.|\overrightarrow{Y}.\overrightarrow{M}_i^* - \overrightarrow{M}_i| \tag{6}$$

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_i^* - \overrightarrow{X}.\overrightarrow{Y}.\overrightarrow{M}_i^* - \overrightarrow{X}.\overrightarrow{M}_i \tag{7}$$

Where κ represents the absolute value, M^* indicates the position vector of the best agent, i indicates the

current iteration, and $\overrightarrow{X}$ and $\overrightarrow{Y}$ denote the coefficient vectors. The humpback whales' updated equation, which takes into account their encircling prey's behavior, is given by the equation above. The Sunflower Optimization Update Equation is expressed as follows:

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_i + \kappa_i \times r_i \tag{8}$$

$$\overrightarrow{M}_i = \overrightarrow{M}_{i+1} - \kappa_i \times r_i \tag{9}$$

By replacing Eq. (9) in Eq. (7), the concluding update equation of the suggested SFWOA is attained. The updated equation that is produced is written as,

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_i^* \left(1 - \overrightarrow{X}.\overrightarrow{Y}\right) - \overrightarrow{X}.\left(\overrightarrow{M}_{i+1} - \kappa_i \times r_i\right) \tag{10}$$

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_i^* \left(1 - \overrightarrow{X}.\overrightarrow{Y}\right) - \overrightarrow{X}.\overrightarrow{M}_{i+1} + \overrightarrow{X}.\kappa_i \times r_i \tag{11}$$

$$\overrightarrow{M}_{i+1} + \overrightarrow{X}.\overrightarrow{M}_{i+1} = \overrightarrow{M}_i^* \left(1 - \overrightarrow{X}.\overrightarrow{Y}\right) + \overrightarrow{X}.\kappa_i \times r_i \tag{12}$$

$$\overrightarrow{M}_{i+1}\left(1 + \overrightarrow{X}\right) = \overrightarrow{M}_i^* \left(1 - \overrightarrow{X}.\overrightarrow{Y}\right) + \overrightarrow{X}.\kappa_i \times r_i \tag{13}$$

$$\overrightarrow{M}_{i+1} = \frac{1}{1+\overrightarrow{X}}\left[\overrightarrow{M}_i^* \left(1 - \overrightarrow{X}.\overrightarrow{Y}\right) + \overrightarrow{X}.\kappa_i \times r_i\right] \tag{14}$$

The final updated equation of the suggested SFWOA resource allocation model is represented by the above Eq. (14). Whereas r_i indicates the sunflower's orientation, κ_i the coefficient directions $\overrightarrow{X}$ and $\overrightarrow{Y}$ are shown as, and the steps of the sunflower towards the direction r_i

$$\overrightarrow{X} = 2\overrightarrow{\rho}.\overrightarrow{c} - \overrightarrow{\rho} \tag{15}$$

$$\overrightarrow{Y} = 2.\overrightarrow{c} \tag{16}$$

where $\overrightarrow{c}$ is the random integer that falls between [0, 1], and $\overrightarrow{\rho}$ is a linear function that declines from 2 to 0 based on the epochs in both the exploration and exploitation phases. Additionally, the sunflower's direction (r_i) and the steps (κ_i) in that direction are shown as,

$$r_i = \frac{M^* - M_i}{\| M^* - M_i \|}, i = 1,2,\ldots\gamma \tag{17}$$

$$\kappa_i = \mu \times N_i(\| M_i + M_{i-1} \|) \times \| M_i + M_{i-1} \| \tag{18}$$

Where $N_i(\| M_i + M_{i-1} \|)$ indicates the probability of pollination and M indicates the constant range that requires the "inertial displacement of plants".

Model of bubble net assault: The bubble net concept is used by the humpback whale to assault its prey. Nonetheless, the shrinking process and the spiral location are two distinct tactics that are used to specify the performance of the bubble net in whales.

i. *Shrinking Mechanism*

Utilizing Eq. (14), the shrinking behavior in this mechanism is achieved by lowering the value of $\overrightarrow{\rho}$. As a result, the fluctuation range of $\overrightarrow{X}$ lowers by utilizing $\overrightarrow{\rho}$. Here, ρ drops from two to zero dependent on the number of repetitions, and $\overrightarrow{X}$ is the random value that falls in the range $[-\rho, \rho]$. The newly updated location of the search agent is quantified among the starting location and the current best agent when the random range for $\overrightarrow{X}$ is set to $[-1, 1]$.

ii. *Spiral update position*

The spiral update position is used to calculate the distance between the prey location at (M^*, J^*) and the humpback whale located at (M, J). Furthermore, the helix-shaped movement generates a spiral equation that can be stated as follows between the location of the whale and the prey:

$$\overrightarrow{M}_{i+1} = \overrightarrow{\kappa'}.e^{ah}.\cos(2\pi h) + \overrightarrow{M}_i^* \tag{19}$$

Here, h is a random number that falls between $[-1, 1]$, marks the element-by-element multiplying, a stand for the constant feature, and $\overrightarrow{\kappa'} = \overrightarrow{M}_i^* - \overrightarrow{M}_i$ signifies the distance between the ith whale and to prey. The decreasing mechanism or spiral model measured typically used to update the location of whales is represented as follows:

$$\overrightarrow{M}_{i+1} = \begin{cases} \frac{1}{1+\overrightarrow{X}}\left[\overrightarrow{M}_i^*\left(1 - \overrightarrow{X}.\overrightarrow{Y}\right) + \overrightarrow{X}.\kappa_i \times r_i\right] \\ \overrightarrow{\kappa'}.e^{ah}.\cos(2\pi h) + \overrightarrow{M}_i^* \end{cases} \tag{20}$$

where x is the random quantity that falls between 0 and 1.

- *Search for Prey*: During the exploration phase, humpback whales randomly look for prey based on the positions of nearby whales. Therefore, relatively than consuming the best search agent, the vector $\left|\overrightarrow{X}\right| > 1$ highlights the exploration stage to change the location of the search agent depending on the arbitrarily designated search agent. In the exploration phase, the mathematical model that postulates how to hunt the prey is expressed as,

$$\overrightarrow{\kappa'} = |\overrightarrow{Y}.\overrightarrow{M}_d - \overrightarrow{M}| \tag{21}$$

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_d - \overrightarrow{X}.\overrightarrow{\kappa} \quad (22)$$

$$\overrightarrow{M}_{i+1} = \overrightarrow{M}_d - \overrightarrow{X}.\left[|\overrightarrow{Y}.\overrightarrow{M}_d - \overrightarrow{M}|\right] \quad (23)$$

Where the "random position vector" chosen from the beginning population is denoted by $\overrightarrow{M}_d$. The search agent modifies the location based on the optimal solution chosen search agent at each iteration. The search agent is randomly chosen to inform its location when $\left|\overrightarrow{X}\right| > 1$, and the best solution is employed to update the search agent's location when $\left|\overrightarrow{X}\right| < 1$.

- *Assessment of fitness*: "The search agent with the best fitness value" is deemed to be the best option, and the fitness function is assessed based on the positions of the search agents.

Equation (1) specifies the function that was used to calculate the fitness value.

- *Termination* Until the optimal solution is found or the termination requirements are met, the aforementioned procedures are repeated. The proposed SFWOA's pseudo code is represented by Algorithm 1.

Parallel task scheduling

A Hybrid Ant Genetic Algorithm (HAGA) is used for parallel task scheduling, minimizing processor usage by allocating tasks efficiently among many processors. It is a hybrid of the evolutionary strategy of Genetic Algorithms (GA) and exploration properties of Ant Colony Optimization (ACO) that overcomes the shortcomings of conventional scheduling algorithms. To advance load balance and task allocation, HAGA combinations the evolutionary ideologies of genetic procedures with the examining competencies of ant colony optimization. The HAGA technique allocates tasks from the VM to the processor with the lowermost existing load afterward monitoring processors for workload. Numbers are used to encode chromosomes in task scheduling. Every chromosome has a workable answer. Because crossover and mutation procedures need a lot of processing power and a data center can have thousands of processes, the GA algorithm takes a while to find the correct answer. To reduce the size of the solution space, we separated the jobs into smaller groups to handle this problem. The HAGA algorithm utilizes the ACO algorithm to monitor previously scheduled jobs that are still pending completion on the server. This aids in virtual machine detection that is loaded. Overloaded virtual machines are not chosen for the subsequent batch of tasks. This method narrows the solution space and increases the likelihood

Initialize the population
Compute f
Select the leading search agent. M^*
While $(i < I)$*; Maximum allowed iterations is I ;*
For every search agent
Update ρ, X, Y, h *and* x
If $1(x < 0.5)$
  If $2(|X| \geq 1)$
    Utilizing eq (14) update the search agent's position
  Else If $2(|X| \geq 1)$
    Select the random search agent. (M_d)
      Applying Eq (23) updates the current search agent's position.
    End if2
  Else If $1(x \geq 0.5)$
      Modify the position of the search agent by Eq. (19)
  End if1
    End for
  Update M^* *with the best solution*
$i = i + 1$
End while
Return m^*

Algorithm 1.

of reaching the global optimum. The HAGA proposal's details are shown in Fig. 3.

