International Journal of Mathematical, Engineering and Management Sciences
Vol. 10, No. 5, 1585-1603, 2025
https://doi.org/10.33889/IJMEMS.2025.10.5.075

Ram Arti Publishers

Cloud Heterogeneous Networks: Cooperative Random Spatial Local Best Particle Swarm Optimization for Load Balancing

Tanu Kaistha
Department of Electronics and Communication Engineering,
I. K. Gujral Punjab Technical University, Kapurthala, Punjab, India.
Corresponding author: tanu.etrx@gmail.com

Kiran Ahuja
Department of Electronics and Communication Engineering,
DAV Institute of Engineering and Technology, Jalandhar, Punjab, India.
E-mail: askahuja2002@gmail.com

(Received on December 9, 2024; Revised on February 8, 2025 & April 19, 2025; Accepted on May 19, 2025)

Abstract
Cloud services are growing in popularity and undergoing substantial change. To maximize performance, it is necessary to distribute the workload efficiently across multiple virtual machines (VMs). Therefore, a new cooperative LB method called Random Spatial Local Best Particle Swarm Optimization (RSLbestPSO) in cloud computing heterogeneous networks is developed to balance the workload on all VMs efficiently. Unlike traditional approaches, RSLbestPSO aims to increase performance by decreasing response time, finding the most efficient VMs, and improving the response time. The RSLbestPSO works by initializing the particles of which the fitness function will be computed, and the solution with the highest fitness is considered the best solution. The experiments showed that the proposed work effectively balanced the load on the VMs by finding the optimal solution, reducing the makespan time, and increasing the response time. The evaluated results show the effectiveness of the proposed RSLbestPSO.

Keywords- Load balancing, Cloud, Heterogenous networks, Random spatial local best particle swarm optimization.

1. Introduction

In the cloud, heterogeneous networks exist in which multiple virtual machines (VMs) work in a distributed manner to handle the massive amount of tasks from all over the world. Clients request many tasks or services. The task of cloud computing is to process all the requests made by the client. The functions requested by the client will include managing the storage, deploying the web applications, and creating the servers (Rashid & Chaturvedi, 2019). These resources can be accessible to clients freely at any time. As billions of people are requesting something, providing these resources is a great challenge. During the allocation of tasks, sometimes more work is allocated to some VMs and less to others, which leads to over-utilized and under-utilized VMs (Tripathy et al., 2023). This results in a decrease in response time, throughput time, and an increase in the average load. Therefore, efficient load balancing (LB) techniques are required for balancing the load.

The distribution of work evenly on the VMs so that resources are provided efficiently is known as load balancing. Several state-of-the-art methods work on balancing the load, but the performance is inadequate. It leads to higher response time, higher latency, and higher throughput. To overcome these, efficient LB techniques are required for decreased response time and throughput. There are two types of LB techniques: static LB and dynamic LB. The basic algorithms like the first come, first served (FCFS) (Saeed et al., 2019), Round Robin (RR) (Prassanna & Venkataraman, 2019), and shortest job first (SJF) (Waheed et al., 2019) comes under the category of static LB. Dynamic LB has two types of algorithms: cooperative and non-cooperative LB. Cooperative, from the name it is clear, that is cooperation from each in completing the

same task. All the processes work together to complete the same task. Non-cooperative on the other side, make their own decisions. They do not listen to others and complete their tasks alone (Houssein et al., 2021). The metaheuristic algorithms like particle swarm optimization (PSO) (Ahuja et al., 2018), ant colony optimization (ACO) (Dam et al., 2014), simulated annealing (SA) (Mondal & Choudhury, 2015), genetic algorithms (GA) (Dasgupta et al., 2013), Hill climbing, local search (Zahid et al., 2019) and Tabu search (Téllez et al., 2018) and non-cooperative algorithms include the game theory techniques. The cloud computing load balancing work started in 1995 (Young, 1995), which was an excellent opportunity for researchers to work on it. In 2020, research was conducted by Mishra et al. (2020) to balance the load in cloud computing. The study was performed on the CloudSim simulator (Sundas & Panda, 2020). This study focuses on the work done by various researchers, which also helps identify future directions in the cloud network (Gamal et al., 2019). The limitations were also mentioned, which include the decreased response, turnaround, and latency. The problem of local optima was also discussed. To solve all these challenges, efficient load-balancing techniques are required to reduce the response time, throughput, and average load (Upadhyay et al., 2018).

This research proposes a cooperative load-balancing algorithm called random spatial local best particle swarm optimization to efficiently balance the workload on the VMs. The task will be migrated from the under-utilized VMs or over-utilized VMs to balanced VMs. In the PSO, the particles get stuck in the local optima and fail to find the global solution. So, the main aim of the proposed research is to find the best global solution. The term cooperative means the cooperation between the particles to find the best solution. The particles cooperate during the velocity equation, where they share information. The experiment is performed in a CloudSim environment. The performance of the proposed work is evaluated using CPU utilization, memory utilization, makespan time, turn around time (TAT), response time (RT), and average load. The main aim of the proposed RLSbestPSO is to reduce the makespan time, TAT time, and RT while maintaining the workload on the VMs. The presented research plays a vital role as it improves the execution time, reduces resource utilization, and decreases the completion time of the tasks and execution costs.

1.1 Contribution

(a) A random spatial local best PSO (RSLbestPSO) has been developed to balance the workload on VMs efficiently.
(b) The system's response time will be decreased by balancing the load on the VMs using the RSLbestPSO.
(c) The performance of the proposed work is evaluated using various parameters, including makespan time, execution time, average TAT, average RT, and average load.

1.2 Motivation

In heterogeneous cloud networks, load balancing among the VMs is a tedious task as the traditional algorithms like PSO, Genetic Algorithms (GA), or Ant Colony Optimization (ACO) failed to explore all possible solutions, which leads to increased response time, makespan time, and imbalanced VMs. This motivates us to design an algorithm that finds solutions globally instead of sticking to local optima, decreasing response time, makespan time, and balancing virtual machines.

The rest of the paper is structured as follows: The literature review is given in Section 2. Section 3 describes the proposed work. The experimental setup and results are shown in Section 4, and Section 5 outlines the conclusion and future work.

2. Literature Review

Multiple researchers have worked on cooperative LB techniques to balance workloads by scheduling the task to the optimal VM in cloud computing heterogeneous networks. A novel approach utilizing logarithm-

based PSO(L-PSO) was proposed by Huang et al. (2020) to effectively distribute the workload across VMs in heterogeneous networks within cloud computing. The logarithmic reduction was used to perform LB operations. Some comparison techniques, such as Artificial Bee Colony (ABC), Gravitational Search Algorithm (GSA), and Dragonfly Algorithm (DA), were used. The result showed that L-PSO had effective performance, and it helped reduce the makespan time by 21.42 %,19.12%, and 15.14%, respectively, compared to the abovementioned technique. Jena et al. (2022) proposed a hybridized approach called QMPSO using the pioneering approach for the dynamic workload distribution across virtual machines in a cloud computing network by combining a modified MPSO learning algorithm. The aim of using Hybridization is to improve machine efficiency by distributing the workload evenly among VMs. When the comparative analysis was done, it was found that QMPSO algorithms perform better than its competitors. Babu & Philip (2016) presented an EBCS-LB algorithm to enhance the Bee colony algorithm for LB.

The foraging behavior of honey bees is used to divide the workload among machines equally. An experimental result showed that EBCA-LB increases the quality of service (QoS) provided to customers. Inspired by the concept of honey bees, Babu & Krishna (2013) introduced an LB framework. The algorithm helped effectively manage the task priorities on VMs, aiming to reduce the queue waiting time. Kumar & Sharma (2020) introduced a dynamic LB algorithm (DLBA) to divide the workload on VMs in a cloud computing network. Two operations were performed: first, the load on each VM is checked, and then the task on overloaded VMs is moved to the next underloaded VM. The reduction in makespan time and improved resource utilization were seen, and DLBA did well.

Pradhan & Bisoy (2022) additionally introduce an improved PSO technique to balance a cloud computing network's load (LBMPSO). For simulation, CloudSim was used, and the result showed that LBMPSO was better in reducing the makespan time.

