

Optimizing cloud energy with deep reinforcement learning: an artificial bee colony-Q-learning hybrid approach

Mohammed Ziaur Rahman¹, Anandaraj Shanthi Pichandi²

¹Department of Computer Science and Engineering, School of Engineering and Technology, Sapthagiri NPS University, Bengaluru, India

²School of Computer Science and Engineering, Presidency University, Bengaluru, India

Article Info

Article history:

Received May 26, 2025

Revised Jul 27, 2026

Accepted Aug 12, 2026

Keywords:

Artificial bee colony

Deep reinforcement learning

Load balancing

Q-learning

Virtual machines

ABSTRACT

Load balancing in the cloud is a crucial characteristic for distributing resources from data centers. However, load balancing in a cloud environment remains a challenging issue due to the influx of numerous incoming tasks, particularly in an infrastructure as a service (IaaS) cloud environment. To address the complexity and high dimensionality of the process, machine learning (ML) techniques combined with optimization algorithms are used, which fasten the process and minimize execution time. The deep reinforcement learning (DRL) – artificial bee colony with Q-learning (ABCQ) approach is proposed to solve load balancing in a cloud environment. The agent in DRL makes intelligent decisions to handle a large number of tasks efficiently at lesser time and increasing speed. The artificial bee colony (ABC) algorithm uses a global guiding factor to initialize position updates, enhancing the searchability and performance of the algorithm. The proposed DRL-ABCQ approach optimizes scheduling and resource utilization, increases the throughput of virtual machines (VMs), and balances the load between VMs. The DRL-ABCQ approach achieves a makespan of 125.52 s, energy utilization of 99.81 kJ, and a throughput of 2.39 jobs/s, compared to existing approaches.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Mohammed Ziaur Rahman

Department of Computer Science and Engineering, School of Engineering and Technology

Sapthagiri NPS University

Bengaluru, India

Email: mohammedziaur@snpsu.edu.in

1. INTRODUCTION

In recent times, the cloud network has become a popular technique that offers various services as per user requests across internet [1]. Cloud computing (CC) provides users with the ability to access computing resources or enables them to purchase required cloud services on-demand via the internet, based on applications [2]–[4]. The CC system handles various services based on processing platforms requested by users [5]. CC is the integration of parallel and distributed computing with resource sharing, such as hardware and software, which operates on a “pay-as-you-go” model [6]. Infrastructure as a service (IaaS) is a technique behind server management, data centers, and virtual machines (VMs) [7]. IaaS provides a cloud service that gives users access to servers or cloud computers to process and store data. Users can operate applications or servers on rental servers, with maintenance and hardware costs [8]. Cloud infrastructure processes on VMs, and IaaS offers benefits including providing user access to servers in the closest locations. Because of the dynamic nature of the cloud environment, loads fluctuate over time, making it challenging to allocate the appropriate VM [9]. This issue leads to high costs, lower profits

for organizations, and reduced model performance [10]. Each physical machine consumes electrical power based on the tasks it performs [11], and consumption increases as the server handles numerous workloads, causing high energy usage [12]. To mitigate this issue, an effective scheduling algorithm is required to manage load between VMs, compute each incoming task with less completion time, and reduce energy consumption in the data center [13].

In a cloud environment, incoming tasks are assigned to the proper VM to allocate necessary resources for computation [14]. Though, when suitable resources are unavailable or server is heavily loaded, incoming tasks require more time for completion [15]. Traditional approaches, like round-robin and weighted load balancing, fail to adapt to real-time fluctuations in resource demand. Additionally, while heuristic-based algorithms can be efficient in certain scenarios, they lack effectiveness in predicting workloads and proactively adjusting resource allocation. These limitations result in underutilization of resources, increased response times, and degraded quality of service (QoS) for cloud users. To mitigate these issues, intelligent load balancing strategies utilizing machine learning (ML) and deep learning (DL) have gained attention. These algorithms dynamically predict workload patterns, optimize resource scheduling, and improve overall effectiveness. This manuscript aims to develop an adaptive load balancing mechanism that combines ML/DL algorithms to address existing limitations and enhance resource distribution in IaaS-enabled cloud environments.

Jena *et al.* [16] suggested the hybridization of modified particle swarm optimization (MPSO) and an enhanced Q-learning approach, named QMPSO, for dynamic load balancing between VMs. The hybridization process adjusted the MPSO velocity through pbest and gbest values of better actions produced by enhanced Q-learning. The goal of the hybridization was to enhance the method's performance through balancing load between VMs, increasing VM throughput, and maintaining task priority balance through optimizing task waiting times. However, the suggested method consumed a significant amount of energy due to the heavy load in the network. Pradhan and Bisoy [17] presented a load balancing method called modified particle swarm optimization (LBMP SO), which scheduled tasks across existing cloud resources, reduced makespan, and increased resource utilization. This was achieved through appropriate matching between resources and tasks within the data center. The proposed method performed better in minimizing makespan and maximizing resource utilization. However, it consumed excessive time due to heavy load conditions. Haris and Zubair [18] employed load balancing approach depending on hybrid approaches, namely mantaray modified multi-objective Harris Hawk optimization (MMHHO). This hybridization enhanced Harris Hawk optimization (HHO) search through integrating the manta ray foraging optimization (MRFO) approach, considering factors like response time, cost, and resource usage. The developed method improved performance by improving the throughput of VMs, balancing load between VMs, and maintaining balance between task priorities through adjusting task waiting times. Though, the method still faced issues related to load balancing and suboptimal resource utilization.