➢ Encoding

Tasks are represented as chromosomes, with one gene for each task mapped to a virtual machine (VM). Chromosomal encoding is done with real numbers, and tasks are broken into smaller groups to decrease complexity. Numerous encoding techniques are applied to various situations, including real number, binary, and symbol encoding. While symbol encoding reflects a solution space according to certain symbols, binary encoding translates a solution space to a set of strings. Similar to this, some real numbers indicate that a solution space is represented by real number encoding. The VM on which the task will be scheduled is represented by the chromosome's gene value, and the length of the chromosome equals the entire number of tasks. In this approach, chromosomal encoding is done using real numbers, and tasks are broken down into smaller groups.

A collection of virtual computers of size w is assigned to each group. Assuming that δ is a collection of groups, $\delta = \delta_1, \delta_2, \delta_3 \ldots \delta_\beta$. The number of tasks T_N assigned to a group δ_ϑ

can be expressed as:

$$T_N = \frac{n}{\nu} \tag{24}$$

In this case, ν represents the total number of groups and n the total number of tasks. A set of chromosomes α can be expressed as $\alpha = \{\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_Z$ if there are z chromosomes/solutions. The length of the chromosomes will match the total tasks (T_N) for a group of k. For instance, basic chromosome where a group is given 5 tasks and 3 virtual computers.

➢ Fitness

The fitness function considers every solution quality regarding task finishing time and load balance. Low values of the fitness function specify more optimal solutions. Crossover and mutation procedures are carried out in a genetic algorithm at each iteration's stage, producing new solutions. Verifying the solution's viability is essential. The solution's quality and viability are assessed using the fitness value. Fitness value makes the span of a solution in this procedure. The better the solution, the lower the fitness value.

➢ Selection

A roulette wheel strategy picks the best chromosomes for crossover based on their fitness values. The most fit solutions are chosen during the selection process, while the less fit ones are eliminated. A variety of selection techniques are employed, such as sorting, roulette, and tournament selection. We pick using a roulette system. The selection probability for every chromosome is determined in the first stage. The likelihood of selection P for a chromosome α_j is calculated by,

$$P_j = \frac{MS_j}{\sum_{i=0}^{z} MS_j} \tag{25}$$

We split a circular disk into z pieces, and the width of each part is proportionate to the selection probability, once the selection probabilities have been calculated. The final step involves creating a random number and choosing a chromosome from the area that the random number falls on.

➢ Crossover

A two-point crossover generates offspring by exchanging parts of two parent chromosomes to provide diversity in the solution pool. A crossover is utilized to produce additional offspring from individuals chosen during the selection process. There are several crossover strategies employed in the literature. Two-point crossover is employed. First, chromosomes are split into two cuts using this approach. Cuts' location on the chromosomes can be chosen at random to be,

$$[\varsigma 1, \varsigma 2] = rand(2) \times (TK - 1) + 1 \tag{26}$$

Where $\varsigma 1$ and $\varsigma 2$ are the two cut locations, and $rand(2) \times (TK - 1) + 1$ yields a random value between 1 and the chromosome's whole length. Next, two new chromosomes are created by swapping the values of the two chromosomes between these incisions.

➢ Mutation

Mutation avoids premature convergence by exchanging genes at random. A novel mutation strategy ensures equal opportunity for all positions to mutate. New features in chromosomes are introduced via mutation operations. Included in it are exchange, uniform, and swap mutations. These methods include selecting a random position in the solution and substituting the gene value of that site with the gene value of another position. We present a novel mutation technique that guarantees an equal probability of a new feature at each place. Using this method, genes are switched at the beginning and finishing places once they are first determined. The position value from the start is increased and decreased from

the last in the subsequent iteration, and the values of the genes are then switched at these positions.

➢ Add Pheromone

Following task scheduling, pheromone values are updated to reflect VM loads. Overloaded VMs are assigned higher pheromone values, which discourage additional task assignments and even out the load. This dynamic task retains the scheme functioning at peak competence by averting any one CPU from being overworked. The load to each processor is recalculated ensuring each task competently adjusts to everchanging workload situations. This technique increases system scalability and sensitivity to varying task demands from VMs and additional applications to refine CPU efficiency and utilization.

$$f_{jk} = \sum {}^{T_{Nk}}_{i=0} \epsilon_i \tag{27}$$

Where T_{Nk} The total tasks given to a group k is represented by ϵ_i, which represents the task's running duration. Mutation process. VM pheromone is dependent on tasks that are planned for it. A virtual machine that exhibits a high pheromone level is loaded and will impede the execution of additional tasks. This facilitates virtual machine detection when loaded.

➢ VM selection

Virtual machines are sorted by their pheromone levels. Overloaded VMs are less likely to be selected for future tasks, optimizing resource allocation. Virtual machines are assigned to each group. δ_i of jobs to reduce solution space. To identify loaded virtual machines, a pheromone is applied when jobs are scheduled on them. The method selects a set of w virtual computers from m virtual machines by first sorting all virtual machines based on pheromone.

$$\omega = \gamma * m \tag{28}$$

In this case, the constant γ has a range of 0.1 to 1. Virtual computers that are overloaded are unlikely to be chosen for the upcoming batch of assignments. This contributes to a reduction in solution space and workflow execution time.

$$\varrho = \frac{\sum_{i=0}^{T_{Nk}} \epsilon_i}{T_{Nk}} \tag{29}$$

Where ϵ_i is the task's execution time and T_{Nk} is the total number of tasks allocated to the group k. A virtual machine's pheromone evaporation can be written as:

$$f_i = f_i - \varrho \tag{30}$$

The pheromone that is deposited in virtual machine i is called f_i, and its value should be greater than zero. Pheromone therefore naturally drops as work on virtual machines is completed. The virtual machine is once more chosen in solution space for the subsequent set of tasks after the pheromone evaporates.

Load monitoring

Load monitoring offers the data essential for well-informed load balancing results by incessantly following performance systems of measurement and resource utilization. By means of the Least Response Time (LRT), the Centralized Load Monitor process gathers and collects real-time load data from all nodes, analyzing it and originating load balancing processes as essential.

$$RT_i = \mathcal{S}_{end_i} - \mathcal{S}_{start_i} \tag{31}$$

Where RT_i is the response time for the task i, $\mathcal{S}_{end_i}$ is the task end time of i, and $\mathcal{S}_{start_i}$ is the task i start time.

$$\mathcal{U}_j = \sum {}^{n}_{i=1} \left(\frac{RT_i}{\mathcal{O}_i} \right) \tag{32}$$

Where $\mathcal{U}_j$ is the load on mode j. $\mathcal{O}_i$ is the computational capacity of task i, and n is the total number of tasks.

$$\Delta \mathcal{U} = \mathcal{U}_{max} - \mathcal{U}_{min} \tag{33}$$

Where $\Delta \mathcal{U}$ is the variance among the maximum load $\mathcal{U}_{max}$ and the minimum load $\mathcal{U}_{min}$ minimum load across all the nodes.

By assembling a load system of measurement from parallel and distributed systems, this central monitoring scheme offers data for dynamic load alterations.

