7-14-2022

Intelligent Decision-Making of Load Balancing Using Deep Reinforcement Learning and Parallel PSO in Cloud Environment

Ali Wagdy Mohamed
The American University in Cairo (AUC), aliwagdy@aucegypt.edu

Arabinda Pradhan

Sukant Kishoro Bisoy

Sandeep Kautish

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Intelligent Decision-Making of Load Balancing using Deep Reinforcement Learning and Parallel PSO in Cloud Environment

Arabinda Pradhan[1], Sukant Kumar Bisoyi[2], Sandeep Kautish[3], Muhammed Basheer Jasser*[4] (Member, IEEE), Ali Wagdy Mohamed [5, 6]

[1,2] Department of Computer Science & Engineering, CV Raman Global University, Bhubaneswar, Odisha, India
[3] LBEF Campus Kathmandu, Kathmandu, Nepal
[4] Department of Computing and Information Systems, School of Engineering and Technology, Sunway University, Petaling Jaya 47500, Malaysia
[5] Operations Research Department, Faculty of Graduate Studies for Statistical Research, Cairo University, Giza 12613, Egypt
[6] Department of Mathematics and Actuarial Science School of Sciences Engineering, The American University in Cairo, Egypt

Corresponding author: Muhammed Basheer Jasser (basheerj@sunway.edu.my)

This work was supported by the Sunway University Publication Support Scheme.

ABSTRACT Machine learning and parallel processing are extremely commonly used to enhance computing power to induce knowledge from an outsized volume of data. To deal with the problem of complexity and high dimension, machine learning algorithms like Deep Reinforcement Learning (DRL) are used, while parallel processing algorithms like Parallel Particle Swarm Optimization (PPSO) are parallelized to speed up the operation and reduce the processing time to train the neural network. Due to the arrival of a large number of incoming tasks in the cloud environment, load balancing is an important issue. To solve this problem, the datacenter controller or an agent makes an intelligent decision to handle a large number of tasks within a minimum time period or at high speed. In this work, we proposed an effective scheduling algorithm named Deep Reinforcement Learning with Parallel Particle Swarm Optimization (DRLPPSO) to solve the load balancing problem and its various parameters with greater accuracy and high speed. Our experimental results show that our proposed scheduling algorithm increases the reward by 15.7%, 12%, and 13.1% when the task set is 2000 and improves the reward by 17.5%, 12.6%, and 15.3% when the task set is 4000, as compared to the Modified Particle Swarm Optimization (MPSO), Asynchronous Advantage Actor-Critic (A3C), and Deep Q-Network (DQN) techniques.

INDEX TERMS Load balancing, Deep Reinforcement Learning, Neural Network, Parallel PSO.

I. INTRODUCTION

Recently, cloud network has become a very popular technology that can offer different services as per user requests over the Internet. The development of distributed computing, parallel computing, and grid computing is widely used in the commercial sector where it offers various services in marketing, technology, and many other areas [1]. It is mainly based on the concept of on-demand delivery of computations, storage, applications, and other resources. However, the development of cloud computing is facing a number of challenges, including security and scheduling. Day-by-day, the number of user requests or loads increases, but the server or physical machine (PM) in the datacenter is limited, which leads to a task allocation problem in cloud networking. This problem can be handled with the concept of virtualization, where one physical machine is logically divided into a number of virtual machines (VMs) and tries to handle all incoming user requests [2, 3]. Due to the dynamic nature of cloud networks, loads fluctuate with respect to time for allocating a suitable VM. It shows a load balancing problem within the VM and has a direct impact on the physical machine. This problem will lead to high costs, minimize the profit of an organization and degrade the system performance. Each physical machine consumes

electric power for the task it does. That means if a server has a number of workloads, it consumes more energy [4, 5, 6]. To overcome this problem, a better scheduling algorithm is required that can handle the load among the VMs and execute all the incoming tasks with less execution time and consume less energy in the datacenter.

A legitimate scheduling technique can be utilized to enhance each parameter of load balancing for fulfilling Quality of Service (QoS) and further develop the framework execution. Various parameters such as makespan time, energy consumption, resource usage, cost, and so forth are considered to improve the performance of the cloud network. Basically, there are two important types of scheduling, such as task scheduling and resource scheduling, or VM scheduling. Task scheduling is responsible for optimizing the makespan time or execution time of all incoming tasks within a physical machine [7]. Resource scheduling is responsible for optimizing resource utilization, resource selection for a given task, and energy consumption [8]. In recent times, most researchers have focused on hybrid scheduling algorithms as well as parallel computing techniques to solve this optimization problem. A hybrid scheduling algorithm, for example, combines various meta-heuristic algorithms with machine learning techniques to build an effective scheduling algorithm [9, 10, 11]. A hybrid scheduling algorithm, which is a combination of Ant Colony Optimization (ACO) and Deep Reinforcement Learning (DRL) algorithms, has been proposed in [9] to increase the system performance. In [10], a method is proposed for avoiding the problem of the continuous nature of the cloud environment and improving the convergence rate by combining the policy gradient algorithm with particle swarm optimization-based parameter exploration (PG-PSOPE). To improve the exhibition, keep up load, and adjusting and incrementing the throughput, a hybrid technique is developed in [11], which combines both modified PSO and a Q-learning algorithm known as QMPSO. The Deep Q-network (DQN) method is broadly utilized in Deep Reinforcement Learning (DRL) to achieve the maximum reward [12, 13, 14]. In [12], a joint optimization is formulated of the task offloading and bandwidth allocation for multi-user mobile edge computing, with the objective of minimizing the overall cost, including the total energy consumption and the delay in finishing the task. To solve the resource allocation issue in the Mobile Edge Computing environment, a smart resource allocation algorithm, known as the Deep Reinforcement Learning based Resource Allocation (DRLRA) algorithm, is proposed in [13] to minimize average service time and balance the resource allocation. To achieve efficient real-time task scheduling in the cloud environment, a double deep Q-network task scheduling (DDQN-TS) scheduling method is proposed in [14] to reduce the task response time while ensuring a high task completion rate. Similarly, compared to various population-based methods, Particle Swarm Optimization (PSO) is more efficient, simple, and easy to learn with fewer parameters that require adjustment. It also gives higher performance than other population-based methods, but it shows convergence as well as local optimum problems [15,16]. Two different strategies are proposed in [15] to solve the traditional PSO problem. A dimensional learning strategy (DLS) is used for finding the personal best value of each particle. By applying the two-swarm learning PSO (TSLPSO) algorithm, it guides the local search of the particles and finds the optimal value from the global search. In [16], a load balancing method is proposed based on the Modified Particle Swarm Optimization (LBMPSO) method that uses the global best inertia weight parameter to avoid the local optimum problem. Commonly, PSO is to observe the ideal outcome of the population with the assistance of particles' individual value, fitness value, and global best value. Parallel computing is used to achieve the desired accuracy and is used to accelerate neural network training for large training data sets [17, 18, 19, 20]. In [17], a parallelization strategy for convolutional neural network (CNN) training is proposed based on two major techniques to maximize the overlap. In the first technique, all the gradients of parameters are divided into two large chunks that reduce the communication time. To reduce communication costs, the second technique involves replicating the gradient calculation in a few fully-connected layers. In [18], a concept of parallel processing is proposed that helps in saving time in artificial neural networks (ANNs) training. In [19], a parallel algorithm called Split and Conquer for solving Verification of Neural Network (VNN) formulas, is proposed using the Reluplex procedure and an iterative-deepening strategy. In [20], a parallel deep neural network architecture with an embedded organization mechanism is proposed, which enforces diversity among the deep neural networks used as base models. The Parallel Particle Swarm Optimization (PPSO) algorithm is an example of a parallel computing process to minimize the processing time of high computation [21, 22, 23]. In [21], a novel way to implement PSO on a Graphic Processing Unit (GPU) is proposed, where the PSO algorithm can be executed in parallel on the GPU to optimize the system performance and speed. In [22], a technique is proposed to solve the highway alignment optimization problem by using an integrated model of parallel processing and a particle swarm optimization algorithm. This method is used to optimize the speed of processing. In [23], two parallel PSO algorithms on the hierarchical GPU are developed. These algorithms are applied to Max-CSPs to improve the GPU's resolution effectiveness and reduce execution time. The first method is a parallel GPU-PSO for Max-CSPs (GPU-PSO), and the second one is a GPU distributed PSO for Max-CSPs (GPU-DPSO). With the help of parallel techniques, the large training dataset can be divided into a number of subparts that speed up the processing [24]. By applying the back propagation learning algorithm, it adjusts the weight and achieves better results. Authors in [25] introduced a unique technique that uses variable lengths of particles to search for optimal topologies in deep convolutional neural networks.