Souravlas *et al.* [19] implemented a workload-per-VM strategy for task load balancing, which aimed to enhance mean response time and system makespan in the cloud. The issue was formulated as a finite Markov process, referred to as the balance equation for each state. By balancing the potential states, the expected usage of VMs could be derived, playing an essential role in the task allocation process. The load balancer acted as the central server, utilizing fair task allocation to distribute incoming tasks, balance the VMs, and account for their current state and processability. However, the implemented method had high loads on VMs because it did not consider optimal solutions. Lahande *et al.* [20] introduced a load balancing mechanism by experimenting and executing tasks utilizing the Sipht task dataset. The RS approaches: first come first served (FCFS), maximum-minimum (max-min), maximum completion time (MCT), and round robin (RR) were considered for balancing execution loads across VMs. The experiment was performed in four stages, with each stage including 16 scenarios, where each scenario varied based on the number of VMs utilized. However, the introduced method had high energy consumption and makespan due to heavy load conditions. By utilizing the artificial bee colony with Q-learning (ABCQ) approach, once a better reward was obtained, the method sought to achieve a global optimal reward in less computation time by exchanging data with neighboring nodes. The essential contributions of this research are described as follows:

The deep reinforcement learning (DRL) – ABCQ approach is proposed to balance load between VMs in cloud, maximizing reward through minimizing makespan time and energy consumption. The agent in DRL makes intelligent decisions to handle huge number of tasks in less time and at higher speed. The artificial bee colony (ABC) algorithm uses the global guiding factor in the position updating phase, which improves the searchability of traditional ABC in finding optimal solutions.

This research paper is organized as follows. Section 2 presents the details of the proposed DRL-ABCQ approach. Section 3 presents the outcomes and discussion. Finally, section 4 provides the conclusions of this research.

2. METHOD

This paper proposes the DRL-ABCQ scheduling approach, where the agent receives a separate reward by considering the action taken in every state of the environment. The ABCQ approach aims to achieve a globally better reward with less computation time by exchanging data with neighboring nodes after obtaining better individual rewards. Initially, the appropriate action is chosen by considering the value function of state-action, and the personal better reward is then obtained for every server, which optimizes load and its parameters. After achieving personal better rewards, every server identifies global better value through sharing their data between servers at high speed. The following section explains the reward function, DRL, Q-learning, and ABC algorithms. Figure 1 represents a process of load balancing in the cloud using the DRL-ABCQ approach.

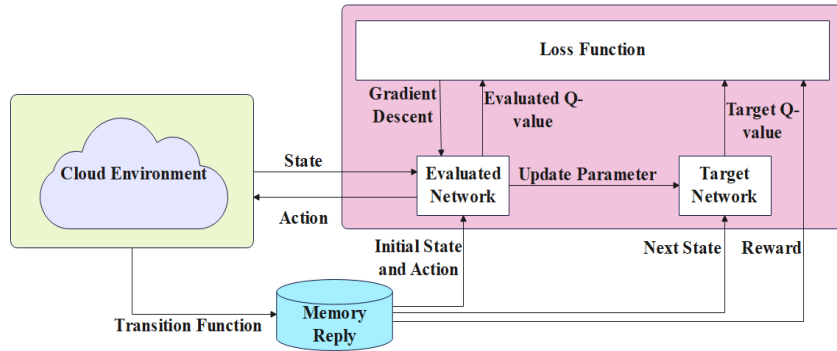


Figure 1. Process of load balancing in cloud

2.1. Reward function

The problem of load balancing is defined as Markov decision process (MDP) because of the simultaneous nature of tasks in a cloud environment. In a developed approach, data center controller is described as an agent, and data center itself is described as the environment, where agent allocates incoming tasks to the appropriate VM in every cycle. The MDP includes four significant variables, namely state, action, transition function, and reward, which are explained as follows [21].

2.1.1. State and action space

In the state space, a perfect move is made by the agent depending on VM data like amount of VMs, the existence of millions of instructions per second (MIPS), the central processing unit (CPU), bandwidth, and memory. The mathematical expression for the state space is given in (1). In the action space, every task is employed to VM for processing. Every action includes different pieces of data like number of tasks, length of task and size of the file. The mathematical expression for the action space is given in (2).

$$S = \{VM_v, VM_{mips}, VM_{cpu}, VM_{mem}, VM_{bw}\} \quad (1)$$

$$A = \{T_w, T_{leng}, T_{fs}\} \quad (2)$$

2.1.2. Transition function and action selection

This phase represents when the agent considers the action in the present state, which reaches the new state. The agent always attempts to select the appropriate action that results in an optimum state, which provides a highest reward. The transition function is described as $P(s_t, a_t, r_t, s_{t+1})$. It represents the possibility of reaching the following state s_{t+1} and obtaining the reward r_t after executing the chosen action a_t in the present state s_t . Due to the cloud's dynamic nature, tasks vary with respect to the length, time and size of a file [22]–[24]. Certain VMs are overloaded, while some are underloaded, as defined using (3). For balancing load between VMs, the agent considers an action to transfer extra tasks from an overloaded VM to underloaded VM at high speed. The speed of task transfer is measured using (4).

$$VM_{id} = \begin{cases} < |TH_{value}| & \text{Underload} \\ > |TH_{value}| & \text{Overload} \\ = |TH_{value}| & \text{Balanced} \end{cases} \quad (3)$$

$$T_{ts} = \sum_{w=1}^n \sum_{x=1}^o \sum_{y=1}^o (1 - DV_{w,x} DV_{w,y}) \frac{ETTR_{x,y}}{BW_{x,y}} \quad (4)$$