Distributed computing-based load prediction and adjustment

This method predicts dissimilarities in load and alters resources consistently by applying past data and predictive schemes. The chance of overload is reduced by this proactive load balancing. Utilizing historical load data, the Linear Regression Prediction algorithm predicts upcoming patterns and kinds of proactive resource alterations. Utilizing historical load data, HSA iteratively

advances the harmony (performance) of the scheme to enhance the linear regression model parameters, like coefficients and intercepts. It has the facility to achieve intricate nonlinear associations in information and adaptively adapt model parameters to improve the efficiency of load prediction and resource allocation. The middleware brands sure that together processing and storing resources are pre-allocated based on expected load changes by accumulating historical information, utilizing Harmony Search Algorithm with Linear Regression (LR-HSA) for future load forecasts, and adapting resource allocation. It does this by using several very critical criteria to optimize resource allocation. The number of solution vectors saved depends on Harmony Memory Size (HMS). Harmony Memory Considering Rate is usually between 0.7 and 0.95; it is the probability of taking values from the harmony memory, where values are chosen from that very memory. Small changes for solution variables are controlled using a Pitch Adjustment Rate to increase variety. The best answers found along the way are finally stored in the Harmony Memory (HM). VM enables distributed storage alterations.

Step 1 Once the parameters are initialized, define the objective function. The desired variation in the problem is denoted by $f(x)$, and x is a potential solution that comprises N choice variables, namely X_i and $\zeta_{ki} \leq \mathrm{X}_i \leq \cup_{ki}$. The lower and upper bounds for every value are ζ_{ki} and $\cup_{ki}$. In addition, this phase initializes the HSA's parameters.

Step 2 Set the harmony memory to its initial value.

Initialization is carried out as follows:

Randomly create a $2 \times HMS$ harmony memory from a uniform distribution in the interval $[\zeta_{ki} \cup_{ki}]$ (where i=1,2, 3, n).

Determine each potential solution's suitability in the harmony memory, then arrange the findings in ascending order.

$[\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_{HMS}]$ generates the harmony memory.

Step 3 Improvisation

To create fresh harmony is the aim of this step. Based on the following rules, the new harmony vector $= \{\mathrm{X}'_1, \mathrm{X}'_2, \ldots, \mathrm{X}'_n\}$ is created. First, create $\mathcal{H}_1$ and $\mathcal{H}_2$ at random within a normal distribution of that range [0,1].

If $\mathcal{H}_1 < HMCR$ and $\mathcal{H}_2 \geq$ PAR, then $\mathrm{X}'_i = \mathrm{X}'_i$

If $\mathcal{H}_1 < HMCR$ and $\mathcal{H}_2 <$ PAR, then $\mathrm{X}'_i = Rnd\,(\mathrm{X}_i^{gbest} - ad,\ \mathrm{X}_i^{gbest} + ad)$, where X_i^{gbest} is the dimension of the best candidate solution and the ad is an arbitrary distance bandwidth (BW).

If $\mathcal{H}_1 \geq HMCR$, then $\mathrm{X}'_i = Rnd(\zeta_{ki}, \cup_{ki})$

Step 4 Update harmony memory.

The new harmony vector will replace the worst harmony in the HM if its fitness is greater than the worst harmony's.

Step 5 Check the stopping criterion.

Terminate when the iteration is reached.

Based on the load in the most recent period, a linear regression model can forecast the load in the following time. The general linear regression model can be used to generate linear expressions with time variations.

$$L_i = \eta_1 + \eta_2 T_i \tag{34}$$

In this case, L_i stands for the load value at that moment, T_i for the period's duration, and 1 and 2 for the regression parameters that need to be resolved.

The most appropriate regression parameters should yield the best estimation results, allowing the sum of squares. ψ between the true value L_i and the estimated value L'_i to approach zero.

$$\psi = \sum_{i=1}^{m} (L'_i - L_i)^2 \tag{35}$$

When (35) and (36) are combined to provide the partial derivatives of $\eta_1 \eta_2$, the two partial derivative equations equal 0, as indicated by (37):

$$\begin{cases} \frac{\partial \psi}{\partial \eta_1} = 0 \\ \frac{\partial \psi}{\partial \eta_2} = 0 \end{cases} \tag{36}$$

Equations (36) and (37) can be utilized in (36), and the equation in simplified form is produced, as illustrated in (37),

$$\begin{cases} m\eta_1 + \eta_2 \sum_{i=1}^{m} T_i = \sum_{i=1}^{m} L_i \\ \sum_{i=1}^{m} T_i \eta_1 + \sum_{i=1}^{m} T^2{}_i \eta_2 = \sum_{i=1}^{m} T_i L_i \end{cases} \tag{37}$$

The equations can be solved using the Clem rule, and the parameters of the regression. η_1 and η_2 can be found in (38),

$$\begin{cases} \eta_1 = \frac{\sum_{i=1}^{m} T^2{}_i \sum_{i=1}^{m} L_i - \sum_{i=1}^{m} T_i \sum_{i=1}^{m} T_i L_i}{m \sum_{i=1}^{m} T^2{}_i - \left(\sum_{i=1}^{m} T_i\right)^2} \\ \eta_2 = \frac{\sum_{i=1}^{m} T^2{}_i L_i - \sum_{i=1}^{m} T_i \sum_{i=1}^{m} L_i}{m \sum_{i=1}^{m} T^2{}_i - \left(\sum_{i=1}^{m} T_i\right)^2} \end{cases} \tag{38}$$

The linear regression model can dynamically modify regression parameters η_1 and η_2, adjusting for variations in the load based on load data collected over a similar time.

Input: *Historical load data L_h, real-time load datac, task resource requirements R_t, and system configuration S_c.*
Output: *Optimized load distribution and resource allocation.*

Initialization
Set the parameters for the HAS (HMS, HMCR, PAR, Iteration limit (I_{max}))
Gather historical load data (L_h), initialize the harmony memory (HM)
Continuously monitoring real-time load (L_h) using (LRT)
Use the historical load data (L_h), to predict future load (L_p)
For each iteration i (up to I_{max})
Generate new harmony
Evaluate fitness
Update harmony memory
if L_n>Threshold using (LRU)
continuously repeat as new real-time data $L_r$is gathered and workload evolves.

Algorithm 2. Load prediction and adjustment algorithm

Algorithm 2 Load Prediction and Adjustment Algorithm (Based on distributed computing), makes dynamic changes in resource allocation with the help of the Harmony Search Algorithm, making predictions for future workloads by applying linear regression to maximize load balancing. Although the Harmony Search Algorithm with Linear Regression (LR-HSA) performs well in workload fluctuations prediction and dynamic resource allocation in real-time, it is important to compare it with sophisticated predictive models such as Long Short-Term Memory (LSTM) and Reinforcement Learning (RL)-based techniques. LSTM networks are well suited to handle sequential data and acquire long-term dependencies to make precise forecasts under complex and dynamic workloads. However, LSTMs are prone to requiring extensive training data, significant computational resources, and selective parameter tuning, factors that can limit their use in real-time dynamic environments. In the same way, RL-based techniques learn the best policies under dynamic conditions through exploration and exploitation by trial and error. Although highly flexible, RL-based techniques are prone to lengthy exploration times and high computational complexity before they arrive at efficient solutions. By contrast, the LR-HSA hybrid combines the linear regression's ease and efficiency with the global searching ability of Harmony Search, such that it presents a computationally efficient option for real-time load prediction. It can present fast and adaptive predictions with negligible computational overheads and fits well into the needs of distributed and parallel systems. This sacrifice of accuracy for efficiency is the basis for the choice to use LR-HSA for workload prediction in this research.

Dynamic load balancing

Real-time task and resource rescheduling is a factor of dynamic load balancing, it keeps performance at its finest. Utilizing threshold-based load balancing, the Dynamic Workload Balancer procedure uninterruptedly modifies task distribution to assurance a balanced load. The middleware assurances real-time task mobility among processors and nodes by maintaining the present loads on all nodes and processors, redistributing tasks as essential, and repeating the cycle of adjustment and monitoring. Merging priority-based task rescheduling with fault-tolerant load balancing, particularly when retaining approaches like Least Recently Used (LRU). This technique assurances that vital tasks are rapidly redistributed to servers that are up and running, decreasing downtime and exploiting resource utilization. If $\mathcal{U}_j > \theta$, then redistribute tasks to maintain $\mathcal{U}_j \leq \theta$ .Where $\mathcal{U}_j$ is the load on the node j and θ is the predefined load threshold.

$$\Upsilon\left(\mathrm{T}_i\right) = \frac{W_i}{\sum_{j=1}^{m} W_j} \tag{39}$$

Where $\Upsilon\left(\mathrm{T}_i\right)$ is the priority of task i, W_i is the weight of task I, and m is the total number of tasks.