As we know, in a cloud environment, incoming tasks are placed into an appropriate virtual machine to allocate the resources for execution. However, when the appropriate resources are not available on the server or it is heavily loaded, these incoming tasks take too much time to complete. If a server is loaded, then it consumes more energy. To overcome this problem, a parallel scheduling method is proposed in this paper which depends on both task and resource scheduling and is named Deep Reinforcement Learning with Parallel Particle Swarm Optimization (DRLPPSO). In this work, we use both DRL and Parallel PSO techniques to optimise the solutions.

In a cloud environment, an appropriate decision can be made for allocating a huge number of incoming tasks to suitable resources. This decision can be made with the help of the DRL algorithm. We consider systems that learn to manage resources directly from experience and train our model by using artificial neural networks. Using deep neural networks, it offers extraordinary capacity to deal with complex control issues in a highly layered and continuous environment [26]. In a cloud environment, task allocation and energy consumption problems are figured out as Markov Decision Processes (MDP). Multiple replay memory is utilized in the DQN method to reduce execution time, allocation time, task transfer time, and energy consumption [27, 28]. According to [29], energy consumption in datacenters has two distinct characteristics: (i) servers use more energy when they are heavily loaded; and (ii) servers use a lot of power when they are idle. Subsequently, server solidification and load balancing can be used to increase the overall system reward rate and accuracy that allows users to receive more benefits. In our proposed algorithm, an agent, such as a datacenter controller, checks the status of a VM to determine whether it is overloaded or underloaded. If the VM is overloaded, then it takes the necessary action to migrate some tasks from the overloaded VMs to the underloaded ones. Also, it helps to reduce the variance between the targeted load and the present load of VMs within a single server.

Due to its promising results, Parallel PSO (PPSO) was chosen as the metaheuristic optimization technique in this paper. It enhances the system performance due to parallelization and improved speed of large-scale analytical test problems [30, 31, 32]. In [30], a coarse decomposition scheme is chosen where the algorithm performs only the fitness evaluations concurrently on a parallel machine. To find the global best result by applying Parallel PSO, a model is proposed in [31] which is based on a master-slave model. With the help of the PPSO algorithm [32], the VMs are selected to minimize the total execution time. To reduce the search time and improve fitness function, PPSO divides the swarm into sub-swarms [33, 34, 35], and each sub-swarm contains a DRL algorithm to find the reward. These sub-swarms are run in parallel to minimize processing time. As a result, it is appropriate to handle several requests from various users in parallel. In [33], a parallel swarm-oriented particle swarm optimization (PSO-PSO) with multi-stage and a single stage of development is suggested. Individual subswarms evolve independently in parallel in multi-stage evolution, but in single-stage evolution, subswarms exchange information to find the global-best. The two intertwined stages of evolution show superior performance on test functions, particularly those with higher dimensions. The PSO-PSO version of the technique is appealing because it does not incorporate any new parameters to boost convergence performance. In [34], a parallel particle swarm optimization algorithm is proposed, which comprises two phases for solving the convergence and local optimum problems. The first is the multi-evolutionary phase, in which the swarm is divided into k sub-swarms. Each sub-swarm evolves independently. Each particle adjusts its position depending on its own experience as well as the swarm's best. For a specific number of iterations, this process is repeated. The swarms are then combined to form a single evolutionary phase. In which the swarm best of each subswarm is compared to determine the global best. In [35], PPSO is employed to find the ideal virtual machine selection in a cloud environment to lower the cost of service based on turnaround time, waiting time, and CPU utilization. The primary contribution of this paper is as per the following:

- We proposed a hybrid method that combines PPSO and DRL techniques, which aims to maximize the reward by reducing makespan time and energy consumption while maintaining high accuracy, as well as speedup the execution of continuous approaching position in a cloud environment.
- We coordinate the numerical model of our method and also depict the point-by-point execution of our DRLPPSO algorithm.
- We evaluate the proposed method by taking different task sets. The experimental outcomes show the sufficiency of the proposed method in relation to current pattern strategies.

The remainder of the paper is spread out as follows: Related work is discussed in Section II; the cloud framework model and optimization objectives are discussed in Section III; Section IV presents the DRLPPSO scheduling algorithm; experimental results are discussed in Section V. The paper is concluded in section VI.

II. RELATED WORK

There have previously been a large variety of scheduling policies that are utilized on static, dynamic, and hybrid policies that are centered on either a single objective, bi-objective, or multi-objective. These policies have focused their research on maintaining machine load balance as well as various load balancing factors such as makespan time, energy consumption, resource utilization, and so on. To address the aforementioned problem, researchers developed

several machine learning and population-based optimization scheduling algorithms for the cloud environment. To improve the service quality and decrease the cost, [36] suggested a technique that relies on the Deep Q-network (DQN) algorithm to reduce energy consumption and makespan time by modifying the reward weight. [37] suggested a two-stage methodology for job scheduling and resource allocation. A heterogeneous distributed deep learning (HDDL) method is employed to manage job scheduling, and a deep Q-network (DQN) is used to address resource allocation. Each algorithm is utilized to reduce the amount of energy required and the time it takes to complete a task. In [38], a task scheduling technique based on a deep reinforcement learning algorithm was suggested to maximize makespan and resource consumption. In [39], a task scheduling method based on deep reinforcement learning architecture (RLTS) was presented to manage the complexity and high dimensionality of the environment by minimizing task execution time. [40] presented foresighted job scheduling based on Q-learning to reduce reaction time and makespan while increasing resource effectiveness. To address the job scheduling problem, [41] proposed a deep reinforcement learning based algorithm to help application providers dispatch jobs to limited resources under QoS requirement constraints. In [42], the DQN method is proposed, which follows the DRL approach to reduce energy consumption and average response time while increasing the success rate. In [43], a modified PSO task scheduling algorithm (MPSO) is proposed, which is utilized to minimize transmission, execution, and energy consumption in a cloud environment by using a modified inertia weight method. To solve the resource allocation problem in cloud datacenters, [44] proposed a method which is based on Actor-Critic Deep Reinforcement Learning. In this method, the author tries to reduce the energy consumption and improve the QoS.

All the above methods have two common objectives, i.e., makespan time and energy consumption. To achieve the goal, there is no suitable action defined to handle the extra task that can increase the reward in a dynamic environment. Therefore, in this paper, we define a suitable action that can handle the extra task and give the maximum reward as compared to the above methods. Also, in this paper, we show that our proposed method has better accuracy and faster processing speed as compared to others. Comparison between various scheduling algorithms with their advantages and disadvantages is represented in Table I.

TABLE I
ADVANTAGES AND DISADVANTAGES OF EACH SCHEDULING ALGORITHM

Ref. No.	Techniques	Advantages	Disadvantages
[36]	Combined a deep convolutional neural network with the traditional Q-learning algorithm	Reduce makespan time and energy consumption	Performance degrades
[37]	Used two different scheduling techniques	Reduce energy consumption and job delay	Less efficient
[38]	Advantage Actor-Critic (A2C)	Optimize makespan and resource utilization	Not consider more task attributes
[39]	DQN is used for training the neural network	Reduce makespan time	Single objective
[40]	Threshold-based approach based on self-adaptive and self-learning capabilities	Decrease makespan but increase resource utilization	Space complexity
[41]	QoS-aware job scheduling framework	Reduce response time and increase VM utilization	Resources are not fully utilized
[42]	Job scheduling generic algorithm	Decrease response time and energy consumption	Jobs are dependent to each other
[43]	Modified inertia weight	Reduce execution and transmission cost	Mismatch between local and global search.
[44]	Asynchronous Advantage Actor-Critic (A3C)	Reduce energy consumption and increase resource utilization.	Task migration is time taking

III. CLOUD FRAMEWORK MODEL AND OPTIMIZATION OBJECTIVES

A basic framework of a cloud network is shown in Fig. 1, which contains two layers: a task layer and a datacenter layer. These two layers store the required information to achieve the objective. In the task layer, all needed information about incoming tasks, such as expected completion time (ECT), task length (), and task file size () is stored in the task queue to find the best VMs from the server. Similarly, the datacenter layer holds all of the necessary information such as storage units, processing units, data transfer capacity, and processing speed for both servers and virtual machines. This information is helpful to the datacenter controller, where it applies the best scheduling algorithm to accomplish the objective and get the optimum result. Our objectives include reducing task completion time and energy consumption. Based on each objective, we found the reward function that helps us achieve the best optimization of the system. Table II. shows the terms and meanings used in the proposed algorithm.