Where $ETTR_{x,y}$ represents the additional task that is transferred between x to y VM; $BW_{x,y}$ represents the required bandwidth between x and y VM. Here, $DV_{w,x} = 1$ signifies that x VM is overloaded, where $DV_{w,y} = 1$ represents that the extra task is transferred from x to y VM. When task transfer speed is increased, task completion time also reduces. The chosen action a in iteration t is measured by (5).

$$a_t = \max_a (T_{ts}) \quad (5)$$

2.1.3. Reward

In this phase, the agent receives a reward for making moves in different states. The agent aims to select the state that yields higher rewards, thereby increasing accuracy. In the proposed method, the reward function is characterized through a reduction in task completion time on a particular VM and energy consumption. The mathematical expression describing the reward is given in (6).

$$r_t = \{minCT_{w,v} + E_{cons}\} \quad (6)$$

2.2. Q-learning approach

Q-learning is a reinforcement learning (RL) approach that helps the agent learn how to act optimally in a given environment. Q-learning aims to identify the optimal policy for action selection that minimizes the expected sum of rewards for the agent. Consider a set of states $S = \{s_1, s_2, s_3, \dots, s_n\}$ in the environment, where every state contains a set of actions. The agent $A = \{a_1, a_2, a_3, \dots, a_m\}$ chooses an action $a_t \in A$ at time t in state $s_t \in S$ to transition to the next state $s_{t+1} \in S$, receiving the reward r_{t+1} from the environment. For task processing, it is essential to choose relevant actions to increase Q-value of every state, that is crucial for identifying the optimum policy in the cloud environment. The Q-value function is based on the action chosen in a state. Figure 2 illustrates the process of Q-learning.

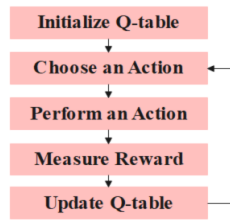


Figure 2. Process of Q-learning

Consider an agent in state s_t and the chosen action a_t . The process of Q-value is directed towards a better state, maximizing total reward in the environment. The mathematical expression for the Q-value is given in (7), where α represents learning rate, γ represents the discount factor, which influences successive states based on past actions, and r_t represents the reward obtained from processing the action in state s_t .

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})] \quad (7)$$

The Q-value is derived from Q-table, which stores all probable states, their corresponding Q-values, and the correct actions. The Q-learning approach attempts to estimate the optimum state from its experience and incorporates a greedy strategy within the Q-learning process. The mathematical expression for computing the Q-value is given in (8), where α represents the learning rate and δ represents the transition function.

$$Q_{(t+1)}(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma \llbracket \max \rrbracket_{at} [Q_t(\delta(s_t, a_t), a'_t)] - Q_t(s_t, a_t)] \quad (8)$$

2.2.1. Fitness function

The fitness function of the DRL-ABCQ approach is based on makespan, waiting time, and resource utilization. The mathematical expression for the fitness function is given in (9). Makespan is minimized when the highest number of tasks are successfully processed in the data center, which impacts the model's throughput. MS represents the total completion time for each task, and the mathematical expression is given in (10). Next, R_u is maximized, and its mathematical expression is given in (11). R_u is measured as the total completion time of the tasks divided by the highest makespan or execution time. Where F is fitness function, WT is waiting time, MS is makespan, and R_u is resource utilization. The algorithm for Q-update is given as in Algorithm 1.

$$F = WT + MS + R_u \quad (9)$$

$$MS = \max\{ET_i\} \quad (10)$$

$$(R_u) = \frac{ET_i}{MS} \quad (11)$$

Algorithm 1. For Q-value updating

```

Initialize values to  $\alpha$  (learning rate),  $\gamma$  (discount factor) and  $r$  (reward)
Initialize the Q-values
for  $i = 1$  to  $n$ 
    for  $j = 1$  to  $p$ 
         $Q(s_i, a_j) = 0$ 
    end for
end for
choose action  $a_1$  form the agent  $A = \{a_1, a_2, a_3, \dots, a_m\}$  and compute that and to following state  $s_{t+1}$ 
Calculate learning rate
Compute reward
Update  $Q_{t+1}$  using (8)
Repeat this phase for the new state till that converges

```

2.3. Artificial bee colony with global guiding factor

The ABC algorithm with a global guiding factor is a swarm intelligence technique which mimics foraging behavior of bees. The algorithm is divided to three phases: leader, follower, and scouter. The leader represents the food source and carries information about it. In ABC, agents are classified based on the roles of bees: employed, onlooker, and scout bees [25]. Each bee group exhibits different behaviors related to exploration and exploitation. The exploitation behavior involves searching for new food sources to avoid local optima, while the exploration behavior focuses on searching for optimal food sources near the current one. In this research, Q-learning is utilized to enhance the solutions and obtain many correct solutions to the problem. The ABC algorithm is characterized through strong exploration behavior and weaker exploitation behavior; thus, goal is to optimize the exploitation behavior. During the initialization stage, ABC algorithm defines a position of food source with possible solutions to issue. First, the food source is randomly produced, and the mathematical expression for this is given in (12).

$$x_{i,j}^0 = x_j^{\min} + \text{rand}(0,1) \times (x_j^{\max} - x_j^{\min}) \quad (12)$$

Where $x_j^{\min}$ defines lower bound of j th optimization parameter and $x_j^{\max}$ defines the upper bound of j th parameter of optimization. Additionally, the initialization stage ABC algorithm divides algorithm into three sub-stages: employed, onlooker, and scout bee phases. The approach repeats the three stages until the highest number of iterations is attained. In employee bee stage, the algorithm identifies position of neighboring food source relative to the current food source. The current location update expression has strong search capabilities but lacks sufficient exploration capabilities. This limitation often causes the algorithm to fall into local optima and reduces the update speed. To overcome this issue, the position updating formula incorporates global guiding factors, and the updated mathematical expression for position update is given in (13).