The system upholds optimal performance and diminishes the effect of server inaccessibility by prioritizing tasks given their priority and earlier procedure patterns. Since this addition, system reliability has increased and the entire user experience is enhanced by diminishing delays and assuring persistent service availability from platforms such as VM.

The proposed system employs a hybrid of the Least Response Time (LRT) method and the Harmony Search Algorithm with Linear Regression (LR-HSA) to monitor real-time loads and make predictive load adjustments. In cases of abrupt jumps in workload or failures in resources, the system takes a threshold-based approach in initiating proactive rescheduling of tasks. Once the load on a node is found to exceed a specified limit, tasks are rescheduled dynamically to idle virtual machines

Table 2 System specifications

Software Specifications	OS	Windows 11-(64-bit)
	Tool	Net- beans 12.3
Hardware Specifications	RAM	4GB
	Hard Disk	500GB

(VMs) to effectively distribute the load. This limit-based technique in addition to the LRT load monitoring mechanism helps keep task migration overhead low as it reschedules only critical tasks.

The system further has a priority-based rescheduling of tasks feature so that important tasks receive preference during rescheduling. To further minimize migration overhead, overloaded VMs are highlighted with higher pheromone levels in the Hybrid Ant Genetic Algorithm (HAGA), lowering their chances of further receiving tasks. The predictive feature of the Harmony Search Algorithm also predicts workload peaks so that anticipatory resource adjustment is made ahead of reaching critical levels. During a resource crash, the system instantly redistributes tasks to provide VMs with the Least Recently Used (LRU) approach so that the system remains stable and there is minimal downtime. The combination of predictive, real-time, and reactive methods strongly resists task migration overhead and yet offers peak system performance with dynamic changes in workload.

Experimental results

This section describes the experimental investigation that was done to assess the suggested method's performance. In addition, this part is divided into three subsections: simulation setup, comparative analysis, and research summary.

Simulation setup

In the experimental setting, CloudSim and NetBeans 12.3 were used, and Windows 11 (64-bit) was the OS installed, which had 500 GB of hard drive space and 4 GB of RAM. It simulated an environment that served as a distributed computing system with 51 VMs, 5 data centers, 3 brokers, 2 cloud service providers, and 100 concurrent users. Resource allocation optimization through a study utilized Round-Robin Allocation with Sunflower Whale Optimization (RRA-SWO), Hybrid Ant Genetic Algorithm (HAGA) for parallel task scheduling, Least Response Time (LRT) for monitoring loads, and Harmony Search Algorithm with Linear Regression (LR-HSA) for predictive load adjustments. The Least Recently Used strategy is also applied in the case of dynamic load balancing. The evaluation has focused on key performance parameters, which include memory utilization rate, task completion ratio, average response time, throughput, and packet delivery ratio. To further ascertain the framework, future upgrades will incorporate real-world workload, test with larger-size datasets, and explore variables such as latency and energy consumption for dynamic scenarios (Table 2).

To get the best outcomes in this research, the parameters of the proposed hybrid metaheuristics were set with much care. Table 3 presents the parameter settings of RRA-SWOA, HAGA, and LR-HSA along with their optimal values. After extensive experimentation, these parameters were determined by balancing solution quality with computing efficiency.

Comparative analysis

This section compares the proposed approach to many existing ones, such as the Deep-Learning (DL) [28], Balanced Criteria Suffrage Value (BCSV) [41], Reinforcement Learning Fog Scheduling Algorithm (RLFSA) [26], and evaluates its efficacy using performance metrics elaborately explained below:

a. *Task Success Rate*

Using this Eq. (40), determines the proportion of successfully finished tasks relative to the total number of activities attempted. It serves as a gauge for the dependability and efficiency of the system in carrying out tasks. To solve this equation, we take into account the following crucial factors:

$$TSR = \frac{T_{Success}}{T_{Total}} \times 100 \tag{40}$$

Table 3 Simulation parameters

Metaheuristic	Parameter	Description	Optimal Value
RRA-SWOA	Population Size (N)	Number of search agents	50
	Number of Iterations (I)	Maximum number of iterations	100
	Exploration-Exploitation Balance (A)	Linear reduction from 2.0 to 0	-
HAGA	Population Size (P)	Number of ants/chromosomes	30
	Crossover Probability (Pc)	Probability of crossover operation	0.8
	Mutation Probability (Pm)	Probability of mutation operation	0.05
LR-HSA	Harmony Memory Size (HMS)	Number of solution vectors in memory	20
	Harmony Memory Consideration Rate (HMCR)	Probability of choosing from memory	0.9

$T_{Success}$ represents the number of successful tasks; T_{Total} indicates the total number of tasks.

The success rate of the task quantifies the ratio of completed tasks to the number of attempts. The higher the success rate, the better the system's reliability and efficiency. The suggested approach performs better compared to other methods (DL, BCSV, RLFSA), which demonstrates its efficiency in managing dynamic workloads and reducing task failures. Figure 4 represents the task success rates (%) of three procedures: DL, BCSV, RLFSA, and a suggested technique, plotted against the number of tasks. Firstly, DL shows the lowermost success rate at 46% for 10 tasks, progressively increasing to 71% for 100 tasks. BCSV shows the lowermost success rate at 50% for 10 tasks, progressively increasing to 75% for 100 tasks. RLFSA starts at 55% for 10 tasks and influences 80% for 100 tasks, reliably executing better than BCSV, and DL across all task counts. The suggested techniques outperform both procedures from the start, initially at 58% for 10 tasks and progressively increasing to 95% for 100 tasks, representing its superior presentation across varying task difficulties and measures.

b. *Average Response Time*

Equation (41) calculates the meantime taken to reply to all tasks. It means the system's efficacy in processing tasks and is vital for measuring performance below varying loads. Wherever n is the number of tasks.

$$ART = \frac{\sum_{i=1}^{n} RT_i}{n} \tag{41}$$

The response time average measures the system's performance in task processing. Smaller response times indicate higher processing speeds. The suggested method consistently produces the minimum response times, proving effective resource utilization with reduced latency against available techniques. Figure 5 demonstrates the average response times (seconds) against the number of tasks. Primarily, DL shows the uppermost response times, starting at 75 s for 10 tasks and increasingly increasing to 99 s for 100 tasks. BCSV shows the uppermost response times, starting at 73 s for 10 tasks and increasingly increasing to 97 s for 100 tasks. RLFSA shows reasonable response times, initial at 70 s for 10 tasks and attainment at 95 s for 100 tasks, reliably acting better than across all task counts. The suggested methods prove the lowest response times throughout, starting at 40 s for 10 tasks and increasing to 65 s for 100 tasks, representing its efficiency in handling variable task loads.