TABLE II
TERMS AND MEANING

Terms	Meaning	Terms	Meaning
VM	Virtual machine	VM_{tld}	Total load of VM

T_w	Number of incoming tasks	VM_{avgld}	Average load of VM
VM_v	Number of VMs	TH_{value}	Threshold value
S_s	Number of servers	b_{avg}	Average bound
VM_{prr}	Processing rate of VM	E_{cons}	Energy consumption
VM_{mips}	Millions of instructions per second of VM	E_{active}	Energy consumption of an active VM
VM_{cpu}	Number of central processing unit	E_{texe}	Energy consumption of task execution
VM_{mem}	Memory of VM	E_{tt}	Energy consumption of task transfer between two VMs
$ECT_{w,v}$	Expected completion time of task w on VM v	CPU_{util}	CPU utilization
T_{leng}	Task length	S_{wt}	Weight of the server
$TA_{w,v}$	Task allocation time of task w on VM v	x and y	Two VM
T_{fs}	Task file size	$TT_{x,y}$	Task transfer between two VM, x and y
VM_{bw}	Bandwidth of VM	$BW_{x,y}$	Bandwidth of two VM
$CT_{w,v}$	Total completion time	E_{idle}	Energy consumption of idle VM
$DV_{w,v}$	Decision variable	T_{ts}	Task transfer speed
VM_{capa}	VM capacity	$ETTR_{x,y}$	Extra task transfer between VM x to VM y
VM_{ld}	VM load		

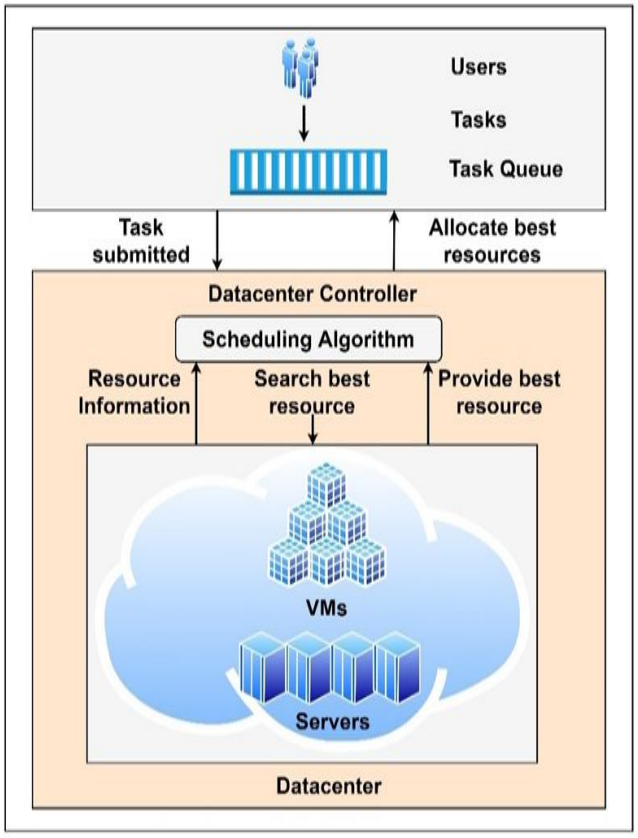

FIGURE 1. **Cloud framework.**

A. TASK COMPLETING TIME

Suppose a datacenter in the cloud network consists of a set of s servers with various resource configurations, such as $S_s, where\ s = \{1,2,\dots,m\}$. Number of incoming tasks w, which are entering into datacenter, such as $T_w, where\ w = \{1,2,\dots,n\}$ and each server contains v number of VMs, such as $VM_v, where\ v = \{1,2,\dots,o\}$, which are the basic units of server resources, but the condition for execution of such tasks is, $w > v$. Each task has a length T_{leng} which is expressed as millions of instructions (MI) and the speed of VM processing is measured in millions of instructions per second (MIPS). To accomplish our objective, we figure the processing rate of VM as in (1). It depends on the properties of VM, such as MIPS, CPU and Memory.

$$\sum_{v=1}^{o} VM_{prr} = \sum_{v=1}^{o}(VM_{mips} + VM_{cpu} + VM_{mem}) \quad (1)$$

The expected completion time (ECT) of task w on virtual machine v can be addressed as in (2).

$$ECT_{w,v} = \frac{\sum_{w=1}^{n} T_{leng}}{\sum_{v=1}^{o} VM_{prr}} \quad (2)$$

When task w is assigned to v then it takes some time, which is referred to as task allocation time ($TA_{w,v}$) and it is determined by the task file size with respect to bandwidth of VM (VM_{bw}) as represented in (3).

$$TA_{w,v} = T_{fs}/VM_{bw} \quad (3)$$

The total completion time ($CT_{w,v}$) of all entering tasks to be executed on VMs is calculated as the addition of expected completion time ($ECT_{w,v}$) and task allocation time ($TA_{w,v}$), as represented in (4) where, $DV_{w,v}$ is the decision variable represented in (5).

$$CT_{w,v} = ECT_{w,v} + TA_{w,v} + DV_{w,v} \quad (4)$$

$$DV_{w,v} = \begin{cases} 1, & if\ T_w\ is\ allocated\ to\ VM_v \\ 0, & Otherwise \end{cases} \quad (5)$$

B. LOAD AND CAPACITY OF VM

In cloud computing, all the incoming tasks are allocated to suitable VMs for execution within a short time period. The controller chooses the best VM to execute the incoming task, which depends upon the task length, VM capacity, and previous load of the VM. The processing speed, memory, CPU, and bandwidth of a VM define its capacity in (6). The load of VM is determined by the total number of tasks and task length that VM already holds with respect to its capacity. The VM load is calculated in (7). The total load and average load of all VMs on a server are represented in (8) and (9), respectively.

$$VM_{capa} = VM_{prr} + VM_{bw} \quad (6)$$

$$VM_{ld} = \frac{T_w \times T_{leng} \times T_{fs}}{VM_{capa}} \quad (7)$$

$$VM_{tld} = \sum_{v=1}^{o} VM_{ld} \quad (8)$$

$$VM_{avgld} = \frac{VM_{tld}}{VM_v} \quad (9)$$

C. VM STATUS

After allocating the entire task to the VM, the controller checks the status of each VM. Each VM can have three states: overload, underload, and balance. If the load of a VM exceeds the threshold value, then the VM status is overload; if the load is below the threshold value, then its status is underload; otherwise, the VM status is balance. The VM status is represented in (10) and threshold value is represented as in (11), where b_{avg} is the average bound, $1<b_{avg}<2$.

$$TH_{value} = (VM_{avgld} \times b_{avg}) - VM_{ld} \quad (10)$$

$$VM_{ld} = \begin{cases} < |TH_{value}| \ \ Underload \\ > |TH_{value}| \ \ Overload \\ = |TH_{value}| \ \ Balanced \end{cases} \quad (11)$$

D. ENERGY CONSUMPTION

The energy consumption (E_{cons}) of the server S_s at time t depends on the number of active and idle VMs. Both active and idle VMs depend on CPU utilization. The energy consumption of an active VM (E_{active}) is calculated as the energy consumption of task execution at a particular VM (E_{texe}) and energy consumption of task transfer between two VMs (E_{tt}), which is represented in (14). E_{texe} depends on the load on VM, CPU utilization of VM and weight of the server (S_{wt}) which is represented in (12). Energy consumption of task transfer between two VM (E_{tt}) depends on task transfer between two VM, their bandwidth and CPU utilization. It is calculated as in (13), where x *and* y are the two VM. According to [45], an idle machine can consume two third energy of CPU utilization, which is represented in (15).