$$v_{i,j}^t = x_{i,j}^t + r_{i,j}(x_{m,j} - x_{k,j}) + \mu(x_{best,j} - x_{i,j}) \quad (13)$$

Where $v_{i,j}^t$ represents the new position produced closer to $x_{i,j}^t$, k, m and j represent random numbers produced by random formulas, and $x_{best,j}$ represents the best value. The global guiding factor $(x_{best,j} - x_{i,j})$ is included to enable the bee search to have a higher direction, and the μ influencing factor is included as the global factor for constraining the search amplitude. At this stage, the value of a Q-table is updated by using (8). If a food source provides good fitness value, employee bee does not replace current food source with a new source. Instead, it updates Q-value through rewarding newly chosen food source and penalizing current food source. Similarly, if a new food source does not provide good fitness value, it is penalized, while current food source is rewarded. The Q-table is updated every time the employee bee identifies an appropriate food source. Hence, number of employee bees is denoted as *Emp*, and Q-table is updated once for each employee bee. In an onlooker stage, the onlooker bee selects a food source from Q-table based on Q-values. The onlooker bee searches for a new food source, and mathematical expression for this is given in (14). If a new food source has high fitness value, it replaces present food source, and Q-value is updated accordingly.

$$v_{i,j}^t = x_{i,k}^t + \phi_{i,j}^t \cdot r_j^t \cdot (x_{i,k}^t - x_{RFS,k}^t) \quad (14)$$

Where $v_{i,j}^t$ represents the parameter for optimization in the neighboring food source, where $\emptyset \in [-1,1]$, and $x_{RFS,k}^t$ represent parameter for optimization in the optimum food source, which is selected through random selection. Using the (14) improves exploitation, rather than just updating the dimension values. The onlooker bees exploit the current food source and update all dimensions with varying weight values, and the Q-values are subsequently updated. In the scout bee stage, if there have been several successful attempts at identifying the best neighbors, the current food source is discarded, and the scout bee randomly searches for new food source. The algorithm for ABC with a global guiding factor is described as in Algorithm 2.

Algorithm 2. ABC with a global guiding factor

```

Initialization
Initialize the population and measure the fitness value
Set parameters such as best solution, highest number of iterations, population size
Identify good solution
Employee Bees stage
    for every position do
        Update employee bee position through (13)
        Evaluate new position
        if fitness value is good replace the present location with a new
location
        end if
        Measure the possibility and update the Q-table to choose a location in
onlooker bee stage
    end for
Onlooker Bees stage
    for every onlooker do
        choose the location depended on Q-value and probability
        Update the location of the onlooker bee through (14)
        Evaluate new location
        if fitness value is good replace the present location with a new
location
        end if
        Update Q-table by (8)
    end for
    Identify a better solution (Q-table)
Scout Bees stage
    for every position do
        Abandon solutions which are not updated and produce new solutions randomly
        Update Q-table by (8)
    end for
end while

```

3. RESULTS AND DISCUSSION

The performance of a DRL-ABCQ approach is simulated in a CloudSim environment with following configurations: an i5 processor, Windows 10 OS, and 8 GB of RAM. The simulation involves two different experiments to assess the algorithm's effectiveness. The initial experiment uses two sets of tasks to evaluate reward percentage and compares energy consumption and computation time of the DRL-ABCQ approach with existing approaches. The second experiment focuses on measuring the accuracy of the DRL-ABCQ approach. Performance metrics such as makespan, load balancing index, throughput, energy consumption, completion time, and reward are considered to evaluate the DRL-ABCQ approach's performance. Key parameters of the proposed DRL-ABCQ algorithm include a learning rate of 0.01, which controls the method's adaptation speed and balances convergence with stability. The exploration-exploitation tradeoff ensures adaptability while preventing sub-optimal decisions. The batch size of 64 impacts the training speed and stability, while the discount factor ($\gamma = 0.9$) strikes a balance between short-term and long-term rewards. Additionally, parameters of the ABC algorithm are set to 50 bees and a maximum iteration of 100, which enhances optimization without excessive computational overhead. Fine-tuning these parameters ensures a balance between energy efficiency and overall performance. Table 1 presents a simulation parameters of a DRL-ABCQ approach.

Table 2 presents the performance of the DRL-ABCQ approach evaluated on different tasks 2,000 and 4,000. The particle swarm optimization (PSO), grey wolf optimization (GWO), whale optimization algorithm (WOA), and dwarf mongoose optimization (DMO) are the existing algorithms utilized to validate performance of DRL-ABCQ approach. In Table 2, the metrics considered include makespan, index of load balancing, throughput, energy consumption, completion time, and reward. The DRL-ABCQ approach obtains 125.25 s for makespan, 0.42 for the index of load balancing, 2.39 jobs/s for throughput, 367,825.9 W for energy consumption, 15.32 s for completion time, and 42.75% for reward with 2000 tasks. For 4000 tasks,

the DRL-ABCQ approach obtains 150.62 s for makespan, 0.43 for the index of load balancing, 2.87 jobs/s for throughput, 441,391.08 w for energy consumption, 18.38 s for completion time, and 42.75% for reward.