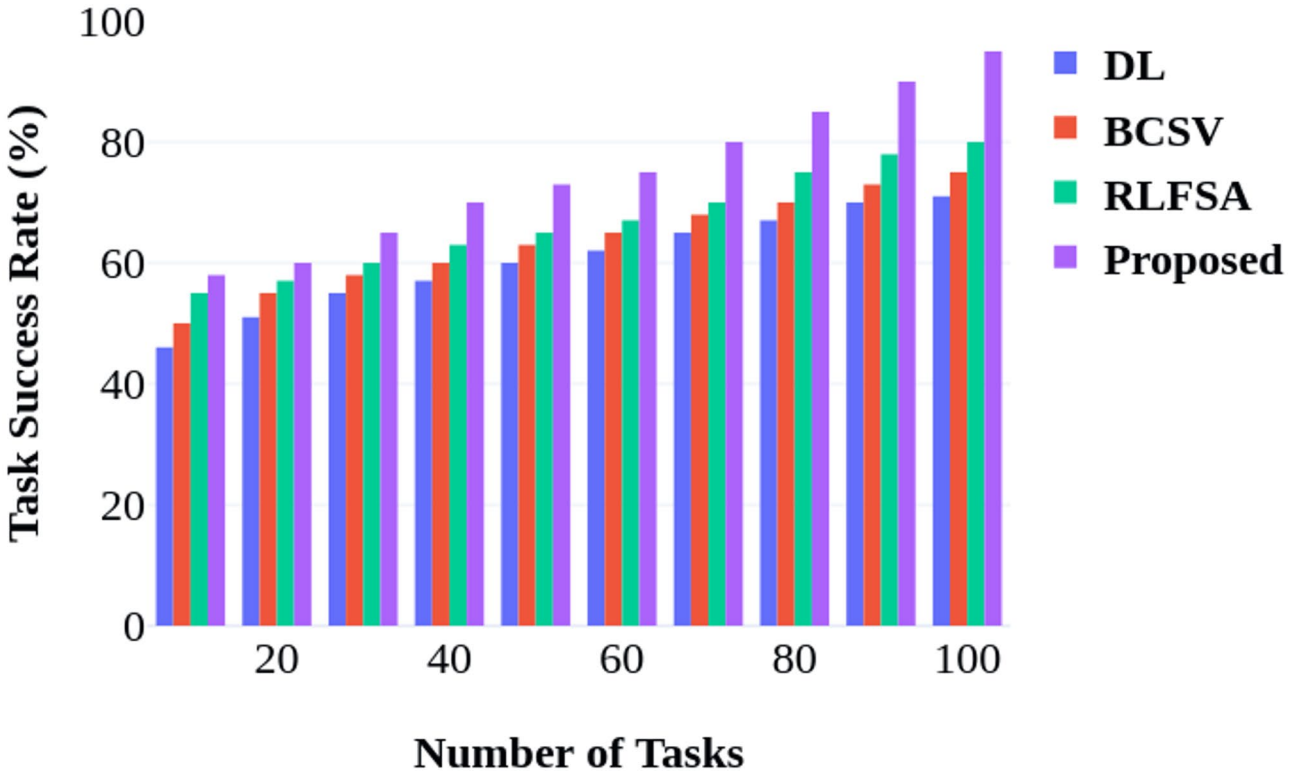

Fig. 4 Task Success Rate

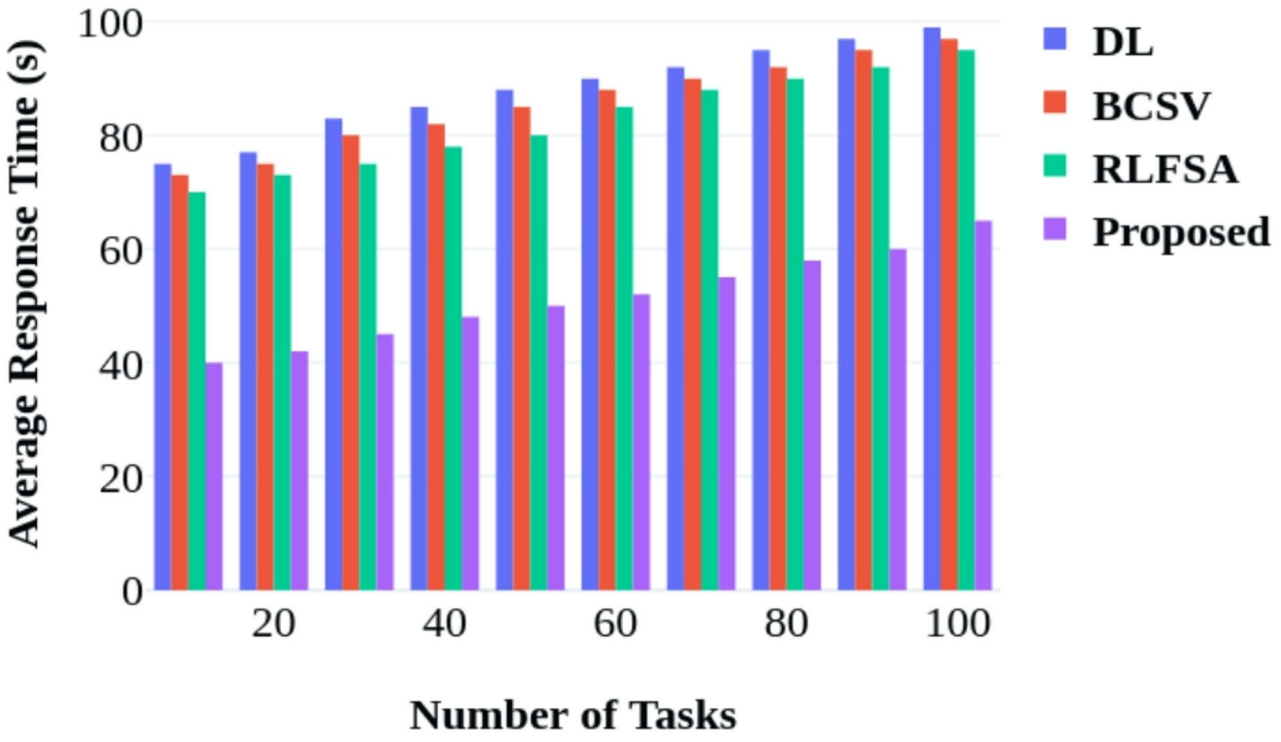

Fig. 5 Average Response Time

c. *Memory Utilization Rate*

Equation (42) adjusts the proportion of memory being used out of the total accessible memory. Greater memory utilization agrees effectual use of resources, but extreme use might lead to possible problems with resource contention. U describes the used memory; $\propto$ and designates the total available memory.

$$MUR = \left(\frac{U}{\propto}\right) \times 100 \tag{42}$$

The measure of the memory utilization rate defines how well the system resources are utilized. Though better utilization means higher resource use, higher use can also create contention problems. The presented strategy keeps optimal memory utilization by achieving efficiency with appropriate balancing against overloading. Figure 6 depicts the memory utilization rates (%) with several tasks. At first, BCSV displays the lowest memory use rate, preliminary at 30% for 10 tasks and regularly growing to 54% for 100 tasks. BCSV displays the lowest memory use rate, preliminary at 33% for 10 tasks and regularly growing to 50% for 100 tasks. RLFSA displays moderate memory procedure, starting at 40% for 10 tasks and success at 60% for 100 tasks, usually developed than BCSV and DL across all task counts. The suggested techniques prove the highest memory utilization rates throughout the range, early at 58% for 10 tasks and increasing to 80% for 100 tasks, demonstrating its greater resource demand associated with the other procedures.

d. *Throughput*

Equation (43) measures the rate at which tasks are finished over a specific period. It is a significant measure for assessing the system's size to handle workload and its overall efficiency.

$$T = \left(\frac{\pi}{\phi}\right) \times 100 \tag{43}$$

π defines the number of tasks completed; ϕ - total time

Throughput is the number of tasks accomplished within a specific time. Increased throughput indicates improved performance and utilization of resources. The method suggested here attains the maximum throughput, reflecting its ability to process tasks at high speeds and support high workloads efficiently. Figure 7 represents the throughput (%) against the number of tasks. To begin with, DLBCSV starts with a throughput of 40% for 10 tasks and rises to 65% for 100 tasks, viewing a steady