FL-GWO is a fuzzy logic method introduced by Xingjun et al. (2020) using a grey wolf optimizer (GWO) for LB in a cloud-based IoT environment. The experiment showed that FL-GWO performed way better than the existing state-of-the-art method. To balance the load on various VMs, Devaraj et al. (2020) introduced a hybrid of Firefly (FF) and an improved multi-objective PSO (IMPSO) algorithm called FFIMPSO. The authors used IMPSO to discover the enhanced response required to schedule the task on multiple VMs and the Firefly to compute the search space. The development was compared with individual FF, improved PSO(IPSO), and combined FF-IPSO. Among comparison with various algorithms like round robin (RR), shortest job first (SJF), first come, first served (FCFS), weighted RR (WRR), diffusive LB (DLB), and LB Bayes and clustering (LB-BC), it was observed that FFIMPSO did well among them and found more efficient in balancing the load on the VMs.

A hybrid of lateral Wolf (LW) and PSO optimization for LB and shifting tasks from overloaded VMs to underloaded VMs was introduced by Malik & Suman (2022). It aimed to make the LW algorithm execute task scheduling simultaneously. The PSO has been used to achieve optimal solutions by leveraging LW, thereby identifying the most optimized virtual machines (VMs). Similarly, Simaiya et al. (2024) presented a hybrid framework combining Convolutional Neural Network (CNN) with Long Short-Term Memory (LSTM) to balance the workload on the VMs. PSO with GA was also used for training. The experiment that was conducted shows the effectiveness of the proposed work. Ajil & Kumar (2025) presented an improved deep belief network (IDBN) for predicting and balancing the workload on the VMs. First, the load was predicted, which was further passed to butterfly optimization, which reduce the response time, throughput, and makespan time efficiently. **Table 1** shows the work done by various researchers on LB using cooperative LB. These algorithms are widely recognized for their robust exploitation capabilities;

once identified, they can efficiently and effectively converge toward a favorable solution. While dealing with high dimensional search space and problems related to task scheduling and LB, this algorithm can pose challenges because of numerous locally optimal solutions. An algorithm aims to search a broader range of the solution space effectively. A random spatial local best PSO (RSLbestPSO)is introduced to remove the local optima problem and to balance the load on VMs in heterogeneous cloud computing networks.

Table 1. Related work on cooperative LB on VMs in cloud computing heterogeneous networks.

Author [Reference]	Algorithm	Work done	Limitations
Huang et al. (2020)	L-PSO	A logarithm-decreasing strategy is proposed to find the optimal solution to assign tasks to heterogeneous VMs.	The results achieved are not optimal. Hence, optimized algorithms are required to improve the performance.
Jena et al. (2022)	QMPSO	QMPSO modifies the velocity of the MPSO by incorporating the best action derived from the enhanced Q-learning algorithm into the gbest and pbest.	The proposed algorithms are static, which limits their performance. Hence, dynamic load balancing is required.
Babu & Philip (2016)	EBCA-LB	The forging behavior of honey bees was used to distribute the workload of VMs in a cloud environment.	The presented technique used a simple swarm algorithm. Hybridization with other algorithms comprising PSO and GA is required for enhanced performance.
Babu & Krishna (2013)	HBB-LB	The highest waiting time is given priority, and the load is reduced from the overloaded VMs.	QoS parameters were not considered while measuring the performance, a significant limitation of the presented work.
Kumar & Sharma (2020)	DLBA	Task migration is done dynamically for LB on overloaded VMs.	The task deadline was not considered while measuring the performance that needs to be done for better results.
Pradhan & Bisoy (2022)	LBMPSO	PSO was improved to find the optimal VM for task allocation and reduce the workload.	The performance achieved is not optimal. QoS parameters will be considered in the future.
Xingjun et al. (2020)	FL-GWO	Balanced the workload on VMs by computing the distance; the least distance was selected.	GWO suffers from the problem of local optima, which can be improved in future studies.
Devaraj et al. (2020)	FFIMPSO	FF found search space, and IMPSO obtained the optimal solution.	Data deduplication was not considered in the proposed work and needs to be done for better results.
Malik & Suman (2022)	LW-PSO	LW calculated fitness, which was passed to PSO to find the optimal solution.	The present study suffers from the local optima problem and cannot find the best solution.
Simaiya et al. (2024)	CNN-LSTM with GA-PSO	CNN+LSTM was used to compute the fitness, which was passed to GA-PSO for training.	Security and privacy are yet to be addressed, and this is a significant concern that needs to be discussed in the future.
Ajil & Kumar (2025)	IDBN	IDBN efficiently balances the workload, reducing the makespan, response, and throughput.	Better prediction models can be used in the future to balance the workload.

3. Proposed Work

In this research, a Random Spatial Local Best PSO (RSLbestPSO) is proposed to discover the most optimized VMs and to balance the load on these VMs by migrating the task from an overloading VM to an under-loading VM. First, the task sequence is created based on user requests and passed to the VM manager. It is followed by making the VMs into heterogeneous networks, and the tasks are randomly assigned to active VMs. Afterward, RSLbestPSO is called, which updates the velocity and position to find the global optimum solution. Then, based on the optimal solution, optimal VMs are selected, and the tasks can be moved from the overloaded VMs to optimal VMs, thus maintaining the load balance on cloud computing networks. The optimized performance of the RSLbestPSO is computed using various parameters comprising makespan time, execution time, average TAT, average RT, average load, CPU utilization, and Memory Utilization. The complete working of RSLbestPSO is shown in **Figure 1**.

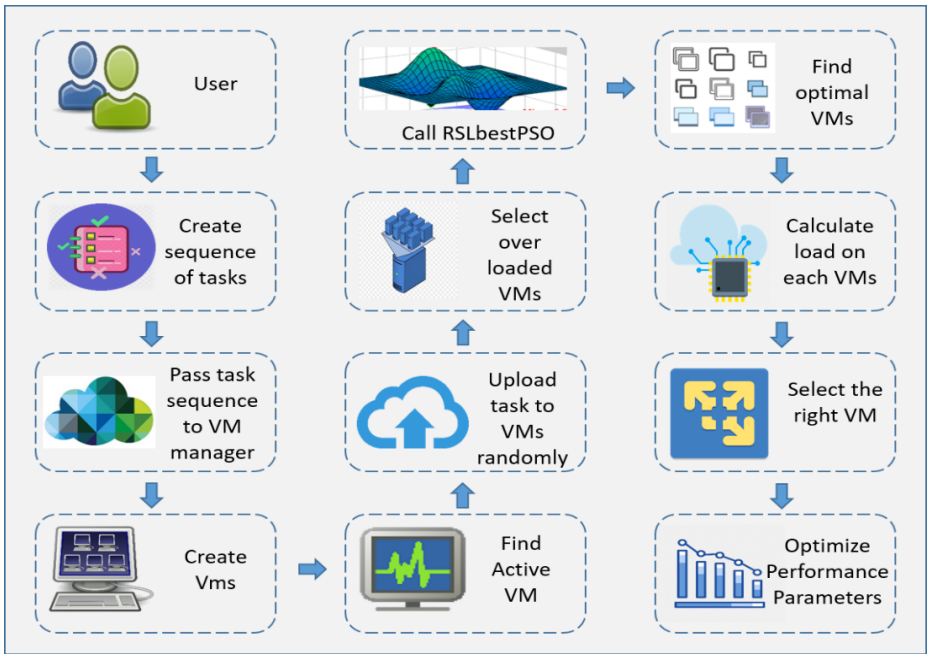


Figure 1. Workflow diagram explaining the working of proposed RSLbestPSO.

3.1 Particle Swarm Optimization (PSO)

PSO is a computational optimization method that draws inspiration from the collective behavior observed in bird flocking or fish schooling (Marini & Walczak, 2015). The algorithm in question employs a metaheuristic approach, operating on a population level, to address optimization problems. PSO involves utilizing a group of potential solutions, referred to as particles, that systematically navigate the search space to identify the most optimal solution through iterative processes. Every individual particle within the system is indicative of a prospective solution. Moving in a search space by modifying the position and responsible for its experience and those adjacent particles. Every particle in the swarm is represented by a point in the search space, which is also known as a vector of real number position. To update the velocity and positions if the particle PSO algorithm utilizes an iterative procedure and gradually moves toward an optimal solution. Using cognitive and social components, PSO effectively searches for the optimal solution by integrating exploration and exploitation strategies. PSO depends on the exchange of data between particles. At the same time, the swarm becomes confined to a local optimum. It may require help in effectively examining the different areas of the search space, and finding the better solution outside of the local optima may be hampered by the limited scope of this investigation (Dhillon et al., 2023). GA, ACO, and reinforcement learning can also be applied, but to find the best solution, several generations of crossovers and mutations are required in GA, which will lead to high computational complexity. There is also a problem of stagnation in the GA, which will lead to a loss of diversity in the candidate solutions.