Table 1. Simulation parameters of the DRL-ABCQ approach

Parameters	Value
Scheduling interval	0.01 - 1 seconds
Worker node speed	200 - 400 instructions/second
Worker node memory size	2 - 8 GB
Worker node swapping cost	2 - 10 instructions/GB
Worker quantum	0.01 - 0.5 seconds
Worker node startup time	120 - 600 seconds
Worker node scheduler notification time	1-5 instructions
Simulation time	10,000
Task range	1,000 – 5,000
Length	1,000 – 2,000
File size	300 - 400

Table 2. Performance of different optimizations with the DRL-ABCQ approach for different tasks

Tasks	Methods	Makespan (s)	Index of load balancing	Throughput (jobs/s)	Energy consumption (watt)	Completion time (s)	Reward (%)
2,000	PSO	147.80	0.85	1.32	384217.8	32.23	38.50
	GWO	133.24	0.77	1.69	412309.67	38.17	31.65
	WOA	159.37	0.92	1.87	495623.74	29.54	29.60
	DMO	178.17	0.76	1.91	517890.43	33.54	39.53
	DRL – ABCQA	125.52	0.42	2.39	367825.9	15.32	42.75
4,000	PSO	177.36	0.86	1.58	461061.38	38.68	38.20
	GWO	160.00	0.78	2.03	494771.6	45.8	32.00
	WOA	191.24	0.93	2.24	594748.49	35.45	30.00
	DMO	213.8	0.77	2.29	621468.52	40.24	39.60
	DRL-ABCQA	150.62	0.43	2.87	441391.08	18.38	42.75

Table 3 presents the performance of the DRL-ABCQ approach evaluated with different numbers of VMs, specifically 10 and 50. The existing algorithms utilized to validate the performance of the DRL-ABCQ approach include PSO, GWO, WOA, and DMO. In Table 3, the metrics considered are makespan, index of load balancing, throughput, energy consumption, completion time, and reward. The DRL-ABCQ approach achieves 140 s for makespan, 0.45 for the index of load balancing, 2.50 jobs/s for throughput, 380,000 W for energy consumption, 20.00 s for completion time, and 45.00% for reward with 10 VMs. For 50 VMs, the DRL-ABCQ approach achieves 115.00 s for makespan, 0.40 for the index of load balancing, 3.00 jobs/s for throughput, 360,000 W for energy consumption, 22.00 s for completion time, and 47.00% for reward.

Table 3. Performance of DRL-ABCQ approach with different VMs

VMs	Methods	Makespan (s)	Index of load balancing	Throughput (jobs/s)	Energy consumption (watt)	Completion time (s)	Reward (%)
10	PSO	160.00	0.88	1.25	420.000	35.00	35.00
	GWO	150.00	0.80	1.55	430.000	40.00	32.00
	WOA	170.00	0.95	1.85	480.000	30.00	30.00
	DMO	185.00	0.78	1.95	500.000	33.00	38.00
	DRL – ABCQA	140.00	0.45	2.50	380.000	20.00	45.00
50	PSO	130.00	0.80	1.40	370.000	28.00	40.00
	GWO	120.00	0.75	1.75	380.000	32.00	35.00
	WOA	135.00	0.85	2.00	400.000	27.00	33.00
	DMO	145.00	0.74	2.05	410.000	30.00	40.00
	DRL – ABCQA	115.00	0.40	3.00	360.000	22.00	47.00

Table 4 presents the performance of the DRL-ABCQ approach evaluated with different task sizes, namely 2000 and 4000, using existing load balancing algorithms. The existing algorithms for comparison include RR, least connections (LC), global server load balancing (GSLB), and random forest (RF). In Table 4, the following metrics are presented: makespan, index of load balancing, throughput, energy consumption, completion time, and reward. The DRL-ABCQ approach achieves 125.25 s for makespan, 0.42 for the index of load balancing, 2.39 (jobs/s) for throughput, 367,825.9 W for energy consumption, 15.32 s for completion time, and 42.75% for reward with 2000 tasks. For 4000 tasks, the DRL-ABCQ approach achieves 150.62 s for makespan, 0.43 for the index of load balancing, 2.87 (jobs/s) for throughput, 441,391.08 W for energy consumption, 18.38 s for completion time, and 42.75% for reward.

Table 4. Performance of DRL-ABCQ approach with different tasks

Tasks	Methods	Makespan (s)	Index of load balancing	Throughput (jobs/s)	Energy consumption (watt)	Completion time (s)	Reward (%)
2,000	RR	182.37	0.68	1.92	192385.7	22.93	29.85
	LC	176.84	0.6	2.07	181472.3	21.65	31.20
	GSLB	171.92	0.57	2.18	176925.6	20.75	32.50
	RF	186.45	0.71	1.88	196734.9	23.27	28.75
	DRL-ABCQA	125.52	0.42	2.39	367825.9	15.32	42.75
4,000	RR	203.48	0.69	2.33	372548.9	27.63	28.40
	LC	198.76	0.62	2.47	362395.4	26.57	30.10
	GSLB	191.59	0.59	2.62	356785.1	25.42	31.50
	RF	207.32	0.73	2.24	382746.5	28.89	27.30
	DRL-ABCQA	150.62	0.43	2.87	441391.08	18.38	42.75

Figures 3 and 4 illustrate the reward obtained from the environment for reducing energy consumption and completion time. These figures compare the reward values of the DRL-ABCQ approach with existing algorithms such as PSO, GWO, WOA, and DMO for task sets of 2000 and 4000. Figure 3 shows the total reward per iteration, from 10 to 100 iterations, for the 2000-task set. Figure 4 shows the total reward per iteration, from 10 to 100 iterations, for the 4000-task set.