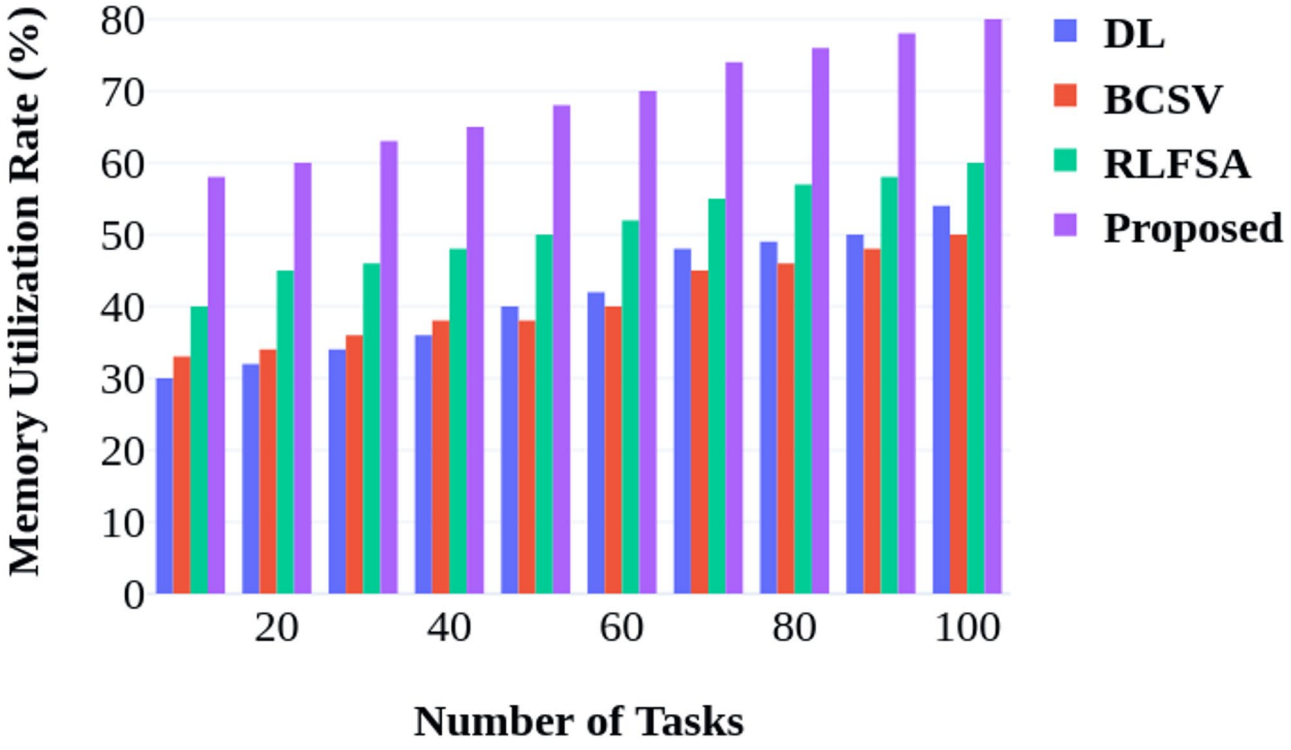

Fig. 6 Memory utilization rate

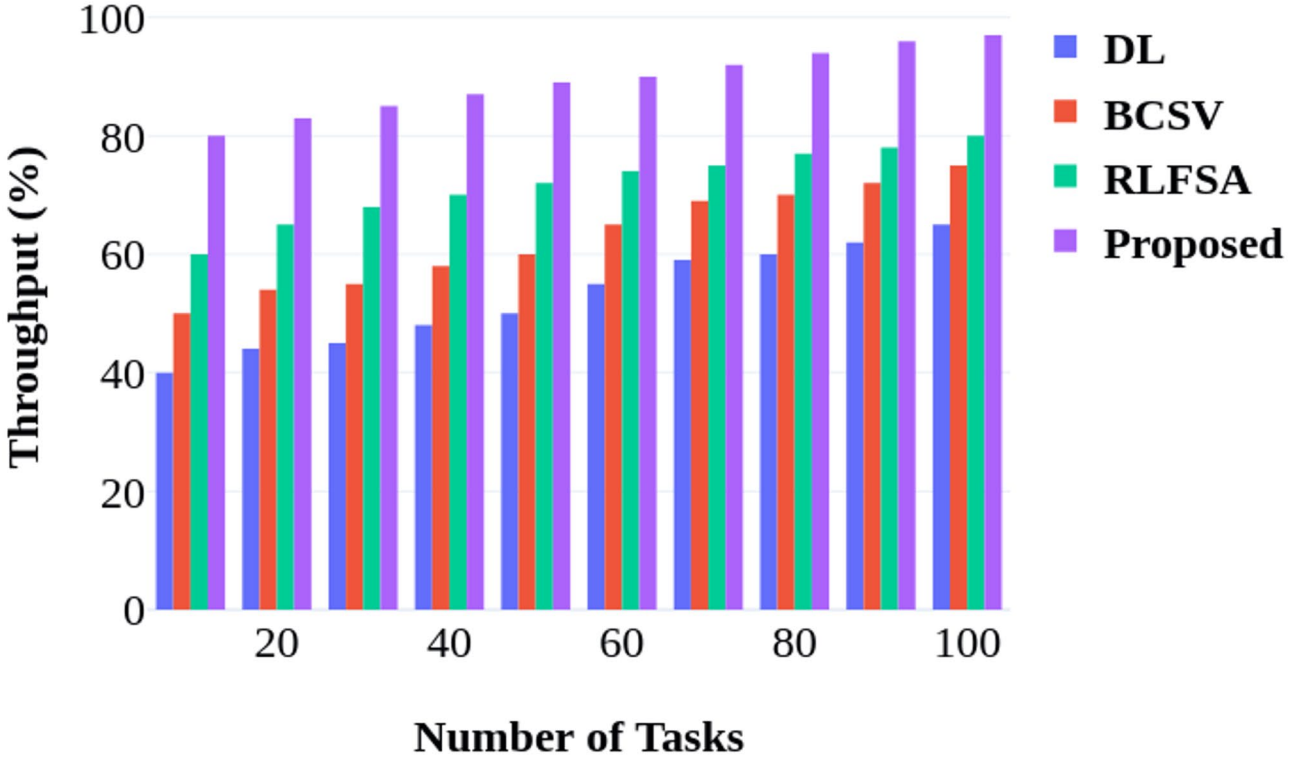

Fig. 7 Throughput

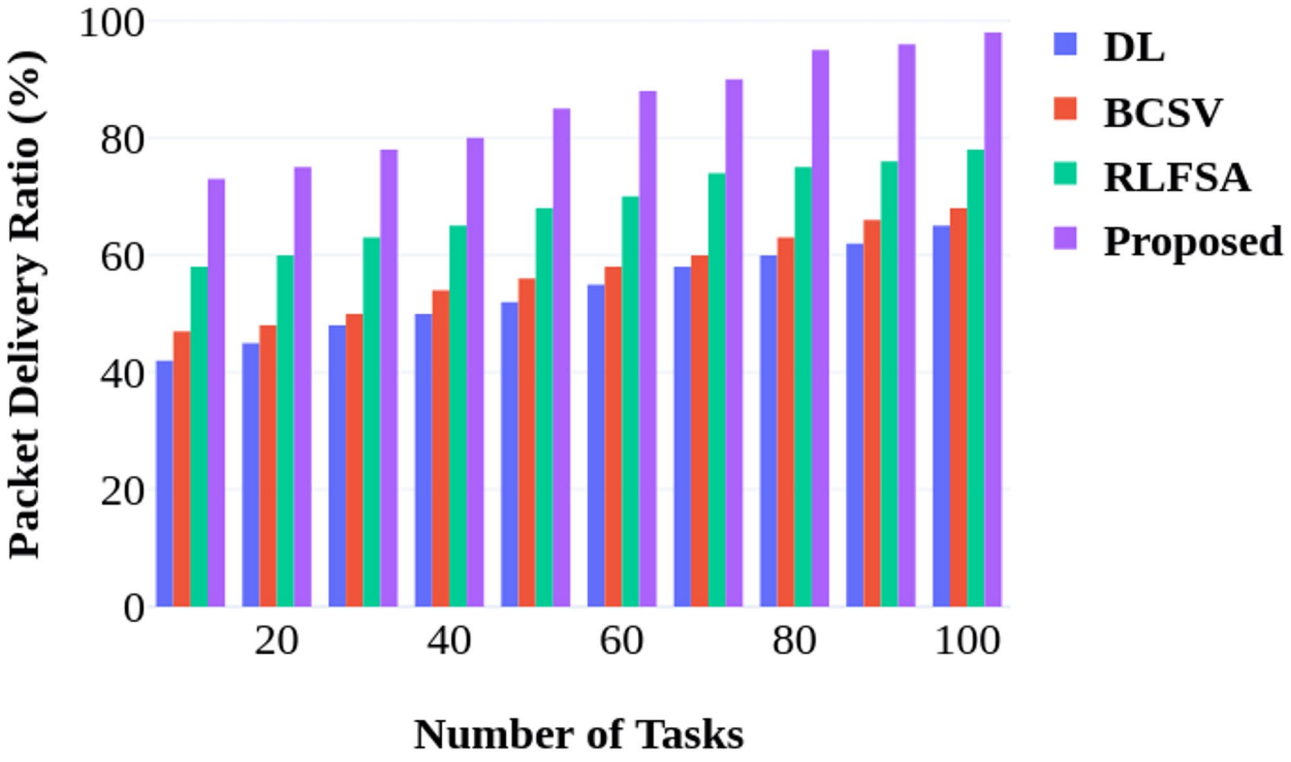

Fig. 8 PDR

development over the range. BCSV starts with a throughput of 50% for 10 tasks and rises to 75% for 100 tasks, viewing a steady development over the range. RLFSA starts at 60% for 10 tasks and influences 80% for 100 tasks, reliably performing improved than BCSV and DL diagonally in all task counts. The suggested techniques prove the highest throughput rates throughout, initially at 80% for 10 tasks and increasing to 97% for 100 tasks, demonstrating its superior efficacy in processing tasks compared to the other procedures.