On the other hand, ACO is well suited for discrete search spaces like routing paths, and CPU utilization requires continuous values. Similarly, in reinforcement learning, several training slots are needed in which the agent has to interact with the environment to compute the reward, which takes a lot of time for execution. Therefore, to surpass the PSO, GA, and reinforcement learning limitation, a local best solution, a random spatial, is added to the current PSO described in the sections below.

3.2 Random Spatial Local Best PSO (RSLbestPSO)

The improvement of PSO is RSLbestPSO. RSLbestPSO has improved the problem of PSO, which is the local optima. Due to local optima, the best solution is ignored as it gets stuck in the small space and thus ignores the benefits of the larger space. The complete mathematical model of the proposed algorithm is given as follows:

Consider the set of tasks like Tk_1, $Tk_{2, \ldots\ldots}$ Tk_n, VMs like VM_1, $VM_{2\ldots\ldots}$ $VM_{n,}$ and particles like $P_{1,}$ $P_{2\ldots\ldots}$ $P_{n.}$ Each of the particle fitness functions is calculated. The objective is to minimize the makespan time.

To enable a particle to explore the search space more comprehensively, we use the RSLbestPSO algorithm, which includes a random spatial element in the velocity update equation. Including random spatial elements helps increase exploration, allowing the particle to traverse random trajectories and potentially evade local optima. By adding a random element, the RSLbestPSO algorithm can explore unknown trajectories, helping a more extensive exploration and enhancing the chance of identifying the global optimum. Particles are used to exchange their information with their neighboring particle in the RSLbestPSO algorithm by considering the local best position. Particles can use the knowledge of their neighboring particle to exchange information about their best position in their local area, helping them make well-informed decisions regarding their movement. Sharing local information within a swarm helps spread valuable information, enabling particles to converge toward improved solutions collectively. The velocity in RSLbestPSO is given by Equation (1):

$$Vel_{new} = I_w * Vel(t) + c_1 * r_1 * \left(Per_{best} - P(t)\right) + c_2 * r_2 * \left(Glbl_{best} - P(t)\right) + c_3 * r_3 * \left(R - P(t)\right) \quad (1)$$

where, $c_3 * r_3 * (R - P(t))$ denotes the random spatial component c_3, the local best accelerator coefficient and r_3 is the random number. The random position in the search space is denoted by R. The inclusion of a random spatial component introduces a random displacement vector to the particle's velocity, thereby promoting exploration by enabling particles to traverse arbitrary directions in each iteration. Similarly, using the velocity update Equation (1), the position of the particle is updated by the given Equation (2):

$$P_{new} = P(t) + Vel_{new} \quad (2)$$

Harmonious equilibrium between exploration and exploitation is attained by promoting the RSLbestPSO algorithm, and it is done by utilizing both random spatial exploration and local best information. Exploration in a random spatial component in the algorithm is done by introducing randomness; it promotes the particle to find different regions of the search space. One more side of the local best component of the algorithm is it promotes exploitation by using the information obtained from neighboring particles to allow the exploitation of hopeful regions within the search space. Embrace an optimized approach in RSLbestPSO, which enables the algorithm to visit various areas of the search space effectively.

We use this approach to enhance the algorithm's overall performance and help explore diverse areas while exploiting promising regions and reducing the risk of premature convergence to local optima. Algorithm 1 and **Figure 2** best describe the complete working of RSLbestPSO.We start the algorithm by initializing a particle population with randomly generated position and velocity. The current position and Glblbest do the assignment of the Perbest position of each particle by selecting the position with the highest fitness value from all the particles. The algorithm iteration loop continues until it reaches the termination condition, either reaching a predetermined maximum number of iterations or attaining a desired fitness level. Each particle is processed by the algorithm independently during each iteration. The objective function is being used to evaluate the current position of each particle's fitness. If the current position's fitness surpasses the particle's personal best fitness, the Perbest fitness and position are subsequently updated. If the present position's fitness exceeds the global best fitness, the $Glbl_{best}$ fitness and position are later revised. The

position and velocity of each particle are given by Equations (1) and (2), respectively. For each particle, a random subset of neighbors is chosen. To identify the optimal position, the comparative analysis involves calculating the fitness values of the current particle's personal best position and the positions of its neighboring particles. The Lbest position of the current particle is updated by incorporating the best position discovered in the preceding step. For the transition to the subsequent iteration, the iterator is increased by one. Once it meets the termination condition, the algorithm assigns the particle position with the highest fitness value ($Glbl_{best}$) as the optimized solution.

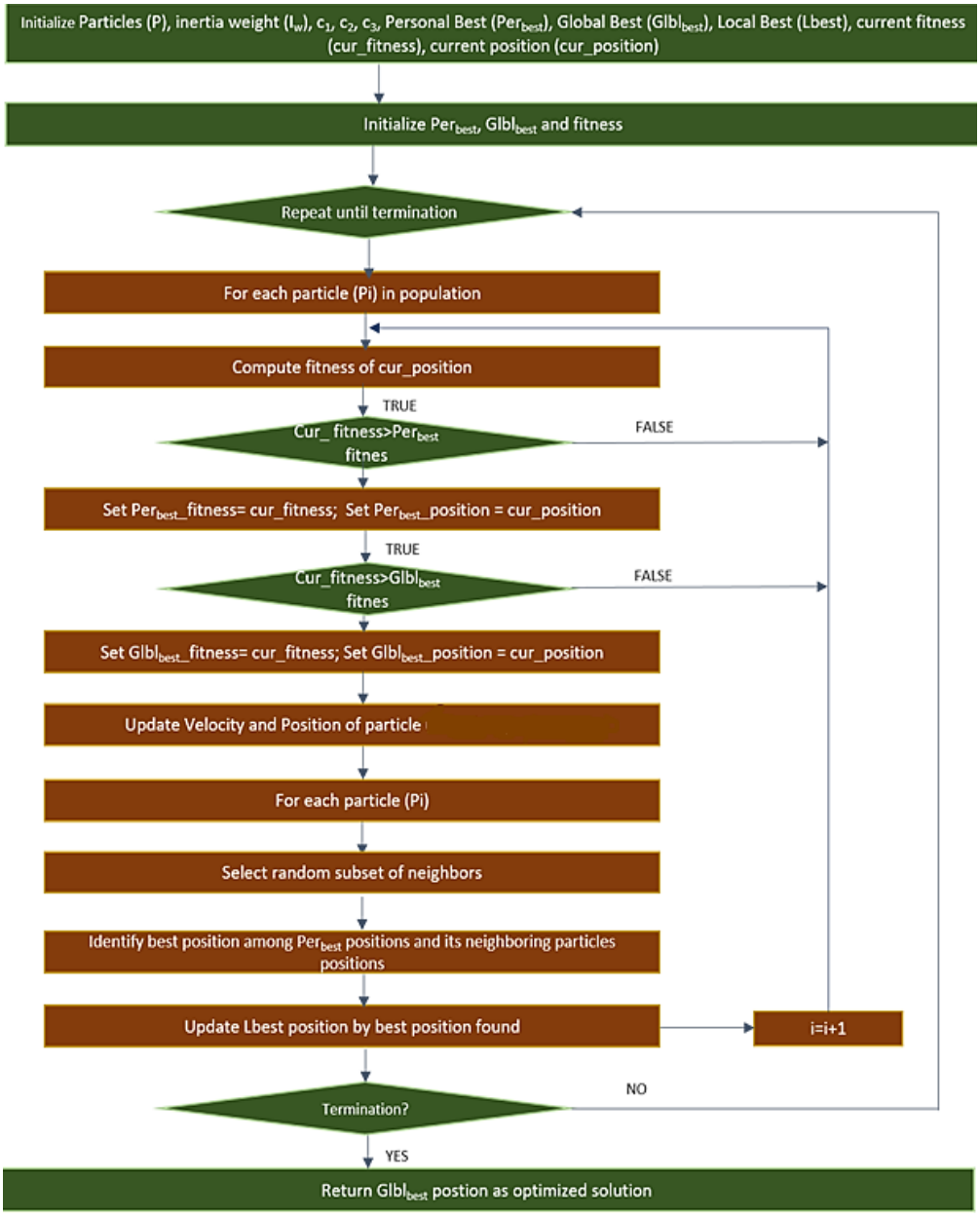


Figure 2. Flowchart of RSLbestPSO.