From both figures, the proposed DRL-ABCQ approach outperforms the existing algorithms. As the number of tasks increases from 2000 to 4000, reward percentage also surpasses the existing algorithms. Figure 5 represents accuracy of a DRL-ABCQ approach, clearly showing that the accuracy increases steadily as the iteration count progresses from 10 to 100. The final outcome of training the neural network is improved accuracy at the end of each iteration. This result reflects enhanced accuracy in dynamic environments. Figure 6 shows the performance of different algorithms with 2000 to 4000 tasks allocated to existing VMs. From Figure 6, it is evident that the DRL-ABCQ approach consumes less completion time than the existing algorithms, such as PSO, GWO, WOA, and DMO. Figure 7 represents the total energy consumption across various VMs in the data center. From Figure 7, it is clear that the DRL-ABCQ approach saves more energy than the existing algorithms, including PSO, GWO, WOA, and DMO. The number of VMs ranges from 10 to 50 to evaluate the performance of the DRL-ABCQ approach.

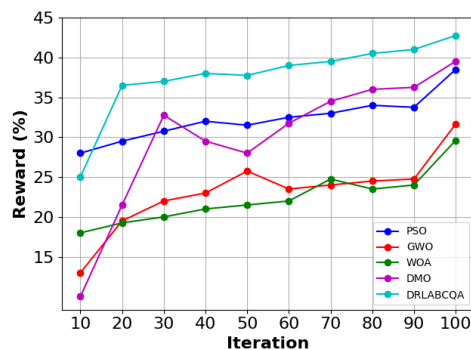


Figure 3. Reward percentage for 2,000 tasks

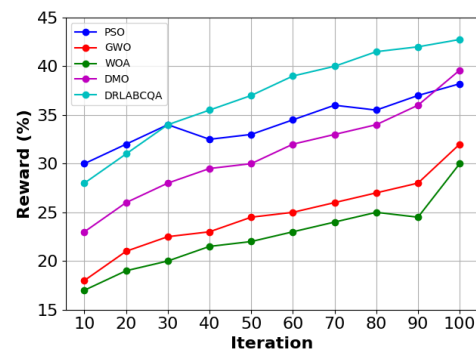


Figure 4. Reward percentage for 4,000 tasks

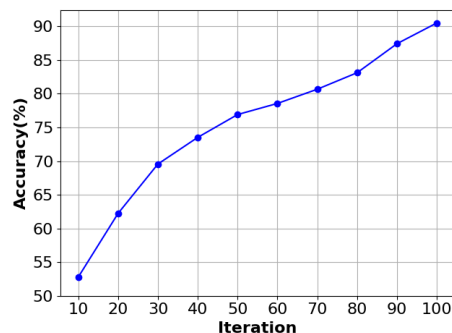


Figure 5. Iteration vs accuracy

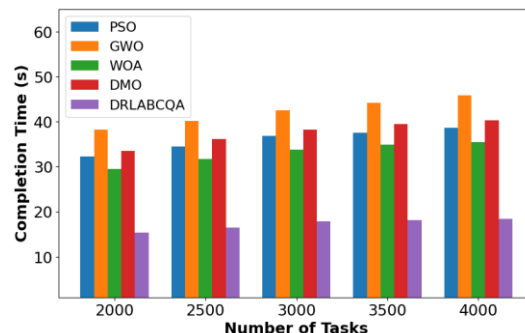


Figure 6. Number of tasks vs completion time

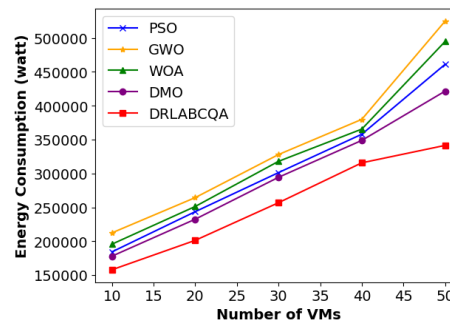


Figure 7. Number of VMs vs energy consumption

3.1. Comparative analysis

The performance of the DRL-ABCQ approach is compared with existing techniques like QMPSO, LBMPSTO, and MMHMO, using metrics like makespan, energy utilization, and throughput. The DRL-ABCQ approach achieves a makespan of 125.52 s, energy utilization of 99.81 kJ, and throughput of 2.39 jobs/s when compared to the existing approaches. Table 5 presents the comparative results of the DRL-ABCQ approach.

Table 5. Comparative table of the DRL-ABCQ approach

Methods	Makespan (s)	Energy utilization (kJ)	Throughput (jobs/s)
QMPSO [16]	NA	173.87	5.46
LBMPSTO [17]	108.87	NA	NA
MMHMO [18]	NA	55.01	4.52
Proposed DRL-ABCQ approach	125.52	99.81	2.39

3.2. Discussion

The simulation of the DRL-ABCQ approach is performed in a Python environment, and three different experiments are conducted to validate the reward percentage and accuracy. The performance of the DRL-ABCQ approach is compared with various optimization algorithms like PSO, GWO, WOA, and DMO. Additionally, DRL-ABCQ approach is compared with existing algorithms like QMPSO, LBMPSTO, and MMHMO. To address the issues of complexity and high dimensionality, the DRL-ABCQ approach is employed, which accelerates the process and minimizes the execution time. The agent in DRL makes intelligent decisions to handle huge number of tasks in less time and at higher speeds. The ABC algorithm utilizes the global guiding factor for initializing position updates, which enhances the searchability performance of the ABC algorithm. The proposed DRL-ABCQ approach optimizes scheduling and resource usage, increases VM throughput, and balances the load between VMs. The experimental outcomes confirm that DRL-ABCQ approach delivers the best rewards when large number of tasks are allocated to data center. The experimental results acquired by proposed DRL-ABCQ model are consistent which employ RL and hybrid optimization algorithms. Prior researches have shown that DRL-based schedulers efficiently adapt to dynamic workloads and minimize makespan and execution time through learning optimum resource allocation policies [20]–[22]. Similarly, hybrid meta-heuristic algorithms like QMPSO [16] and MMHMO [18] enhanced convergence behavior shows high energy consumption under heavy load conditions. The combination of a global guiding factor in ABC model in proposed DRL-ABCQ algorithm improves exploitation when avoiding premature convergence, resulting in low energy consumption and enhanced load balancing. Moreover, an increase in throughput with high VM counts determines the scalability and robustness of the proposed model in large-scale cloud environments.