$$E_{texe} = VM_{ld} \times CPU_{util} \times S_{wt} \quad (12)$$

$$E_{tt} = \left(\frac{TT_{x,y}}{BW_{x,y}}\right) \times CPU_{util} \quad (13)$$

$$E_{active} = E_{texe} + E_{tt} \quad (14)$$

$$E_{idle} = \frac{2}{3} \times CPU_{util} \quad (15)$$

Finally, energy consumption on a server is the sum of the energy consumption of all active and idle VMs, which is represented in (16).

$$E_{cons} = \sum_{v=1}^{o} (E_{active} + E_{idle}) \quad (16)$$

IV. DRLPPSO SCHEDULING ALGORITHM

In this section, we describe the essential idea of the proposed DRLPPSO scheduling algorithm, where an agent can get an individual reward by taking an appropriate action on each state in an environment by using the DRL algorithm. By using the PPSO algorithm, after getting their individual best reward, they would try to get the global best reward with the minimum processing time by exchanging their information with the neighbors. We first select the suitable action by taking a state-action value function and then get the personal best reward for each server that optimizes the load and its parameters. After getting the personal best reward, each server can find the global best value by sharing its information among other servers at high speed. This section contains various sub-sections to describe the reward function, DRL, PSO, and PPSO.

A. REWARD FUNCTION

The load balancing problem is described as a Markov Decision Process (MDP) due to the continuous nature of tasks in cloud computing. In our proposed method, the datacenter controller is represented as an agent and the datacenter is represented as the environment where the agent takes action by allocating incoming tasks to a suitable VM in each cycle. MDP has four important variables that are described as below.

1) STATE SPACE

Inside this space, the ideal move is made by an agent based on the current VM information such as the number of VMs, their available MIPS, CPU, memory, and bandwidth. Our state space can be characterized as:

$$S = \{VM_v, VM_{mips}, VM_{cpu}, VM_{mem}, VM_{bw}\}$$

2) ACTION SPACE

In this action space, each task is allocated a VM for execution. Each action has various pieces of information, such as: the number of tasks, task length, and file size. Our action space can be addressed as:

$$A = \{T_w, T_{leng}, T_{fs}\}$$

3) TRANSITION FUNCTION AND ACTION SELECTION

It shows that when an agent takes an action in a current state, it reaches a new state. Each time an agent tries to take an appropriate action to reach an optimal state that gives the highest reward. Transition function is represented as $P(s_t, a_t, r_t, s_{t+1})$. It is the probability of reaching the next state s_{t+1} and getting reward r_t after executing selected action a_t at the current state s_t. Due to the dynamic nature of cloud networks, tasks vary with respect to time, length, and

file size. Therefore, the load of the VM also changes. Some of the VMs are overloaded and some of them are underloaded, which is determined by (11). To balance the load among VMs, the agent takes the action that transfers the extra task from an overloaded to an underloaded VM at a high speed. Task transfer speed can be calculated by using (17). If we increase the task transfer speed, then it minimizes the completion time of tasks. The selected action a at iteration t, i.e., can be calculated by using (18).

$$T_{ts} = \sum_{w=1}^{n} \sum_{x=1}^{o} \sum_{y=1}^{o} (1 - DV_{w,x} DV_{w,y}) \frac{ETTR_{x,y}}{BW_{x,y}} \quad (17)$$

where, $ETTR_{x,y}$ is the extra task transfer between VM x to VM y. $BW_{x,y}$ is the required bandwidth between two VM x and VM y. $DV_{w,x} = 1$, indicates that the VM x is overloaded. $DV_{w,y} = 1$, indicates, extra task has been transferred from VM x to VM y.

$$a_t = \max_{a}(T_{ts}) \quad (18)$$

4) REWARD

An agent will get a reward for making specific moves in various states. An agent is attempting to choose a state having a higher reward to maximize its accuracy. In our model, the reward function is characterized by the minimization of the completion time of the task on a specific VM and energy consumption. The reward is represented in (19).

$$r_t = \min\{CT_{w,v} + E_{cons}\} \quad (19)$$

B. DEEP REINFORCEMENT LEARNING

Recently, a number of machine learning algorithms, such as RL and DRL algorithms, have been applied to computing platforms to optimize the respective parameters. These learning algorithms acquire knowledge from the environment by choosing actions. It is helpful for maximizing reward by optimizing the factors in the environment. RL algorithm is a model-free Q-learning algorithm that uses a state-action value function $Q^{\Pi}(s_t, a_t)$ to represent a value for selecting an action a_t in a current state s_t that follows a policy Π. This function is stored in Q-table or gets the reward. The state-action value function follows the Bellman condition [46,47] and it is represented in (20).

$$Q^{\Pi}(s_t, a_t) = r_t + \gamma \max Q^{\Pi}(s_{t+1}, a_{t+1}) \quad (20)$$

To increase the algorithm's performance, the learning rate β is added, as represented in (21).

$$Q^{\Pi}(s_t, a_t) = Q^{\Pi}(s_t, a_t) + \beta((r_t + \gamma \max Q^{\Pi}(s_{t+1}, a_{t+1})) - Q^{\Pi}(s_t, a_t)) \quad (21)$$

This procedure will be carried out iteratively till the terminal condition is reached. Each time, these Q-value or state-action value functions are stored in the Q-table. This shows the drawback of the Q-learning algorithm. As the quantity of actions grows, the intricacy of computation also grows, hence it diminishes the exhibition of the system. Thus, Q-learning algorithm in RL will not show data efficiency, learning efficiency, and stability. To overcome this challenge, we use DQN, an altered rendition of typical Q-learning that employs experience replay, target networks, exploration, and exploitation techniques. This strategy enables our proposed algorithm to be more suitable for training large neural networks with faster convergence speeds, as demonstrated in Fig. 2. Two neural networks are shown in Fig. 2, which are target Q-network and evaluated Q-network, and they have the same network structure. Both are used in the training process for choosing N experiences from the experience memory reply D. We set θ as the parameter of evaluated Q-network and θ' is the parameter of target Q-network. At each iteration t, state s_t, action a_t and parameter θ are used to generate state-action value function $Q(s_t, a_t; \theta)$ with reward. This serves as an input to the target Q-network to acquire the highest state-action value function of all actions in the target Q-network. In this paper, we use the $\in$-greedy strategy to select the random action otherwise we choose the action by using (18). After getting the reward, we compare the state-action value function of evaluated Q-network with the state-action value function of target Q-network to get more accuracy.

The goal of the proposed algorithm is to get the evaluated Q-network as near to the target Q-network as possible to achieve better outcomes. As a result, we train our neural network to calculate the loss function (which decreases the difference between the evaluated and target Q-networks), and the neural network's parameters are updated using backpropagation and gradient descent [48]. The loss function can be reduced to update the parameters of Q-networks. Equation (22) can be used to construct the loss function and (23) can be utilized to establish the target state-action value function.

$$L(\theta) = E[(target_t - Q(s_t, a_t; \theta))^2] \quad (22)$$

$$target_t = r_t + \gamma \max_{a_{t+1}} Q'(s_{t+1}, a_{t+1}; \theta') \quad (23)$$

where $target_t$ is the target state-action value function and it is evaluated by the action performed on the target Q-network with parameters θ'. All the parameters of the evaluated Q-network θ are updated at each iteration, t. However, the parameter of target Q-network θ' is fixed and updated only at stationary stages. As a result, the target Q-network update rate is slower than the evaluated Q- network. The pseudo code of the DRL algorithm is described in Algorithm 1. The datacenter controller uses this algorithm to gather the vital network parameters, for example, individual load, overall load, and completion time, which are refreshed

as the states of the environment change. Following the initial transition function (s_t, a_t, r_t, s_{t+1}), network parameters are utilized to develop the loss function and then fine-tuned. Each time the load is assigned, an action a_t is chosen using (18), followed by the reward depending on choosing the state s_{t+1}. This decision is forwarded to the controller, who is in charge of allocating resources for subsequent tasks.