In the current study, the population size used is 100 particles with c_1, c_2, and c_3 values of 10, 10, and 20, respectively. The parameters c_1, c_2, and c_3 are selected with the random optimization. The random values have been chosen from the pool of the parameters, and performance is evaluated. The results showed that the specified value using random optimization performed best, reducing the makespan time of the VMs.

The RSLbestPSO added a stochastic displacement factor in Equation (1), improving the particles' search capabilities in finding the solution. The displacement factor helps the particles go around in all the random directions, avoiding early convergence with the local optima problem. The normal PSO worked only on the personal best and global best problems. Thereby, particles get stuck in the local optima. This problem is removed by adding $c_3 * r_3 * \left(R - P(t)\right)$ to increase the exploration process.

Algorithm 1: Pseudocode explaining the step-by-step working of RSLbestPSO

Input: Particles (P), inertia weight (I_w), c_1, c_2, Personal Best (Per_{best}), Global Best ($Glbl_{best}$), current fitness (cur_fitness), current position (cur_position), local best acceleration coefficient (c_3), Local Best (Lbest)
Output: Best optimized solution
Begin:
Initialize population
Initialize the Per_{best} positions of particles with their current positions
Initialize the $Glbl_{best}$ position and fitness
Repeat until the termination condition is met:
For each particle (P_i):
  Compute the fitness of the current position
  If (cur_fitness>Per_{best}_fitness):
    Set Per_{best}_fitness = cur_fitness
    Set Per_{best}_position = cur_position
  If (cur_fitness>$Glbl_{best}$_fitness)
    Set $Glbl_{best}$_fitness = cur_fitness
    Set $Glbl_{best}$_position = cur_position
  Compute the new velocity of the particle with Equation (1)
  Compute the new position of the particle with Equation (2)
  For each particle (P_i):
    Select a random subset of neighbors
    Identify the best position among Per_{best} positions and its neighboring particle positions
    Update the Lbest position by the best position found
i=i+1
Return the Gbest as the optimized solution
End

In the current research, the task sequence is first created and passed to the VM manager. The next step involves the creation of virtual machines. Once the machines are created, tasks are assigned to VMs randomly. Then, the check for overloaded VMs is done, which is then passed to the RSLbestPSO algorithm. The algorithm selects the most optimal VMs and assigns tasks based on the best global solution it found. Then, the loads on the VM are computed, and the VM with the fewest loads is selected. Assigning tasks from overloaded VMs to the least-loaded or under-loaded VMs balances the load. Finally, the optimized performance is computed using makespan time, execution time, average TAT, average RT, average load, CPU utilization, and memory utilization.

4. Experimental Setup and Results

The minimum hardware requirement to perform the work is 8 GB of RAM and an i5 processor. In the current research, a population size of 100 is used. This value is selected because the more significant the search space is, the better the exploration is. When the search space is ample, diversity is improved, and the early convergence to the solutions is also prevented. The coefficients c_1, and $c_2$are 10 and 20 respectively. c_1 and c_2 are selected based on the literature. A higher value of c_2 will lead to better results for the global best, which will help the swarms find solutions effectively. The software required is Netbeans and CloudSim, where Netbeans is used to implement the RSLbestPSO, and CloudSim-3.0.3 is used to create the virtual nodes and tasks. The datasets used for experimenting are synthetic and generated by the CloudSim tool. A comparison is also made with the works that performed experiments on the synthetic data.

4.1 Implementation Details

The experiment is performed in several ways, completing each possible outcome. In the first case, RSLbestPSO computes the makespan time for a network of six different types of VMs comprising VM5, VM6, VM7, VM8, VM9, and VM10, and other numbers of tasks comprising 100, 150, 200, 250, 300, and 400 are allocated to each type of VM. **Table 2** below provides information on the VM and task properties for the first case.

Table 2. Different types of tasks and their sizes.

Task/ VM	Properties
Task range	100-400
Task size	30000-50000
VM range	5-10
Processing speed (MIPS)	100000
Bandwidth	1000000
Type of VMM	XEN

The second scenario computes the average TAT, average RT, and average load, and compares them with the existing work. In the last scenario, a different number of tasks, varying in size and comprising small, medium, large, and extra-large, are created for which CPU utilization and memory utilization are created. **Table 3** below provides the total number of small, medium, large, and extra-large tasks.

Table 3. Different types of tasks and their sizes.

Task type	Number of tasks	Tasks size
Small	100-200	30000-50000
Medium	400-500	50000-70000
Large	600-700	70000-100000
Extra-large	800-100	100000-200000

4.2 Results

In the first scenario, different numbers of VMs are created, and different numbers of tasks are assigned to them, i.e., for 5 VMs, 6 VMs, 7 VMs, 8 VMs, 9 VMS, and 10 VMs, 100, 150, 200, 250, 300, and 400 tasks are allocated, respectively. RSLbestPSO calculates the makespan time for each set of VMs and tasks, observing a decrease in the makespan time as the number of VMs and tasks increases. The total makespan time for 100, 150, 200, 250, 400, and 400 tasks assigned to 5 VMs, 6 VMs, 7 VMs, 8 VMs, 9 VMS, and 10 VMs is 189.16s, 143.1s, 111.42s, 87.53s, 83.38s, and 66.32s, respectively. **Table 4** shows the makespan time results for different VMs and tasks.

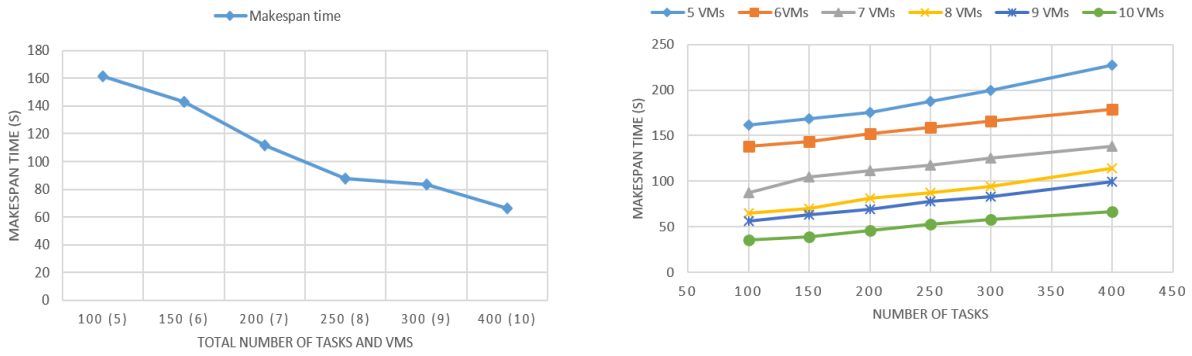
Table 4. Makespan time for different no. of VMs and tasks.

VMs	Tasks	Makespan time (s)
5	100	161.65
6	150	143.1
7	200	111.42
8	250	87.53
9	300	83.38
10	400	66.32

RSLbestPSO has created various combinations, assigning each set of tasks to a different number of VMs and computing the corresponding makespan time. First, 100 tasks are assigned to 5, 6, 7, 8, 9, and 10 VMs, followed by 150, 200, 250, 300, and 400 tasks assigned to different VMs, and it has been found that as the VMs are increasing, the makespan time is decreasing. **Table 5** below displays the computed makespan time results for each variation. The RSLbestPSO used a structured way to analyze all the combinations. First, 100 tasks are assigned to various VMs, and makespan time is computed. This is followed by taking a different set of tasks on the same varying number of VMs. The makespan time decreases as the VMs increase, thus showing effective parallel processing and lower response time. The makespan time produced by QMPSO (Babu & Philip, 2016) is 43.85 ms on 10 VMs when 100 tasks have been given. Similarly, the makespan time of an LMBPSO (Pradhan & Bisoy, 2022) on 5VMs is 108ms. The proposed RSLbestPSO algorithm achieves a makespan of approximately 35.91 ms, demonstrating a noticeable enhancement in performance when compared to both QMPSO and LMBPSO.