4. CONCLUSION

This research proposes an effective approach for load balancing in a cloud environment. The DRL-ABCQ approach is proposed to balance load in cloud network. This approach combines both DRL algorithm and ABC with the Q-learning approach. By using DRL, the neural network is trained to achieve optimal rewards. Through the ABCQ approach, overall processing time of incoming load is minimized. The DRL-ABCQ approach improves various load balancing parameters in less time compared to existing algorithms in the cloud network. The DRL-ABCQ approach achieves a makespan of 125.52 s, energy utilization of 99.81 kJ, and throughput of 2.39 jobs/s when compared to existing approaches. The throughput of the proposed DRL-ABCQ approach is slightly lower than that of different benchmark algorithms, which is an essential trade-off for achieving improved energy efficiency. The intelligent scheduling mechanism in

DRL-ABCQ minimizes unnecessary task migrations, and optimized CPU utilization leads to an overall enhancement in energy efficiency. Future work focuses on optimizing the trade-off between throughput and energy consumption by including an adaptive learning mechanism that dynamically adjusts the scheduling parameters based on real-time workload variations.

ACKNOWLEDGMENTS

The authors would like to thank the School of Computer Science and Engineering, Presidency University, Bengaluru, for providing the necessary support and facilities to carry out this research work.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Mohammed Ziaur Rahman	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
Anandaraj Shanthi Pichandi		✓	✓	✓		✓		✓		✓	✓	✓	✓	✓

C : Conceptualization

M : Methodology

So : Software

Va : Validation

Fo : Formal analysis

I : Investigation

R : Resources

D : Data Curation

O : Writing - Original Draft

E : Writing - Review & Editing

Vi : Visualization

Su : Supervision

P : Project administration

Fu : Funding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not applicable to this paper as no new data were created or analyzed in this study.

REFERENCES

- [1] A. Jangra and N. Mangla, "An efficient load balancing framework for deploying resource scheduling in cloud based communication in healthcare," *Measurement: Sensors*, vol. 25, Feb. 2023, doi: 10.1016/j.measen.2022.100584.
- [2] P. Udayasankaran and S. J. J. Thangaraj, "Energy efficient resource utilization and load balancing in virtual machines using prediction algorithms," *International Journal of Cognitive Computing in Engineering*, vol. 4, pp. 127–134, Jun. 2023, doi: 10.1016/j.ijcce.2023.02.005.
- [3] D. Tennakoon, M. Chowdhury, and T. H. Luan, "Cloud-based load balancing using double Q-learning for improved quality of service," *Wireless Networks*, vol. 29, no. 3, pp. 1043–1050, Apr. 2023, doi: 10.1007/s11276-018-1888-8.
- [4] M. S. Al Reshan *et al.*, "A fast converging and globally optimized approach for load balancing in cloud computing," *IEEE Access*, vol. 11, pp. 11390–11404, 2023, doi: 10.1109/ACCESS.2023.3241279.
- [5] B. Xu and M. Lu, "Agent-based virtual machine migration for load balancing and co-resident attack in cloud computing," *Applied Sciences*, vol. 13, no. 6, Mar. 2023, doi: 10.3390/app13063703.
- [6] S. Sefati, M. Mousavinasab, and R. Z. Farkhady, "Load balancing in cloud computing environment using the grey wolf optimization algorithm based on the reliability: performance evaluation," *The Journal of Supercomputing*, vol. 78, no. 1, pp. 18–42, Jan. 2022, doi: 10.1007/s11227-021-03810-8.
- [7] K. J. Rajashekar, Channakrishnaraju, P. C. Gowda, and A. B. Jayachandra, "SCEHO-IPSO: a nature-inspired meta heuristic optimization for task-scheduling policy in cloud computing," *Applied Sciences*, vol. 13, no. 19, Sep. 2023, doi: 10.3390/app131910850.
- [8] W. Almuselem, "Energy-efficient and security-aware task offloading for multi-tier edge-cloud computing systems," *IEEE Access*, vol. 11, pp. 66428–66439, 2023, doi: 10.1109/ACCESS.2023.3290139.
- [9] M. Adil, S. Nabi, M. Aleem, V. G. Diaz, and J. C. Lin, "CA-MLBS: content-aware machine learning based load balancing scheduler in the cloud environment," *Expert Systems*, vol. 40, no. 4, May 2023, doi: 10.1111/exsy.13150.
- [10] H. Mikram, S. El Kafhali, and Y. Saadi, "HEPGA: a new effective hybrid algorithm for scientific workflow scheduling in cloud computing environment," *Simulation Modelling Practice and Theory*, vol. 130, Jan. 2024, doi: 10.1016/j.simpat.2023.102864.