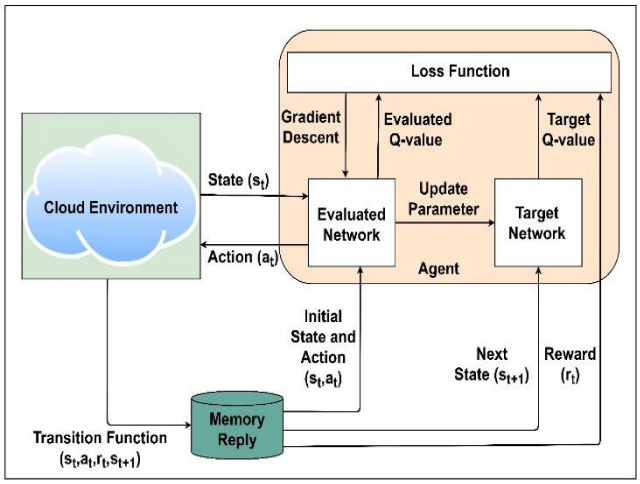

FIGURE 2. **Structure of neural network.**

Algorithm 1 of DRL

Input: Learning rate; discount factor; exploration factor; replay memory capacity; information of each task, VM and server;
Output: Reward of each server.

1. Initialize memory D to capacity N
2. Initialize evaluated Q-network parameters Θ
3. Initialize target Q-network with parameters θ' where $\theta' = \Theta$
4. **Begin**
5. **For** each episode e **do**
6. initialize the state s_t with load
7. **For** each task in task-queue **do**
8. **if** probability ∈ **then**
   choose a random action a_t
   else
   Select $a_t = \max_a(T_{ts})$
   End if
9. Applying action a_t, calculate total reward r_t by using (19)
10. Move to the new state s_{t+1}
11. Store transition (s_t, a_t, r_t, s_{t+1}) in memory D
12. Execute Algorithm 1.1 for evaluated Q-network training
13. Every $\mathcal{T}$ step, update target Q-network $\theta' = \Theta$
14. **End For**
15. **End For**
16. Return reward
17. **End**

Algorithm 1.1 Evaluated Q-network Training

1. Sample random mini-batch of transition (s_u, a_u, r_u, s_{u+1}) from memory D
2. **if** episode terminates at step j+1 **then**
   Set $target_v = r_v$
   else
   Set $target_u = r_u + \gamma \max_{a_{u+1}} Q'(s_{u+1}, a_{u+1}; \theta')$
3. **End if**
4. Perform a gradient descent using loss function: $(target_u - Q(s_u, a_u; \theta))^2$
5. Repeat till least loss value is achieved with update parameter

C. PARTICLE SWARM OPTIMIZATION

In particle swarm optimization (PSO), each VM in a server is represented as a particle. The total load of a VM on a server is referred to as its position, and task transfer from one VM to another is referred to as velocity. We take the reward function as the fitness function of each server. To discover the best fitness as a personal best, we compare the fitness function to each particle's personal best (such as current load, status, and CPU utilization). Finally, from all the personal bests, we find the global best and allocate the next incoming task to the best particle. After finding, each particle can update its position and velocity according to (24) and (25). Each term used in PSO with its meaning is explained in Table III.

TABLE III
TERMS AND MEANING

Terms	Meaning	Terms	Meaning
i	Particle	t	Iteration
X	Position	V	Velocity
$X_i(t)$	Current position	$X_i(t+1)$	Modified position
$V_i(t)$	Current velocity	$V_i(t+1)$	Modified velocity
P_i	Personal best	G_{best}	Global best
$c_1 and\ c_2$	Co-efficient	$r_1\ and\ r_2$	Random number
ω	Weight factor		

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{24}$$

$$\mathrm{V_i(t+1)} = \omega \mathrm{V_i(t)} + \mathrm{c_1 r_1}\left(\mathrm{P_i(t)\text{-}X_i(t)}\right) + \mathrm{c_2 r_2}\left(\mathrm{G_{best}(t)\text{-}X_i(t)}\right) \tag{25}$$

D. PARALLEL PARTICLE SWARM OPTIMIZATION

The PPSO algorithm is a suitable method for solving an optimization problem with less processing time. The PPSO algorithm leads to an enhanced throughput due to parallelization and improved speed, even if the environment contains a large population size. The working principle of PPSO is same as PSO, but the difference is that in PPSO, the

main swarm is divided into a number of sub-swarms where each sub-swarm works as a single PSO and each sub-swarm runs in parallel to reduce the processing time for getting the best result.

The PPSO algorithm has two phases. One is a multi-evolutionary phase and the other is a single evolutionary phase. The swarm or datacenter is randomly divided into k sub-swarms or servers during the multi-evolutionary phase. Each sub-swarm consists of a number of particles or virtual machines (VMs), each of which can be evaluated independently of the swarm. After determining the optimal value, each particle uses (24) and (25) to update its position and velocity based on its own and the swarm's best experiences. For a specific number of iterations, this process is repeated. The sub-swarms are then combined to form a single evolutionary phase. The global best (G_{best}) is determined by comparing the swarm best (S_{best}) of each sub swarm. For minimization problems, (26) represents the G_{best} in a given iteration.

$$G_{best} = \min\{S_{best\,1}, S_{best\,2}, \dots, S_{best\,k}\} \tag{26}$$

Following the discovery of the global best, each particle is updated depending on its personal best, swarm best, and global best. Individual swarms begin communicating by referencing the global best. Equations (27) and (28) are used to update the velocity and position of all particles.

$$V_{ij}(t+1) = \omega V_{ij}(t) + c_1 r_1 \left(P_{ij}(t) - X_{ij}(t)\right) + c_2 r_2 \left(P_{sj}(t) - X_{ij}(t)\right) + c_3 r_3 \left(G_{best}(t) - X_{ij}(t)\right) \tag{27}$$

$$X_{ij}(t+1) = X_{ij}(t) + V_{ij}(t+1) \tag{28}$$

For a specific number of iterations, this process is repeated. In comparison to the original PSO, the parallel version of the method is simple to implement and delivers better assignments. Table IV. represents the terms and meanings used in PPSO.

TABLE IV
TERMS AND MEANINGS

Terms	Meaning	Terms	Meaning
X_{ij}	Position	V_{ij}	Velocity
P_{ij}	Personal best	P_{sj}	Swarm best
i	Particle	j	Swarm
ω	Weight factor	G_{best}	Global best
c_1, c_2 and c_3	Co-efficient	r_1, r_2 and r_3	Random number
t	Iteration		

E. FLOWCHART AND ALGORITHM OF DRLPPSO

The flowchart of our proposed model is represented in Fig. 3, and the pseudo code of the DRLPPSO algorithm is described in Algorithm 2.

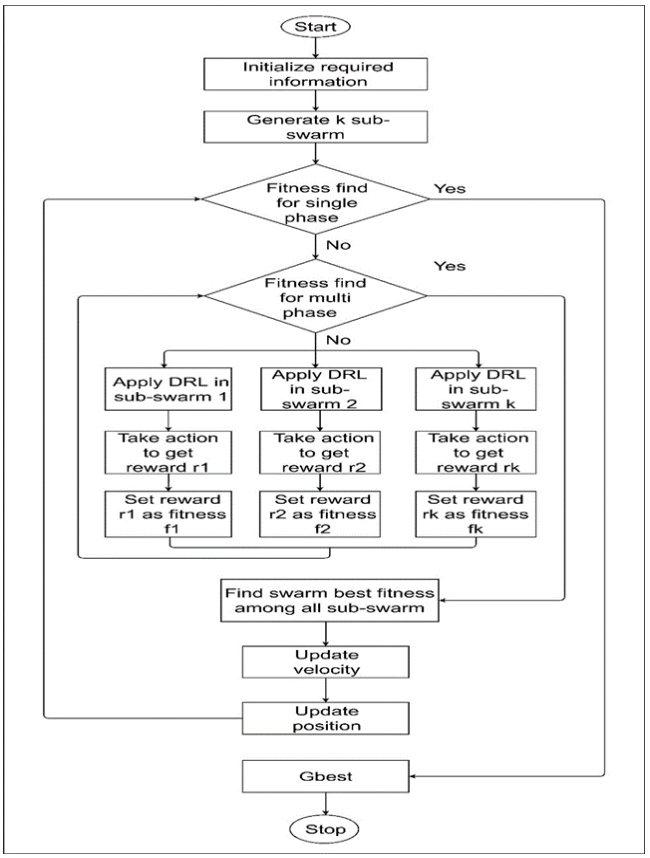

FIGURE 3. **Flowchart of DRLPPSO.**

V. EXPERIMENTAL SET UP

This section evaluates our proposed DRLPPSO scheduling algorithm and compares it with three existing algorithms, such as MPSO, A3C, and DQN algorithm. From the simulation results, it is clearly shown that our model gives better rewards as compared to existing algorithms. It also handles the machine load and minimizes load balancing parameters. The results also show the high accuracy and the speed increase of our system performance. Overall tests were conducted in Google Colab with the Python environment and TensorFlow. In this environment, we use the PPSO technique for training the neural network to optimize its load. All the simulation results are shown in Fig. 4 to 9.