Table 5. Makespan time for different combinations of tasks and VMs.

Tasks	5 VMs	6VMs	7 VMs	8 VMs	9 VMs	10 VMs
100	161.65	137.98	87.36	64.5	55.78	35.91
150	168.91	143.1	104.83	70.21	62.85	39.12
200	175.7	152.21	111.42	81.66	69.02	45.88
250	187.71	158.83	117.86	87.53	77.42	52.78
300	199.74	165.88	125.28	94.28	83.38	58.21
400	227.26	178.94	138.16	114.36	98.99	66.32



(a) Different no. of tasks and VMs.

(b) Different combinations of tasks and VMs.

Figure 3. Makespan time plots computed by RSLbestPSO for different tasks and VMs.

Additionally, displays the scatter plot, which illustrates the variations in Makespan time. The decreasing trend in the different numbers of tasks and VMs shows that the makespan time increases as the tasks increase per VM. Conversely, the makespan time increases as the number of tasks increases for a particular

set of VMs. For a total of 5 VMs, at 100 tasks, the makespan time is 161.65s, and at 150 tasks, it rises to 168.91s. Therefore, the makespan time increases as the load on a particular set of VMs increases. Alternatively, when the number of tasks is fixed, and the corresponding number of VMs increases, the makespan time decreases.

Furthermore, the execution time of RSLbestPSO for different numbers of tasks assigned to 5 VMs is also computed and compared with several existing works comprising EBCA-LB (Babu & Philip, 2016), HBB-LB (Babu & Krishna, 2013), FL-GWO (Xingjun et al., 2020), LW-PSO (Malik & Suman, 2022), ICBEA (Ajil & Kumar, 2025), and DQL (Haris & Zubair, 2025), respectively, and is shown in **Table 6**. The results demonstrate the excellent performance of RSLbestPSO, balancing the load with execution times of 10.56s, 20.76s, 28.21s, 40.29s, 48.83s, and 60.32s for 100, 150, 200, 250, 300, and 400 tasks, respectively.

Table 6. Execution time computed by RSLbestPSO for different number of tasks.

Number of tasks	**100**	**150**	**200**	**250**	**300**	**400**
EBCA-LB (Babu & Philip, 2016)	77.8	98.55	61.75	165.5	192.25	221
HBB-LB (Babu & Krishna, 2013)	28.5	61.5	72.5	85.5	95	110
FL-GWO (Xingjun et al., 2020)	23.5	51	61.75	76.9	86.25	96.5
LW-PSO (Malik & Suman, 2022)	20.25	46.89	57.16	70.65	76.39	89.57
ICBEA (Ajil & Kumar, 2025)	72.14	-	90.32	-	149.6	216
DQL (Haris & Zubair, 2025)	47.81	-	59.71	-	99.28	133.87
RSLbestPSO	10.56	20.76	28.21	40.29	48.83	60.32

The computational cost is also evaluated. The EBCA, HBB-LB, FL-GWO, and LW_PSO have a cost of O(MDN), where M is the no. of particles, D is the distance covered in finding the solution, and N is the number of iterations. However, the computation complexity of RSLbestPSO is slightly higher due to adding extra randomness. The randomness is denoted by k. The complexity of RSLbestPSO will be O(MN(D+k)). So, if k is 0.2, the RSLbestPSO increases by 20% compared to normal PSO. However, this high computational cost is justifiable, leading to better exploration and faster convergence. Moreover, the bar plot of the comparison of RSLbestPSO with existing works is also plotted for execution time and shown in **Figure 4**. The comparison indicates that the RSLbestPSO takes less time when different numbers of tasks are assigned to 5 VMs. It is clear from the plot that RSLbestPSO outperformed existing techniques and showed an improvement of approximately 9.69%, 26.13%, 28.95%, 30.36%, 27.56%, and 29.25% for 100, 150, 200, 250, 300, and 400 tasks when compared with the second-best-performing technique.

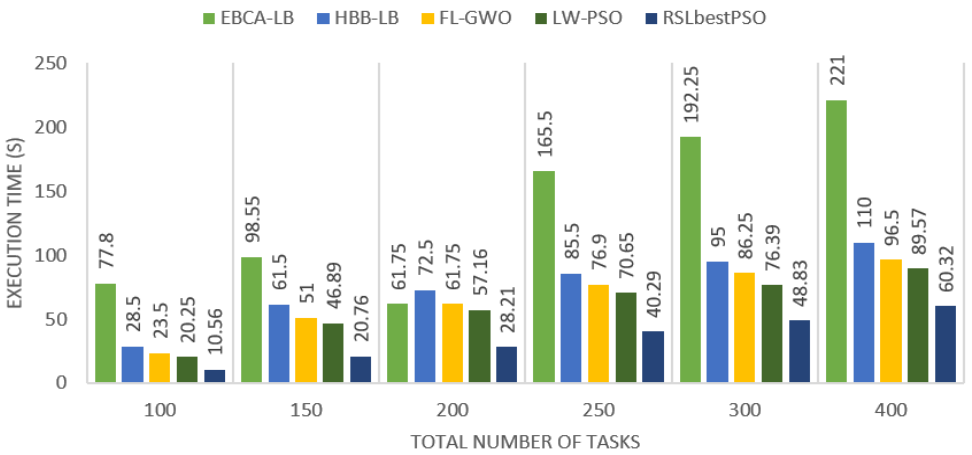
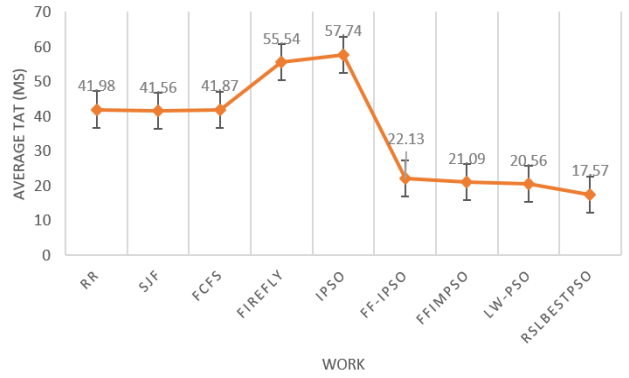


Figure 4. Bar plots for execution time for different numbers of tasks assigned to 5 VMs.

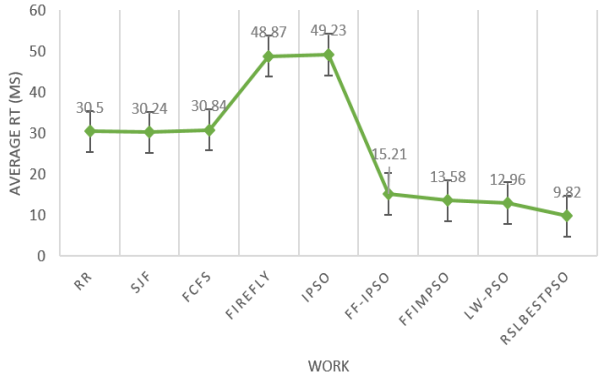
In the second-case scenario, we compute the average TAT, RT, and load for 1000 tasks assigned to 5 virtual machines. In terms of average TAT, RT, and load, RSLbestPSOLB is also compared with some static LB techniques such as RR (Devaraj et al., 2020), SJF (Devaraj et al., 2020), and FCFS (Devaraj et al., 2020), as well as some dynamic LB techniques such as Firefly (Devaraj et al., 2020), IPSO (Devaraj et al., 2020), FF-IPSO (Devaraj et al., 2020), FFIMPSO (Devaraj et al., 2020), and LW-PSO (Malik & Suman, 2022). **Table 7** shows the comparison results for TAT, RT, and load, and it is found that RSLbestPSO performed best with an average TAT of 17.57 ms, an average RT of 9.82 ms, and an average load of 0.19 ms, respectively. It takes less TAT, RT, and load than existing work.

Table 7. Comparison results of RLBestPSO in terms of Average TAT, RT, and load.