e. *Packet Delivery Ratio*

Equation (44) computes the percentage of packets that are effectively delivered out of the entire number of packets sent. It is a key measure for measuring the reliability and effectiveness of data communication within the scheme.

$$P = \left(\frac{\alpha}{\beta}\right) \times 100 \tag{44}$$

α – number of packets delivered; β –Total number of packets sent

PDR stands for the ratio of packets successfully sent to the sender to the overall packets sent. It is instrumental in determining reliability in communication between distributed systems. The new technique has the largest PDR to ensure less loss of packets with intact data sent during communication. Figure 8 displays the Packet Delivery Ratio (PDR) (%) against the number of tasks. Primarily, DL starts with a PDR of 42% for 10 tasks and improves to 65% for 100 tasks, BCSV starts with a PDR of 47% for 10 tasks and improves to 68% for 100 tasks, indicating its performance in distributing packets over increasing task numbers. RLFSA starts at 58% for 10 tasks and reaches 78% for 100 tasks, consistently execution improved than BCSV, and DL across all task counts. The suggested techniques prove the uppermost PDR throughout, preliminary at 73% for 10 tasks and increasing to 98% for 100 tasks, showcasing its superior facility to deliver packets compared to the other procedures.

The choice of parameters in the hybrid metaheuristics, for every case, substantially affected performance metrics. In HAGA, increasing population size improved success rates in a task, though this raises the computational cost. With this aim to avoid complex time consumption and performance degradation, optimum settings were achieved. Even though the study underway has average performance metrics in terms of task success rate, average response time, memory utilization, throughput, and packet delivery ratio, the variance and expected value will enhance the ability to understand whether the result is consistent and reliable or not. Reporting these statistical measurements at multiple simulation runs will provide evidence of the resilience of the proposed system

under changing workloads and conditions, ensuring that the performance is not only optimum but also steady and predictable in various scenarios. The presented algorithms in the paper exhibit low-time complexities specialized for dynamic load balancing and scheduling. The time complexity of the Round-Robin Allocation with Sunflower Whale Optimization (RRA-SWO) is $O(n * m)$,where m represents the iterations and n the tasks, so it can distribute resources effectively. The Hybrid Ant Genetic Algorithm (HAGA) is $O(n^2 * m)$ complex because it blends Ant Colony Optimization and Genetic Algorithms, optimizing exploration and exploitation. The Least Response Time (LRT) algorithm works under $O(n)$ time complexity, appropriate for real-time monitoring. The Harmony Search Algorithm using Linear Regression (LR-HSA) works under $O(n * m)$ time complexity, efficiently predicting and adapting loads. The LRU method has the optimal time complexity of $O(1)$ when it comes to accessing memory, with efficient rescheduling. Compared to existing methods like Deep Reinforcement Learning with Parallel Particle Swarm Optimization (DRLPPSO) with a time complexity of $O(n^3)$ and Meta-RHDC with $O(n^2 * m)$, the new methods have an equilibrium between computational ability and accuracy. This makes them highly efficient for large-scale, real-time use cases in distributed and parallel systems. The performance of SFWOA and HAGA is very sensitive to parameter tuning. For SFWOA, some of the important parameters such as exploration-exploitation balance (A)and population size (N) affect convergence and computational burden. Too much exploration can make runtime longer, and too little exploration may cause premature convergence. For HAGA, the population size (P), mutation probability (Pm), and crossover probability (Pc) play important roles in determining solution diversity and convergence rate. Inefficient use of resources and poor scheduling are caused by inappropriate parameter settings, as was shown in a sensitivity analysis, highlighting the need for judicious parameter selection.

Research summary

To begin, the current configuration contains 51 virtual machines, 5 data centers, and 100 concurrent users, the method might be tested in more expansive and dynamic conditions to find out whether it is more robust and flexible. Finally, we establish a cloud-sim environment. RRA-SWO is the method that we use to arrive at a procedure for resource allocation. After that, we use the HAGA to schedule tasks in parallel according to their respective priorities. The LRT for Load Monitoring procedure is then put into action with our assistance. In the following step, we will use the LR-HSA to carry out Distributed Computing-based Load Prediction and Adjustment. This is followed by the implementation of the LRU approach for Dynamic Load Balancing techniques. In conclusion, Figs. 4, 5, 6, 7 and 8 contain graphical evaluations of the performance measures that were analyzed in the proposed work. Scalability would then be proved by the assessment of the system's performance with heavy workloads, like higher user demands or even more complex job dependencies. Further, testing various configurations of the system, such as changing the number of virtual machines, data centers, or cloud service providers, can illustrate how well task scheduling and resource usage work. In these scenarios, cost-effectiveness, energy usage, and latency should be included as metrics to confirm the viability of the approach. Future work will also compare the performance of the proposed algorithms under different scenarios by comparing them with more advanced approaches, such as deep reinforcement learning-based schedulers. The understanding of the framework's ability to maintain effectiveness and dependability in real distributed systems will be enhanced.

Conclusion and future direction

The investigation of dynamic scheduling approaches for load balancing in parallel and distributed schemes using the results of CloudSim to optimize resource usage and enhance system performance has been encouraging. Through the application of multiple protocols and techniques in a CloudSim simulation, we evaluated the effectiveness of these strategies in a range of scenarios and workloads. Resource distribution was suggestively enhanced by the addition of RRA-SWO, indicating a balanced load and the best utilization of available resources. HAGA's robustness in managing dynamic workloads is demonstrated by its ability to minimize task execution durations and optimize throughput when used for parallel task scheduling. Furthermore, while the LR-HSA load forecasting method precisely predicted future loads, it also allowed for proactive resource management. The LRT methodology for load monitoring offered real-time insights into the system's efficiency, allowing for appropriate adjustments. The LRU method for dynamic load balancing has shown usefulness in task reallocation and bottleneck evasion, foremost to improved response times and diminished latency. The results from the simulation verified important enhancements in key performance metrics.

Even if the proposed approach invariably shows better performance on most key metrics like task completion rate, average response time, memory usage rate, throughput, and packet delivery ratio, some constraints must be kept in mind. Its potential widespread application may be constrained because its metrics have not been measured. Such metrics include cost-effectiveness, scalability under heavy loads, and energy efficiency. In addition, the

adaptability of HAGA in different environments could be constrained by its dependency on parameters like HMS or pheromone levels. Last but not least, even if simulated scenarios provide much insight, more research is required to understand how real-world systems perform because they are susceptible to network delay, dynamic change, and unexpected breakdowns. These are features that require further development and validation of the methodology.

Acknowledgements
The authors extend their appreciation to the Deanship of Scientific Research at Northern Border University, Arar, KSA, for funding this research work through the project number "NBU-FFR-2025-1260-02".

Authors' contributions
Albalawi conceptualized the study, designed the research framework, and developed the methodology. He conducted an extensive literature review, performed data collection and analysis, and implemented. Additionally, he contributed to software development, experimental validation, and result interpretation. Albalawi also wrote the initial draft of the manuscript, revised and edited the content for clarity and technical accuracy, and handled final manuscript preparation and submission.

Funding
Northern Border University supported this research. The Article Processing Charge (APC) was covered by Northern Border University.

Data availability
No datasets were generated or analysed during the current study.

Declarations

Competing interests
The authors declare no competing interests.