- [11] J. P. Gabhane, S. Pathak, and N. M. Thakare, "A novel hybrid multi-resource load balancing approach using ant colony optimization with Tabu search for cloud computing," *Innovations in Systems and Software Engineering*, vol. 19, no. 1, pp. 81–90, Mar. 2023, doi: 10.1007/s11334-022-00508-9.
- [12] M.-L. Chiang, H.-C. Hsieh, Y.-H. Cheng, W.-L. Lin, and B.-H. Zeng, "Improvement of tasks scheduling algorithm based on load balancing candidate method under cloud computing environment," *Expert Systems with Applications*, vol. 212, Feb. 2023, doi: 10.1016/j.eswa.2022.118714.
- [13] S. Muruganandam, V. Natarajan, R. S. Peter, and R. Murugesan, "HHO-ACO hybridized load balancing technique in cloud computing," *International Journal of Information Technology*, vol. 15, no. 3, pp. 1357–1365, Mar. 2023, doi: 10.1007/s41870-023-01159-0.
- [14] K. Ramya and S. Ayothi, "Hybrid dingo and whale optimization algorithm-based optimal load balancing for cloud computing environment," *Transactions on Emerging Telecommunications Technologies*, vol. 34, no. 5, May 2023, doi: 10.1002/ett.4760.
- [15] A. A. Adewojo and J. M. Bass, "A novel weight-assignment load balancing algorithm for cloud applications," *SN Computer Science*, vol. 4, no. 3, Mar. 2023, doi: 10.1007/s42979-023-01702-7.
- [16] U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2332–2342, Jun. 2022, doi: 10.1016/j.jksuci.2020.01.012.
- [17] A. Pradhan and S. K. Bisoy, "A novel load balancing technique for cloud computing platform based on PSO," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 3988–3995, Jul. 2022, doi: 10.1016/j.jksuci.2020.10.016.
- [18] M. Haris and S. Zubair, "Mantaray modified multi-objective harris hawk optimization algorithm expedites optimal load balancing in cloud computing," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 10, pp. 9696–9709, Nov. 2022, doi: 10.1016/j.jksuci.2021.12.003.
- [19] S. Souravlas, S. D. Anastasiadou, N. Tantalaki, and S. Katsavounis, "A fair, dynamic load-balanced task distribution strategy for heterogeneous cloud platforms based on Markov process modeling," *IEEE Access*, vol. 10, pp. 26149–26162, 2022, doi: 10.1109/ACCESS.2022.3157435.
- [20] P. V. Lahande, P. R. Kaveri, J. R. Saini, K. Kotecha, and S. Alfarhood, "Reinforcement learning approach for optimizing cloud resource utilization with load balancing," *IEEE Access*, vol. 11, pp. 127567–127577, 2023, doi: 10.1109/ACCESS.2023.3329557.
- [21] A. Pradhan, S. K. Bisoy, S. Kautish, M. B. Jasser, and A. W. Mohamed, "Intelligent decision-making of load balancing using deep reinforcement learning and parallel PSO in cloud environment," *IEEE Access*, vol. 10, pp. 76939–76952, 2022, doi: 10.1109/ACCESS.2022.3192628.
- [22] A. R. Khan, "Dynamic load balancing in cloud computing: optimized RL-based clustering with multi-objective optimized task scheduling," *Processes*, vol. 12, no. 3, Mar. 2024, doi: 10.3390/pr12030519.
- [23] S. Singhal *et al.*, "Energy efficient load balancing algorithm for cloud computing using rock hyrax optimization," *IEEE Access*, vol. 12, pp. 48737–48749, 2024, doi: 10.1109/ACCESS.2024.3380159.
- [24] R. Rathinam, P. Sivakumar, S. Sigamani, and I. Kothandaraman, "SJFO: sail jelly fish optimization enabled VM migration with DRNN-based prediction for load balancing in cloud computing," *Network: Computation in Neural Systems*, vol. 35, no. 4, pp. 403–428, Oct. 2024, doi: 10.1080/0954898X.2024.2359609.
- [25] I. Ahmad, F. Qayum, S. U. Rahman, and G. Srivastava, "Using improved hybrid grey wolf algorithm based on artificial bee colony algorithm onlooker and scout bee operators for solving optimization problems," *International Journal of Computational Intelligence Systems*, vol. 17, no. 1, May 2024, doi: 10.1007/s44196-024-00497-6.

BIOGRAPHIES OF AUTHORS



Mohammed Ziaur Rahman    received B.E. (Computer Science and Engineering) from Visvesvaraya Technological University, Belgaum, Karnataka (distinction with honor); M.Tech. (Computer Science and Engineering) with 2nd rank from Visvesvaraya Technological University, Belgaum, Karnataka; and Ph.D. from Presidency University, Bengaluru, Karnataka. He is currently working as an assistant professor in the Department of Computer Science and Engineering, School of Engineering and Technology, Sapthagiri NPS University, Bengaluru. He has 19 years of teaching experience in various academic institutions. His areas of interest include cloud computing, big data, and Java full-stack development. He has published papers in various national and international conferences and journals. He also attended many national workshops, FDPs, and seminars. He is a member of IEEE, ISTE, CSI, and IAENG. He can be contacted at email: mohammedziaur@snpsu.edu.in or ziyarmn@gmail.com.



Anandaraj Shanthi Pichandi    received B.E. (Computer Science and Engineering) degree from Madras University, Chennai in the year 2004, M.Tech. (Computer Science and Engineering) with a gold medal from Dr. MGR Educational and Research Institute, University in the year 2007 (distinction with honor) and Ph.D. in August 2014. He is currently working as a professor and HoD-cybersecurity and internet of things under School of Computer Science and Engineering at Presidency University, Bengaluru. He has 20+ years of teaching experience. His areas of interest include information security, data science, machine learning, and networks. He wrote five book chapters in IGI Global Publications, Springer, and Elsevier. He filed 7 patents for the growth of research. He has published 50+ papers in various national and international journals and at national and international conferences. He serves as an editorial member for reputed journals. He also attended many national workshops, FDP, and seminars. He is a member of ISTE, CSI, IACSIT, and IAENG. He can be contacted at email: anandaraj@presidencyuniversity.in or anandsofttech@gmail.com.