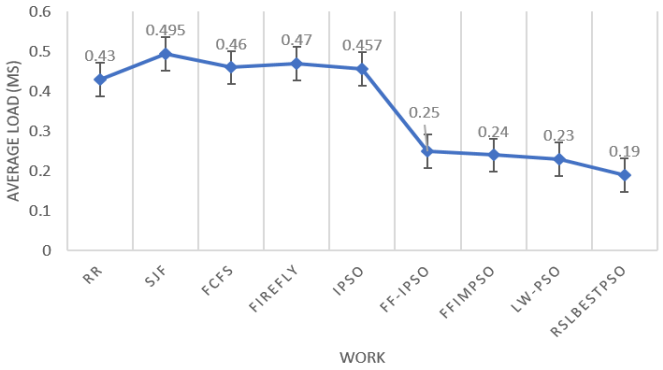
Work	Average TAT (ms)	Average RT (ms)	Average load (ms)
RR (Devaraj et al., 2020)	41.98	30.5	0.43
SJF (Devaraj et al., 2020)	41.56	30.24	0.495
FCFS (Devaraj et al., 2020)	41.87	30.84	0.46
Firefly (Devaraj et al., 2020)	55.54	48.87	0.47
IPSO (Devaraj et al., 2020)	57.74	49.23	0.457
FF-IPSO (Devaraj et al., 2020)	22.13	15.21	0.25
FFIMPSO (Devaraj et al., 2020)	21.09	13.58	0.24
LW-PSO (Malik & Suman, 2022)	20.56	12.96	0.23
RSLbestPSO	17.57	9.82	0.19



(a) Average TAT (ms).



(b) Average RT (ms).



(c) Average Load (ms).

Figure 5. Line plot of average TAT, RT, and load for RSLbestPSO and existing works.

The RSLbestPSO showed varied improvement when compared to existing work. RSLbestPSO reduced the response time by 60% compared to RR (Devaraj et al., 2020) and SJF (Devaraj et al., 2020). It shows an improvement of approximately 80% compared with Firefly (Devaraj et al., 2020) and IPSO (Devaraj et al., 2020). Similarly, RSLbestPSO improved the results by 60% in terms of load compared to existing works (Devaraj et al., 2020). In the case of TAT, RSLbestPSO improved the results by 15% and 58% compared with Devaraj et al. (2020) and Malik & Suman (2022). Furthermore, the line plots for average TAT, RT, and load have been plotted to compare RSLbestPSO with existing works. **Figure 5** displays the line plots, clearly demonstrating that RSLbestPSO performed effectively by taking less time. When compared with LW-PSO (Malik & Suman, 2022), it takes approximately 2.99 ms, 3.14 ms, and 0.04 ms less for TAT, RT, and load, respectively, and 3.52 ms, 3.76 ms, and 0.05 ms less for FFIMPSO (Devaraj et al., 2020).

Next is the third-case scenario, where the tasks are divided into four subtypes: small, medium, large, and extra-large, and their corresponding CPU and memory utilization are computed. **Table 8** shows the CPU and memory utilization results for RSLbestPSO and existing works (Devaraj et al., 2020). The comparison shows that the RSLbestPSO utilized more CPU, with 75% for small sets of tasks, 83% for medium tasks, 97% for large tasks, and 99% for extra-large tasks. Similarly, RSLbestPSO uses 64%, 77%, 89%, and 98% of the memory for small, medium, large, and extra-large tasks, respectively.

Table 8. Comparison of RSLbestPSO's CPU and memory utilization results with existing works.

Work	CPU utilization (%)				Memory utilization (%)			
	Small	Medium	Large	Extra-large	Small	Medium	Large	Extra-large
RD (Devaraj et al., 2020)	45	50	64	70	40	48	59	70
WRR (Devaraj et al., 2020)	48	58	67	75	44	53	63	75
DLB (Devaraj et al., 2020)	50	63	70	80	48	58	66	77
LB_BC (Devaraj et al., 2020)	57	68	75	85	54	62	70	80
FF-IPSO (Devaraj et al., 2020)	67	77	89	90	57	70	80	86
FFIMPSO (Devaraj et al., 2020)	70	79	95	97	60	72	83	89
RSLbestPSO	75	83	97	99	64	77	89	98

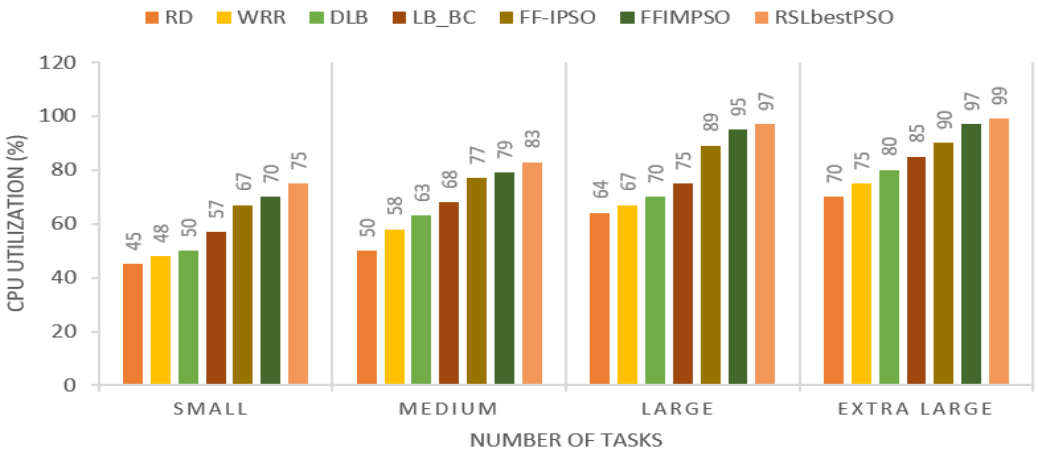
The improved performance is due to the random selection of neighbors in the proposed RSLbestPSO. This random selection will help in a better search, thereby resulting in better results. The personal best and global best positions are updated iteratively, and the resources are allocated efficiently, leading to lower resource utilization, execution, TAT, RT, and load. As the task is distributed and balanced, the CPU and Memory utilization will automatically be enhanced, leading to good results.

Additionally, **Figure 6** shows a bar plot for CPU and memory utilization that compares existing works with RSLbestPSO. RSLbestPSO performed effectively, showing improvements of 5%, 4%, 2%, and 2% in CPU utilization and 4%, 5%, 6%, and 9% in memory utilization, respectively.

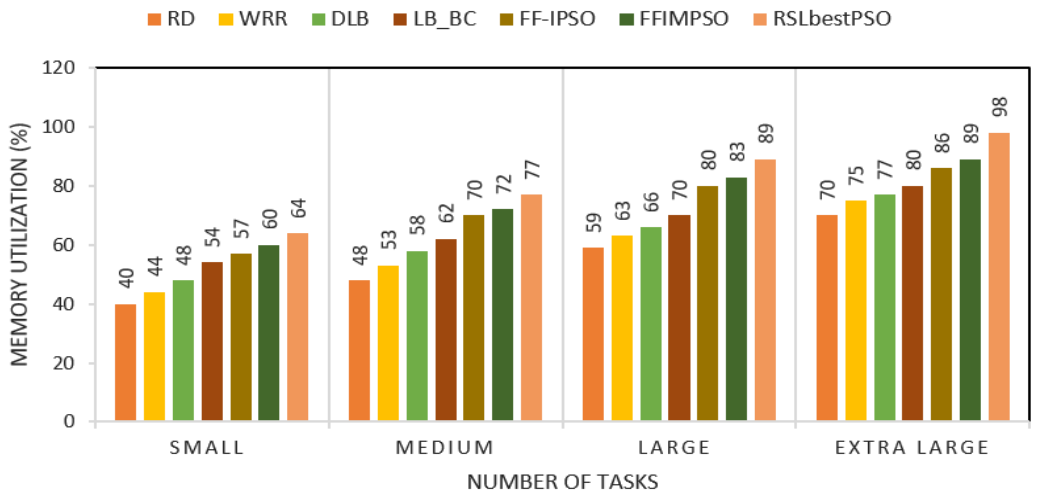
Table 9. RSLbestPSO results before and after parameter optimization.

Performance parameter	Result before optimization	Result after optimization
Makespan time	189.11s	161.65s
Execution time	15.78s	10.56s
Average TAT	19.88ms	17.57ms
Average RT	10.43ms	9.82ms
Average load	0.21ms	0.19ms
CPU utilization	72%	75%
Memory utilization	61%	64%

To show the superiority of the proposed RSLbestPSO, the results before and after optimization of performance parameters are also computed and shown in **Table 9**. The results are calculated for 100 tasks assigned to 5 VMs. The findings show that the RSLbestPSO shows effective performance with an improvement of 27.46s, 5.22s, 2.31ms, 0.61ms, 0.02ms, 3%, and 4% for makespan time, execution time, average TAT, RT, load, CPU utilization, and memory utilization, respectively.