A. PARAMETERS USED FOR SIMULATION

In this section, we provide all the necessary information for the simulation. For the test, we want to get maximum rewards from the environment where we perform continuous and independent tasks. These tasks are distributed between a number of VMs, and these VMs are assigned to a number of servers in a datacenter. Table V displays task properties, Table VI displays server and VM properties, and Table VII displays various PPSO and DRL approach properties.

Algorithm 2 DRLPPSO

Input: Server set as $S_s = \{S_1, S_2, \dots, S_m\}$, VM set as $VM_v = \{VM_1, VM_2, \dots, VM_o\}$ and Task set as $T_w = \{T_1, T_2, \dots, T_n\}$
Output: Minimize Task Completing Time and Energy consumption

1. **Begin**
2. Initialize ω, c_1, c_2, k sub-swarm size, iterations, r_1, r_2 and particle personal best P_{ij}
3. **For** 1 to M
4. **For** 1 to N
5. Apply DRL algorithm to compute $reward$
6. Compute $sub-swarm\ fitness$
7. $sub-swarm\ fitness = reward$
8. Compare current $sub-swarm\ fitness$ with P_{ij} and find S_{best}
9. **If** ($P_{ij} \leq sub-swarm\ fitness$)
10. set S_{best} is $sub-swarm\ fitness$
11. **End If**
12. Update position and velocity of the particle according to (24) and (25)
13. **End For**
14. Compare all S_{best} and find G_{best}
15. $G_{best} = \min\{S_{best\,1}, S_{best\,2}, \dots, S_{best\,k}\}$
16. Return G_{best}
17. Update iteration
18. Change velocity and position of the particle according to (27) and (28)
19. Repeat step until we get the optimum result
20. **End For**
21. **End**

TABLE V
TASK PROPERTIES

Parameters	Value
Task range	2000-4000
Length	1000-20000
File Size	250-300

TABLE VI
SERVER AND VM PROPERTIES

Server Properties		VM Properties	
Parameters	Value	Parameters	Value
Server range	5	VM range	50
MIPS	3000	MIPS	1000-2000
Memory	512	Memory	256-512
CPU	7	CPU	5
Bandwidth	3000	Bandwidth	1000
		VMM	XEN

TABLE VII
PPSO AND DRL PROPERTIES

PPSO Properties		DRL Properties	
Parameters	Value	Parameters	Value
Number of particles	50	Maximum iteration	100
$c_1, c_2 \& c_3$	2	Learning rate	0.1
$r_1, r_2 \& r_2$	[0,1]	Discount factor	0.9
Maximum iteration	100	Value of ∈	0.5 to 0.9

B. RESULT AND ANALYSIS

Our simulation is based on three different types of experiments, where we find the best result to optimize our objective. In the first experiment, we take two different sets of tasks and calculate the reward percentage. Also, we show the comparison of computation time and energy consumption between our proposed method and its competitors. In the second experiment, we show the accuracy of our proposed algorithm, and in the third experiment, we show the speedup process.

Fig. 4 and Fig. 5 show the reward that we get from the environment to minimize the completion time and energy consumption. From both figures, we compare the reward value of our proposed algorithm with three other existing algorithms. From Fig. 4 and Fig. 5, we take two different sets of tasks, 2000 and 4000. They are independent and have varying lengths as well as speeds. But for all situations, our server and VM numbers are fixed as per Table VI. In both Fig. 4 and Fig. 5, the y-axis represents the reward percentage and the x-axis represents the iteration number.

Fig. 4 shows the total reward per iteration under 2000 tasks set. From iteration 10 to iteration 40, the reward percentage increases from 25.5% to 41.5%. After reaching iteration 40, the reward percentage marginally fluctuates in between 40% and 42.5%. Finally, an approximate 40% reward is obtained. Table VIII provides detailed information about various scheduling algorithms in terms of energy consumption, completion time, and reward percentage.

From the experiment, we find that our proposed algorithm shows 15.7%, 12% and 13.1% better rewards as compared to MPSO, A3C, and DQN scheduling algorithms.

TABLE VIII
VARIOUS SCHEDULING ALGORITHM WITH REWARD PERCENTAGE OF 2000 TASK SET

Scheduling Algorithm	Energy Consumption (watt)	Completion Time (s)	Reward (%)
MPSO	186036.03	35.99	24.3

A3C	167111.79	20.92	28
DQN	168088.92	29.75	26.9
DRLPPSO	163564.75	20.85	40

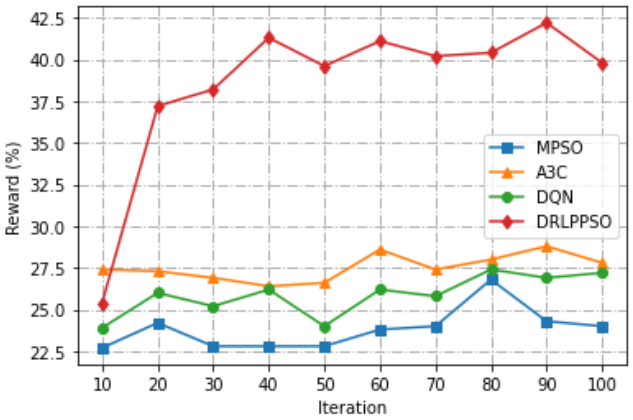

FIGURE 4. **Reward comparision of 2000 task sets.**

TABLE IX
VARIOUS SCHEDULING ALGORITHM WITH REWARD PERCENTAGE OF 4000 TASK SET

Scheduling Algorithm	Energy Consumption (watt)	Completion Time (s)	Reward (%)
MPSO	450172.95	77.3	24.2
A3C	404304.06	63.33	29.1
DQN	415256.39	65.97	26.4
DRLPPSO	384126.82	61.03	41.7

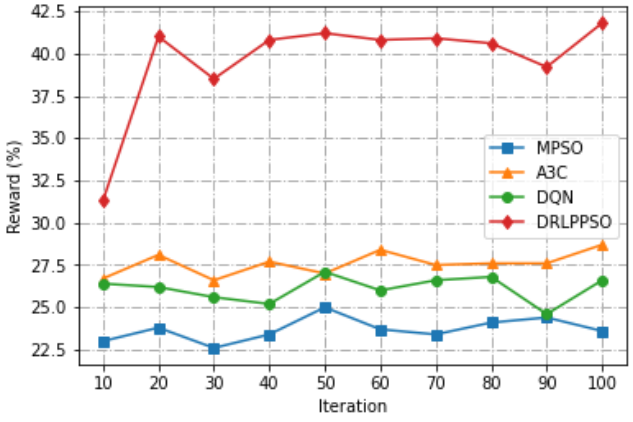

FIGURE 5. **Reward comparision of 4000 task sets.**

Fig. 5 shows the total reward per iteration under 4000 tasks set. As similar to Fig. 4, Fig. 5 shows that our algorithm also gives better rewards as compared to other algorithms. Table IX. provides detailed information about various scheduling algorithms in terms of energy consumption, completion time, and reward percentage. It is observed that DRLPPSO shows 17.5%, 12.6%, and 15.3% better rewards as compared to MPSO, A3C, and DQN scheduling algorithms.