Received: 14 February 2025 / Accepted: 21 May 2025
Published online: 01 July 2025

References

1. Shirvani MH, Rahmani AM, Sahafi A (2020) A survey study on virtual machine migration and server consolidation techniques in DVFS-enabled cloud datacenter: taxonomy and challenges. J King Saud University-Computer Inform Sci 32(3):267–286
2. Hung LH, Wu CH, Tsai CH, Huang HC (2021) Migration-based load balance of virtual machine servers in cloud computing by load prediction using genetic-based methods. IEEE Access 9:49760–49773
3. Ahmed H, Syed HJ, Sadiq A, Ibrahim AO, Alohaly M, Elsadig M (2023) Exploring performance degradation in virtual machines sharing a cloud server. Appl Sci 13(16):9224
4. Singh P, Kaur R, Rashid J, Juneja S, Dhiman G, Kim J, Ouaissa M (2022) A fog-cluster-based load-balancing technique. Sustainability 14(13):7961
5. Hu B, Cao Z, Zhou M (2021) Energy-minimized scheduling of real-time parallel workflows on heterogeneous distributed computing systems. IEEE Trans Serv Comput 15(5):2766–2779
6. Li J, Gu C, Xiang Y, Li F (2022) Edge-cloud computing systems for smart grid: state-of-the-art, architecture, and applications. J Mod Power Syst Clean Energy 10(4):805–817
7. Aminizadeh S, Heidari A, Toumaj S, Darbandi M, Navimipour NJ, Rezaei M, Unal M (2023) The applications of machine learning techniques in medical data processing based on distributed computing and the Internet of Things. Comput Methods Programs Biomed. 107745
8. Agbaje MO, Ohwo OB, Ayanwola TG, Olufunmilola O (2022) A Survey of game-theoretic approach for resource management in cloud computing. J Comput Netw Commun. 2022(1):9323818
9. Wang J, Wang L (2021) A computing resource allocation optimization strategy for massive internet of health things devices considering privacy protection in cloud edge computing environment. J Grid Comput 19(2):17
10. Laghari AA, Zhang X, Shaikh ZA, Khan A, Estrela VV, Izadi S (2023) A review on quality of experience (QoE) in cloud computing. J Reliable Intell Environ. 1–15
11. Hosseini Shirvani M (2024) A survey study on task scheduling schemes for workflow executions in cloud computing environment: classification and challenges. J Supercomputing 80(7):9384–9437
12. Tang X, Cao W, Tang H, Deng T, Mei J, Liu Y, Zeng Z (2021) Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds. IEEE Transact Parallel Distributed Syst 33(9):2079–2092.
13. Asghari A, Sohrabi MK, Yaghmaee F (2021) Task scheduling, resource provisioning, and load balancing on scientific workflows using parallel SARSA reinforcement learning agents and genetic algorithm. J Supercomputing 77(3):2800–2828
14. Al-Masri E, Souri A, Mohamed H, Yang W, Olmsted J, Kotevska O (2023) Energy-efficient cooperative resource allocation and task scheduling for internet of things environments. Internet Things 23:100832
15. Ng JS, Lim WYB, Luong NC, Xiong Z, Asheralieva A, Niyato D, Miao C (2021) A comprehensive survey on coded distributed computing: Fundamentals, challenges, and networking applications. IEEE Commun Surveys Tutorials 23(3):1800–1837
16. Xiao H, Yi K, Peng R, Kou G (2021) Reliability of a distributed computing system with performance sharing. IEEE Trans Reliab 71(4):1555–1566
17. Kashani MH, Mahdipour E (2022) Load balancing algorithms in fog computing. IEEE Trans Serv Comput 16(2):1505–1521
18. Wu Z, Sun J, Zhang Y, Wei Z, Chanussot J (2021) Recent developments in parallel and distributed computing for remotely sensed big data processing. Proc IEEE 109(8):1282–1305
19. Kumar S, Mohbey KK (2022) A review of big data based parallel and distributed approaches of pattern mining. J King Saud University-Computer Inform Sci 34(5):1639–1662
20. Galante G, da Rosa Righi R, de Andrade C (2024) Extending parallel programming patterns with adaptability features. Cluster Comput 1–22
21. Pradhan A, Bisoy SK, Kautish S, Jasser MB, Mohamed AW (2022) Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment. IEEE Access 10:76939–76952
22. Mishra K, Pati J, Majhi SK (2022) A dynamic load scheduling in IaaS cloud using binary JAYA algorithm. J King Saud University-Computer Inform Sci 34(8):4914–4930
23. Al Reshan, MS Syed, D Islam, N Shaikh, A Hamdi, M Elmagzoub, MA Talpur, KH (2023) A fast converging and globally optimized approach for load balancing in cloud computing. IEEE Access, 11, 11390–11404.
24. Jena UK, Das PK, Kabat MR (2022) Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment. J King Saud University-Computer Inform Sci 34(6):2332–2342
25. Kaur M, Kadam S, Hannoon N (2022) Multi-level parallel scheduling of dependent-tasks using graph-partitioning and hybrid approaches over edge-cloud. Soft Comput 26(11):5347–5362
26. Ramezani Shahidani F, Ghasemi A, Toroghi Haghighat A, Keshavarzi A (2023) Task scheduling in edge-fog-cloud architecture: a multi-objective load balancing approach using reinforcement learning algorithm. Computing 105(6):1337–1359
27. Souravlas S, Anastasiadou SD, Tantalaki N, Katsavounis S (2022) A fair, dynamic load balanced task distribution strategy for heterogeneous cloud platforms based on Markov process modeling. IEEE Access 10:26149–26162
28. Khan AR (2024) Dynamic load balancing in cloud computing: optimized RL-Based clustering with Multi-Objective optimized task scheduling. Processes 12(3):519
29. Jangra A, Mangla N (2023) An efficient load balancing framework for deploying resource schedulingin cloud based communication in healthcare. Measurement: Sens 25:100584
30. Muthusamy A, Dhanaraj RK (2023) Dynamic Q-Learning-Based optimized load balancing technique in cloud. Mob Inform Syst 2023(1):7250267
31. Ghafir S, Alam MA, Siddiqui F, Naaz S (2024) Load balancing in cloud computing via intelligent PSO-based feedback controller. Sustainable Computing: Inf Syst 41:100948
32. Priyadarshini A, Pradhan SK, Laha SR, Nayak S, Pattanaik BC (2024) Dynamic load balancing with task migration: a genetic algorithm approach for optimizing cloud computing infrastructure. In 2024

International Conference on Advancements in Smart, Secure and Intelligent Computing (ASSIC). pp. 1–6. IEEE
33. Banerjee P, Roy S, Sinha A, Hassan MM, Burje S, Agrawal A, El-Shafai W (2023) MTD-DHJS: makespan-optimized task scheduling algorithm for cloud computing with dynamic computational time prediction. IEEE Access
34. Devi N, Dalal S, Solanki K, Dalal S, Lilhore UK, Simaiya S, Nuristani N (2024) A systematic literature review for load balancing and task scheduling techniques in cloud computing. Artif Intell Rev 57(10):276
35. Zhanuzak R, Ala'Anzy MA, Othman M, Algarni A (2024) Optimising cloud computing performance with an enhanced dynamic load balancing algorithm for superior task allocation. IEEE Access
36. Krishna MSR, Vali KD (2025) Meta-RHDC: Meta reinforcement learning driven hybrid lyrebird Falcon optimization for dynamic load balancing in cloud computing. IEEE Access (99):1–1
37. Li H, Li J, Duan X, Xia J (2025) Energy-aware scheduling and two-tier coordinated load balancing for streaming applications in Apache Flink. Future Generation Comput Syst 166:107681
38. Li H, Xia J, Luo W, Fang H (2022) Cost-efficient scheduling of streaming applications in Apache Flink on cloud. IEEE Trans Big Data 9(4):1086–1101
39. Li H, Fang H, Dai H, Zhou T, Shi W, Wang J, Xu C (2022) A cost-efficient scheduling algorithm for streaming processing applications on cloud. Cluster Comput 1–23
40. Ziyath SPM, Subramaniyan S (2022) An improved Q-learning-based scheduling strategy with load balancing for infrastructure-based cloud services. Arab J Sci Eng 47(8):9547–9555
41. Chiang ML, Hsieh HC, Cheng YH, Lin WL, Zeng BH (2023) Improvement of tasks scheduling algorithm based on load balancing candidate method under cloud computing environment. Expert Syst Appl 212:118714
42. Nabi S, Ibrahim M, Jimenez JM (2021) DRALBA: dynamic and resource aware load balanced scheduling approach for cloud computing. IEEE Access 9:61283–61297
43. Chung WK, Li Y, Ke CH, Hsieh SY, Zomaya AY, Buyya R (2021) Dynamic parallel flow algorithms with centralized scheduling for load balancing in cloud data center networks. IEEE Trans Cloud Comput 11(1):1050–1064
44. Talaat FM, Ali HA, Saraya MS, Saleh AI (2022) Effective scheduling algorithm for load balancing in fog environment using CNN and MPSO. Knowl Inf Syst 64(3):773–797

Publisher's Note