(a) CPU utilization



(b) Memory utilization

Figure 6. Comparison of RSLbestPSO with existing works in terms of (a) CPU and (b) memory utilization.

The results showed a lower TAT, RT, and load. If the TAT is less in cloud based applications, it means that the requests made by the user have been executed efficiently with any delays, thereby improving the performance and responsiveness. Similarly, if the response time is low in gaming applications, there will be no lags in the game. Users will play the game without any disruption to achieve user satisfaction. If the load is low, the system will perform the task quickly, and zero fault tolerance will be achieved with less infrastructure cost.

The results are compared with the traditional PSO, genetic algorithm (GA), and ant colony optimization (ACO) and are shown in **Table 10**. The findings showed that RSLbestPSO performed best among the GA, ACO, and PSO in terms of makespan time, execution time, average TAT, average RT, average load, CPU utilization, and Memory utilization.

Table 10. Comparison of RSLbestPSO with GA, ACO, and PSO.

Parameter	GA	ACO	PSO	RSLbestPSO
Makespan time	195ms	189ms	170ms	161.65s
Execution time	18.14s	16.43s	14.10s	10.56s
Average TAT	27.98s	22.34s	19.46s	17.57ms
Average RT	13.89s	12.28s	10.55s	9.82ms
Average load	0.4ms	0.35ms	0.2ms	0.19ms
CPU utilization	68%	70%	74%	75%
Memory utilization	75%	69%	63%	64%

Different scenarios have been designed to test the efficiency of the proposed work. One scenario is that various tasks like 100, 150, 200, 250, 300, and 400 have been considered and allocated to different VM numbers of VMs, including 5, 6, 7, 8, 9, and 10, respectively. The makespan time is computed for all and it has been found that the proposed RSLbestPSO performed effectively. The other scenario is that the tasks are divided into small, medium, large, and extra-large sizes, and their corresponding CPU and Memory utilization is computed. The performance is evaluated for RSLbestPSO and the existing work, and RSLbestPSO outperformed, increasing CPU utilization.

The statistical analysis of RSLbestPSO is also done to show that statistical significance. The statistical significance shows whether the results are actual or achieved by chance. The Wilcoxon signed-rank test (p-value) is done in which the p-value is calculated. The model is statistically significant if the p-value is less than 0.05. The test was done, and it was found that one case was there where RSLbestPSO was not statistically significant as the value went beyond 0.05. In the rest of the cases, i.e., in RSLbestPSO vs. GA and RSLbestPSO vs. PSO, the p-value achieved is less than 0.05, which shows that the proposed model RSLbestPSO is statistically significant. The results are shown in **Table 11**.

Table 11. Statistical analysis of RSLbestPSO with ACO, GA, and PSO.

Model	Wilcoxon signed-rank test (p-value)
RSLbestPSO vs ACO	0.1484
RSLbestPSO vs. GA	0.0391
RSLbestPSO vs PSO	0.049

Moreover**,** we have increased the dataset size from 400 to 2000 tasks distributed on 100 VMs to check the performance of the proposed RSLbestPSO. We have computed the throughput, standard deviation, and efficiency to consider the other statistical and performance parameters. The results are compared with standard PSO, GA, and ACO and are shown in **Table 12**. The results showed that RSLbestPSO performed well with 14.67, 0.4, and 0.085 throughput values, standard deviation, and efficiency, respectively. The results show that increasing the scalability in terms of tasks does not affect the performance of the proposed RSLbestPSO. Therefore, the proposed RSLbestPSO is scalable and has increasing tasks and VMs. Furthermore, energy consumption while implementing the proposed work is also computed based on the different resources (CPU, memory, and network) and power consumed. The energy consumption is given by the following Equation (3).

$$Energy = \sum_{i=0}^{n} Power_i * Time_i \quad (3)$$

where, *n* are the different resources. The energy consumption is given in **Table 12**, and it has been found that RSLbestPSO consumed a total of 1400 Joules (J) to perform the experiment.

Table 12. Comparison of RSLbestPSO in terms of increased data size.

Model	Throughput	Standard deviation	Efficiency	Energy consumption (J)
GA	11.32	0.42	0.21	1200
ACO	12.65	0.5	0.18	1300
PSO	12.33	0.43	0.15	1250
RSLbestPSO	14.67	0.4	0.085	1400

4.3 Technological Constraints

Although RSLbestPSO performed best in makespan time, throughput, RT, CPU utilization, memory utilization, and average load, there are some real-world applications where RSLbestPSO performs sub-optimally. As the computational complexity of RSLbestPSO is high, it may take a lot of time for tasks like video surveillance and IoT-driven healthcare services. In these applications, the RSLbestPSO takes a longer convergence time to find the best solution. Also, the current RSLbestPSO is static; we are initially predefining the number of tasks and VMs. So, if the VMs fail or the task size increases, RSLbestPSO may not be able to adapt to the changes.

5. Conclusion and Future Scope

The present research addresses task scheduling issues in cloud systems by introducing RSLbestPSO, a cooperative AI-LB approach. This method seeks to maximize cloud computing's heterogeneous network performance metrics and virtual machine selection. The study considers optimized factors such as makespan time, execution time, average TAT, average RT, average load, CPU usage, and memory utilization. The results show that the RSLbestPSO outperforms current state-of-the-art methods in terms of efficiency. Improvements in CPU and memory usage, along with decreases in makespan time, execution time, average load, RT, and TAT, demonstrate this. We compute the findings using three alternative scenarios, assigning different amounts of jobs to varying numbers of virtual machines to determine the makespan time. By analyzing every possible combination, the program determines the makespan time. With the previous research (Devaraj et al., 2020; Malik & Suman, 2022), we calculate and compare the second case's average system load, reaction time, and turnaround time. According to the results, the RSLbestPSO algorithm minimizes average reaction time, load, and turnaround time by 3.52 ms, 3.76 ms, and 0.05 ms, respectively. In the third hypothetical scenario, the jobs fall into one of four sizes: short, medium, big, or extra-large. We will use five VMs to divide the jobs by using the RSLbestPSO method. RSLbestPSO's CPU and memory use calculation is compared with current methods (Malik & Suman, 2022). Going with the results, RSLbestPSO performs well, showing improvement in the usage of CPU 5%, 4%, 2%, and 2% and in memory consumption of 4%, 5%, 6%, and 9% for small, medium, big, and extra-large workloads, respectively. To show the effectiveness of RSLbestPSO, the outcome of parameter optimization before and after is also calculated. The job is assigned to virtual machines (VMs) using the RSLbestPSO method while avoiding overloading and underloading, showing effective results in a small amount of time. This research aims to make it easy for VMs to attain minimum LB, which will reduce the difficulties in task scheduling. The proposed RSLBestPSO is also effective for routing or energy-aware VM scheduling. Although the proposed work is practical, various deep learning algorithms like CNN and LSTM algorithms are still required for irregular task patterns. Reinforcement learning is expected to divide the workload across diverse cloud networks effectively. For efficient load balancing in a heterogeneous cloud network, the multi-agent independent Deep-Q network algorithm, a non-cooperative load balancing technique, is supposed to be proposed. Although the proposed RSLbestPSO performed effectively, there are some real-world fluctuations like video surveillance, and the performance may be hindered. In those cases,

reinforcement learning performed better in which agents dynamically learned the adaptive task allocation. Moreover, new algorithms like sine cosine optimization will be hybridized with evolutionary or swarm optimization for better load balancing.

Conflict of Interest

The authors declare that they have no competing interests.

Acknowledgments

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The authors would like to thank the editor and anonymous reviewers for their comments that helped improve the quality of this work.

AI Disclosure

The author(s) declare that no assistance is taken from generative AI to write this article.

References

Ahuja, K., Singh, B., & Khanna, R. (2018). Network selection in wireless heterogeneous environment by CPF hybrid algorithm. *Wireless Personal Communications*, *98*(3), 2733-2751. https://doi.org/10.1007/s11277-017-4998-1.