From Fig. 4 and Fig. 5, we observe that our proposed algorithm is better compared to other algorithms even if there is a greater number of handled tasks. If the number of tasks set increases from 2000 to 4000, then the percentage of rewards also increases as compared to MPSO, A3C, and DQN scheduling algorithms.

The outcome displayed in Fig. 6 is obtained by various algorithms having 2000 to 4000 task number allocated to available VMs. From this figure, our proposed algorithm takes less computation time than the other three algorithms. Table VIII and IX represent the datasets for computation time where the minimum computation time of MPSO, A3C, DQN, and DRLPPSO is 35.99, 20.92, 29.75, and 20.85 seconds and the maximum computation time is 77.3, 63.33, 65.97, and 61.03 seconds.

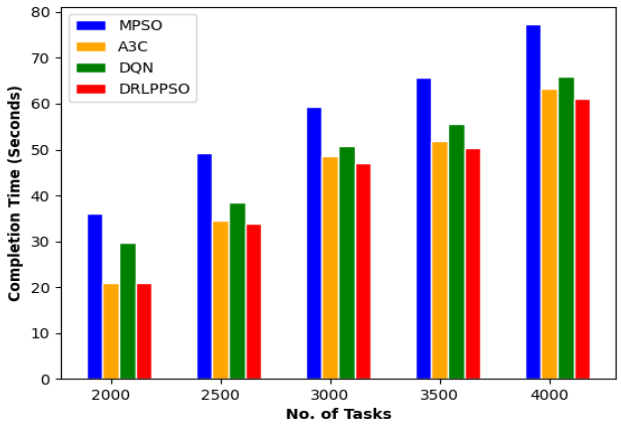

FIGURE 6. **Completion time under different task sets.**

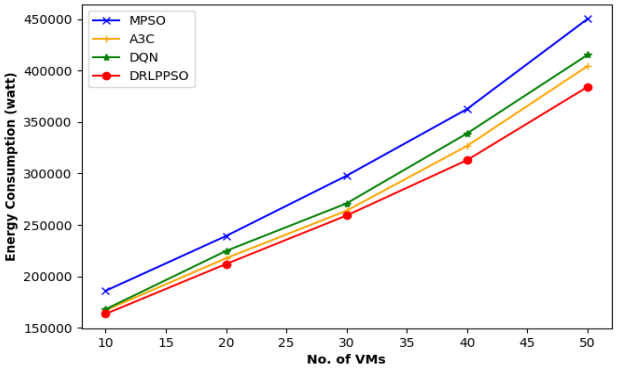

FIGURE 7. **Energy consumption under different virtual machines.**

Fig. 7 shows the total energy consumption of different VMs in a datacenter. From this figure, we found that our proposed algorithm saves more energy than the other three algorithms. Table VIII and IX represent the datasets for energy consumption where the minimum energy consumption of MPSO, A3C, DQN, and DRLPPSO is 186036.03, 167111.79, 168088.92, and 163564.75 watts and

the maximum consumption is 450172.95, 404304.06, 415256.39, and 384126.82 watts.

The trials are carried out based on performance measures such as accuracy and speedup. The accuracy behavior of our suggested algorithm is depicted in Fig. 8, where the y-axis addresses the accuracy value and the x-axis addresses the number of iterations. From this figure, it is clearly observed that our system accuracy has continuously increased from the start to iteration number 20. DRLPPSO accuracy is marginally fluctuating after iteration 20, and it is nearly at 0.838889 at the end of 100 iterations. Thus, the final result of training a neural network to get the best accuracy value at the end of each iteration is approximately 0.838889. This result clearly shows the improved accuracy in dynamic environments. Table X. represents the iteration number and its corresponding accuracy value of our proposed scheduling algorithm.

TABLE X
ITERATION NUMBER WITH ACCURACY VALUE

Iteration Number	Accuracy	Iteration Number	Accuracy
10	0.481111	60	0.833333
20	0.816667	70	0.824556
30	0.832889	80	0.838889
40	0.803111	90	0.838889
50	0.805556	100	0.838889

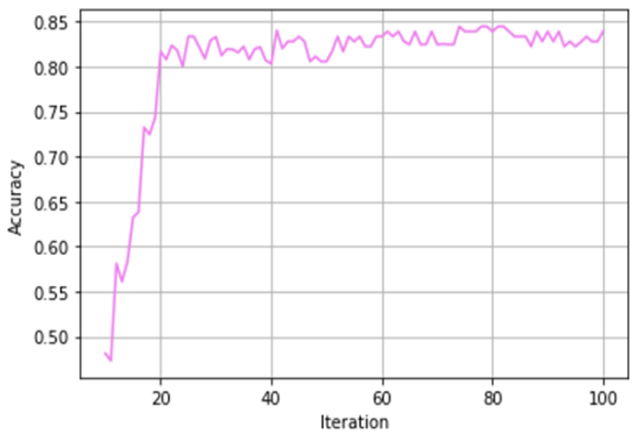

FIGURE 8. **Accuracy value.**

According to Amdahl's Law, speedup is calculated by the ratio of execution time on one CPU in sequential and the execution time for all CPUs in parallel. The speedup of our proposed algorithm is shown in (29).

$$S_u = E_s / E_p \tag{29}$$

where, S_u is the speedup, E_s is the execution time on one CPU in sequential mode, and E_p is the execution for all CPU in parallel mode. But in this paper, we deal with parallel processing computing. Therefore, we follow Gustafson law. This law simplifies Amdahl's Law. According to Gustafson's law, speedup is represented in (30).

$$S_u = 1 + (C - 1) \times p \tag{30}$$

where, C is the number of CPU and p is the fraction which lies between 0.2 to 0.99.

In our experiment, we took a total number of seven CPUs and took $p = 0.5$. Fig. 9 represents the speedup process of our proposed algorithm where we compare our speedup process with the existing PSO algorithm. From the figure, the speedup increases when the number of CPUs increases. DRLPPSO shows a great improvement in speedup compared to the PSO method. This proves that DRLPPSO is very suitable to be implemented in parallel computing and to solve large problems. Table XI. represents the details of the speedup value between DRLPPSO and existing PSO algorithm.

TABLE XI
SPEEDUP VALUES IN SECONDS

Number of CPU	DRLPPSO	PSO
1	1	0.56
2	1.5	0.71
3	2	0.92
4	2.5	1.18
5	3	1.37
6	3.5	1.65
7	4	1.92

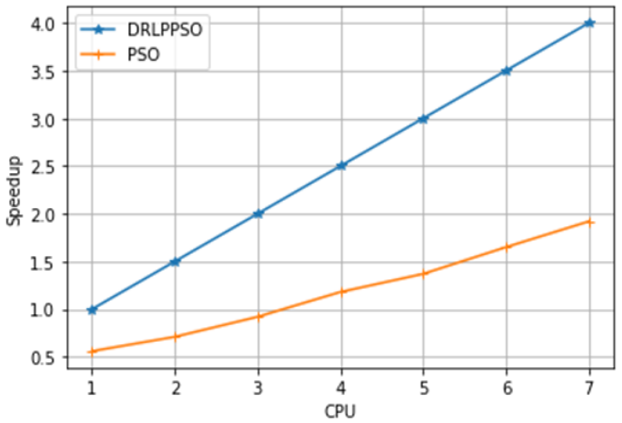

FIGURE 9. **Speedup value.**

VI. CONCLUSION

This paper presents a parallel computing scheduling algorithm which is known as the Deep Reinforcement Learning with Parallel PSO (DRLPPSO) scheduling algorithm. This algorithm is based on both the DRL learning algorithm and the Parallel PSO algorithm. Through the DRL learning algorithm, we train our neural network to get the best reward. By using PPSO, the overall processing time of all the incoming load is reduced. This scheduling algorithm

is proposed to achieve improvement in various parameters of load balancing with a minimum time period as compared to other popular existing scheduling algorithms in a cloud environment. Our simulation experiment is done with the help of Google Colab with the Python environment and TensorFlow. In the simulation, we carried out three different experiments that showed reward percentage, accuracy, and speedup process. When compared to the MPSO, A3C, and DQN scheduling algorithms, the DRLPPSO scheduling algorithm improves rewards by 15.7%, 12%, and 13.1% when the task set is 2000, and by 17.5%, 12.6%, and 15.3% when the task set is 4000. This information concludes that our proposed algorithm gives better rewards even if a large number of tasks come to the datacenter. The second experiment result shows the accuracy value, which is approximately 0.838889. The final experiment result presents the speedup process which is compared between DRLPPSO and the existing PSO algorithm.