Ajil, A., & Kumar, E.S. (2025). IDBNWP: improved deep belief network for workload prediction: hybrid optimization for load balancing in cloud system. *Multimedia Tools and Applications*, *84*(16), 15715-15733. https://doi.org/10.1007/s11042-024-19495-z.

Babu, K.R.R., & Philip, S. (2016). Enhanced bee colony algorithm for efficient load balancing and scheduling in cloud. In: Snášel, V., Abraham, A., Krömer, P., Pant, M., Muda, A. (eds) *Innovations in Bio-Inspired Computing and Applications.* Springer International Publishing, Cham, pp. 67-78. ISBN: 978-3-319-28031-8. https://doi.org/10.1007/978-3-319-28031-8_6.

Babu, L.D.D., & Krishna, P.V. (2013). Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing Journal*, *13*(5), 2292-2303. https://doi.org/10.1016/j.asoc.2013.01.025.

Dam, S., Mandal, G., Dasgupta, K., & Dutta, P. (2014). An ant colony based load balancing strategy in cloud computing. In: Kundu, M.K., Mohapatra, D.P., Konar, A., Chakraborty, A. (eds) *Advanced Computing, Networking and Informatics* (Vol. 2, pp. 403-413). Springer International Publishing, Cham. ISBN: 978-3-319-07350-7. https://doi.org/10.1007/978-3-319-07350-7_45.

Dasgupta, K., Mandal, B., Dutta, P., Mandal, J.K., & Dam, S. (2013). A genetic algorithm (GA) based load balancing strategy for cloud computing. *Procedia Technology*, *10*, 340-347. https://doi.org/10.1016/j.protcy.2013.12.369.

Devaraj, A.F.S., Elhoseny, M., Dhanasekaran, S., Lydia, E.L., & Shankar, K. (2020). Hybridization of firefly and improved multi-objective particle swarm optimization algorithm for energy efficient load balancing in cloud computing environments. *Journal of Parallel and Distributed Computing*, *142*, 36-45. https://doi.org/10.1016/j.jpdc.2020.03.022.

Dhillon, A., Singh, A., & Bhalla, V.K. (2023). Biomarker identification and cancer survival prediction using random spatial local best cat swarm and Bayesian optimized DNN. *Applied Soft Computing*, *146*, 110649.

Gamal, M., Rizk, R., Mahdi, H., & Elnaghi, B.E. (2019). Osmotic bio-inspired load balancing algorithm in cloud computing. *IEEE Access*, *7*, 42735-42744. https://doi.org/10.1109/access.2019.2907615.

Haris, M., & Zubair, S. (2025). Battle royale deep reinforcement learning algorithm for effective load balancing in cloud computing. *Cluster Computing*, *28*(1), 19. https://doi.org/10.1007/s10586-024-04718-7.

Houssein, E.H., Gad, A.G., Wazery, Y.M., & Suganthan, P.N. (2021). Task scheduling in cloud computing based on meta-heuristics: review, taxonomy, open challenges, and future trends. *Swarm and Evolutionary Computation*, *62*, 100841. https://doi.org/10.1016/j.swevo.2021.100841.

Huang, X., Li, C., Chen, H., & An, D. (2020). Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies. *Cluster Computing*, *23*(2), 1137-1147. https://doi.org/10.1007/s10586-019-02983-5.

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

Kumar, M., & Sharma, S.C. (2020). Dynamic load balancing algorithm to minimize the makespan time and utilize the resources effectively in cloud environment. *International Journal of Computers and Applications*, *42*(1), 108-117. https://doi.org/10.1080/1206212x.2017.1404823.

Malik, M., & Suman. (2022). Lateral wolf based particle swarm optimization (LW-PSO) for load balancing on cloud computing. *Wireless Personal Communications*, *125*(2), 1125-1144. https://doi.org/10.1007/s11277-022-09592-3.

Marini, F., & Walczak, B. (2015). Particle swarm optimization (PSO). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, *149*(part B), 153-165. https://doi.org/10.1016/j.chemolab.2015.08.020.

Mishra, S.K., Sahoo, B., & Parida, P.P. (2020). Load balancing in cloud computing: a big picture. *Journal of King Saud University - Computer and Information Sciences*, *32*(2), 149-158. https://doi.org/10.1016/j.jksuci.2018.01.003.

Mondal, B., & Choudhury, A. (2015). Simulated annealing (SA) based load balancing strategy for cloud computing. *International Journal of Computer Science and Information Technologies*, *6*(4), 3307-3312.

Pradhan, A., & Bisoy, S.K. (2022). A novel load balancing technique for cloud computing platform based on PSO. *Journal of King Saud University - Computer and Information Sciences*, *34*(7), 3988-3995. https://doi.org/10.1016/j.jksuci.2020.10.016.

Prassanna, J., & Venkataraman, N. (2019). Threshold based multi-objective memetic optimized round robin scheduling for resource efficient load balancing in cloud. *Mobile Networks and Applications*, *24*(4), 1214-1225. https://doi.org/10.1007/s11036-019-01259-x.

Rashid, A., & Chaturvedi, A. (2019). Cloud computing characteristics and services a brief review. *International Journal of Computer Sciences and Engineering*, *7*(2), 421-426. https://doi.org/10.26438/ijcse/v7i2.421426.

Saeed, F., Javaid, N., Zubair, M., Ismail, M., Zakria, M., Ashraf, M.H., & Kamal, M.B. (2019). Load balancing on cloud analyst using first come first serve scheduling algorithm. In: Xhafa, F., Barolli, L., Greguš, M. (eds) *Advances in Intelligent Networking and Collaborative Systems*. Springer International Publishing, Cham, pp. 463-472. https://doi.org/10.1007/978-3-319-98557-2_42.

Simaiya, S., Lilhore, U.K., Sharma, Y.K., Rao, K.B.V.B., Maheswara Rao, V.V.R., Baliyan, A., Bijalwan, A., & Alroobaea, R. (2024). A hybrid cloud load balancing and host utilization prediction method using deep learning and optimization techniques. *Scientific Reports*, *14*(1), 1337. https://doi.org/10.1038/s41598-024-51466-0.

Sundas, A., & Panda, S.N. (2020). An introduction of cloudsim simulation tool for modelling and scheduling. In *2020 International Conference on Emerging Smart Computing and Informatics* (pp. 263-268). IEEE. Pune, India. https://doi.org/10.1109/esci48226.2020.9167549.

Téllez, N., Jimeno, M., Salazar, A., & Nino-Ruiz, E.D. (2018). A tabu search method for load balancing in fog computing. *International Journal of Artificial Intelligence*, *16*(2), 1-30.

Tripathy, S.S., Mishra, K., Roy, D.S., Yadav, K., Alferaidi, A., Viriyasitavat, W., Sharmila, J., Dhiman, G., & Barik, R.K. (2023). State-of-the-art load balancing algorithms for mist-fog-cloud assisted paradigm: a review and future directions. *Archives of Computational Methods in Engineering, 30*(4), 2725-2760. https://doi.org/10.1007/s11831-023-09885-1.

Upadhyay, S.K., Bhattacharya, A., Arya, S., & Singh, T. (2018). Load optimization in cloud computing using clustering: a survey. *International Research Journal of Engineering and Technology*, *5*(4), 2455-2459.

Waheed, M., Javaid, N., Fatima, A., Nazar, T., Tehreem, K., & Ansar, K. (2019). Shortest job first load balancing algorithm for efficient resource management in cloud. In: Barolli, L., Leu, F.Y., Enokido, T., Chen, H.C. (eds) *Advances on Broadband and Wireless Computing, Communication and Applications.* Springer International Publishing, Cham, pp. 49-62. https://doi.org/10.1007/978-3-030-02613-4_5.

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

Young, A.L. (1995). Issues and challenges [biotechnology for bioengineers]. *IEEE Engineering in Medicine and Biology Magazine*, *14*(2), 204-206. https://doi.org/10.1109/51.376762.

Zahid, M., Javaid, N., Ansar, K., Hassan, K., Khan, M.K., & Waqas, M. (2019). Hill climbing load balancing algorithm on fog computing. In: Xhafa, F., Leu, F.Y., Ficco, M., Yang, C.T. (eds) *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*. Springer International Publishing, Cham, pp. 238-251. ISBN: 978-3-030-02607-3. https://doi.org/10.1007/978-3-030-02607-3_22.