In the future, we will compare our proposed algorithm with other meta-heuristic algorithms such as Parallel PSO (PPSO), Genetic Algorithm (GA), and Ant Colony Optimization (ACO). Also, to improve resource allocation and resource management concepts, a hybrid algorithm has been proposed, which is a combination of both swarm optimization and machine learning algorithms. This algorithm will provide real-time analytics on the complex and dynamic cloud network.

REFERENCES

[1] Z. Peng, Q. Gong, Y. Duan and Y. Wang, "The Research of the Parallel Computing Development from the Angle of Cloud Computing," Journal of Physics: Conf. Series 910 012002, 2017.

[2] R. M. Alguliyev, Y. N. Imamverdiyev and F. J. Abdullayeva, "PSO-based Load Balancing Method in Cloud Computing," Automatic Control and Computer Sciences. Vol. 53, No. 1, pp. 45–55, 2019.

[3] A. Pradhan, S. K. Bisoy and A. Das, "A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment," Journal of King Saud University –Computer and Information Sciences, Elsevier, 2021.

[4] M. E. Hatem and A. R. Rabie, "Resource Scheduling for Offline Cloud Computing Using Deep Reinforcement Learning," International Journal of Computer Science and Network Security (IJCSNS), VOL.19 No.4, pp 54-60, 2019.

[5] X. Hu and Y. Sun, "A Deep Reinforcement Learning-Based Power Resource Management for Fuel Cell Powered Data Centers," Electronics 9, 2054, pp-1-14, 2020.

[6] C. Chi, K. Ji, P. Song, A. Marahatta, S. Zhang, F. Zhang, D. Qiu and Z. Liu, "Cooperatively Improving Data Center Energy Efficiency Based on Multi-Agent Deep Reinforcement Learning," Energies 14, 2071, 2021.

[7] A. Pradhan, S. K. Bisoy, and A. Das, "A survey on PSO based meta-heuristic scheduling mechanism in cloud computing environment," Journal of King Saud University - Computer and Information Sciences, 2021.

[8] F. Jiang, L. Dong, K. Wang, K. Yang and C. Pan, "Distributed Resource Scheduling for Large-Scale MEC Systems: A Multiagent Ensemble Deep Reinforcement Learning with Imitation Acceleration," in IEEE Internet of Things Journal, vol. 9, no. 9, pp. 6597-6610, 2022.

[9] U. Rugwiro, C. Gu and W. Ding, "Task Scheduling and Resource Allocation Based on Ant-Colony Optimization and Deep Reinforcement Learning," Journal of Internet Technology, Vol 20, No.5, pp 1463-75, 2019.

[10] T. Liu, L. Li, G. Shao, X. Wu, and M. Huang, "A novel policy gradient algorithm with PSO-based parameter exploration for continuous control," Engineering Applications of Artificial Intelligence, Elsevier 90 (2020) 103525, pp 1-11, 2020.

[11] 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. Elsevier, pp 1-11, 2020.

[12] L. Huang, X. Feng, C. Zhang, L. Qian and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," Digital Communications and Networks, 2018.

[13] J. Wang, L. Zhao, J. Liu and N. Kato, "Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach," IEEE Transactions on Emerging Topics in Computing, 2019.

[14] Z. Tong, F. Ye, B. Liu, J. Cai and J. Mei, "DDQN-TS: A novel bi-objective intelligent scheduling algorithm in the cloud environment," Neurocomputing, 455, 419–430, 2021.

[15] G. Xu, Q. Cui, X. Shi, H. Ge, Z. -H. Zhan, H. P. Lee, Y. Liang, R. Tai and C. Wu, "Particle swarm optimization based on dimensional learning strategy," Swarm and Evolutionary Computation, 45, 33–51, 2019.

[16] 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, Elsevier, 2020.

[17] S. Lee, D. Jha, A. Agrawal, A. Choudhary and W. K. Liao, "Parallel Deep Convolutional Neural Network Training by Exploiting the Overlapping of Computation and Communication," IEEE 24th International Conference on High Performance Computing (HiPC), pp 183-192, 2017.

[18] M. H. Sharif and O. Gursoy, "Parallel Computing for Artificial Neural Network Training using Java Native Socket Programming," Periodicals of Engineering and Natural Sciences, Vol. 6, No. 1, pp. 1 – 10, 2018.

[19] H. Wu, A. Ozdemir, A. Zeljic, K. Julian, A. Irfan, D. Gopinath, S. Fouladi, G. Katz, C. Pasareanu and C. Barrett, "Parallelization Techniques for Verifying Neural Networks," arXiv:2004.08440v3 [cs. LO], 2020.

[20] P. S. Mashhadi, S. Nowaczyk and S. Pashami, "Parallel orthogonal deep neural network," Neural Networks, Elsevier, 140 167–183, 2021.

[21] Y. Tan and Y. Zhou, "Parallel Particle Swarm Optimization Algorithm Based on Graphic Processing Units," Handbook of Swarm Intelligence, 133–154, 2011.

[22] S. F. Kazemi and Y. Shafahi, "An Integrated Model of Parallel Processing and PSO Algorithm for Solving Optimum Highway Alignment Problem," 27th Conference on Modelling and Simulation, 2013.

[23] N. Dali and S. Bouamama, "GPU-PSO: Parallel Particle Swarm Optimization Approaches on Graphical Processing Unit for Constraint Reasoning: Case of Max-CSPs," Procedia Computer Science, 60, 1070–1080, 2015.

[24] P. Chanthini and K. Shyamala, "A Survey on Parallelization of Neural Network using MPI and Open MP," Indian Journal of Science and Technology, Vol 9, 2016.

[25] F. E. F. Junior and G. G. Yen, "Particle swarm optimization of deep neural networks architectures for image classification," Swarm and Evolutionary Computation 49, pp 62–74, 2019.

[26] M. Cheng, J. Li and S. Nazarian, "DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC, 129–134, 2018.

[27] X. Xiong, K. Zheng, L. Lei and L. Hou, "Resource Allocation Based on Deep Reinforcement Learning in IoT Edge Computing," IEEE Journal on Selected Areas in Communications, 38, 1133–1146, 2020.

[28] S. Sheng, P. Chen, Z. Chen, L. Wu and Y. Yao, "Deep Reinforcement Learning-Based Task Scheduling in IoT Edge Computing," Sensors, 21, 1666, 2021.

[29] L. A. Barroso, J. Clidaras and U. Holzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," Synthesis lectures on computer architecture, vol. 8, no. 3, pp. 1–154, 2013.

[30] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka and A. D. George, "Parallel global optimization with the particle swarm algorithm," International Journal for Numerical Methods in Engineering, 61:2296–2315, 2004.
[31] E. S. Alkayal, N. R. Jennings and M. F. Abulkhair, "Automated Negotiation using Parallel Particle Swarm Optimization for Cloud Computing Applications," International Conference on Computer and Applications (ICCA), pp 26-35, 2017.
[32] A. Abdelaziz, M. Anastasiadou and M. Castelli, "A Parallel Particle Swarm Optimisation for Selecting Optimal Virtual Machine on Cloud Environment," Applied Sciences, 10, 6538, 2020.
[33] T. Gonsalves and A. Egashira, "Parallel Swarms Oriented Particle Swarm Optimization," Applied Computational Intelligence and Soft Computing, Volume 2013, Article ID 756719, pp 1-7, 2013.
[34] M. Vidhya and N. Sadhasivam, "Parallel Particle Swarm Optimization for Task Scheduling in Cloud Computing," International Journal of Innovative Research in Science, Engineering and Technology, Vol. 4, Special Issue 6, pp 136-140, 2015.
[35] A. Abdelaziz, M. Elhoseny, A. S. Salama and A. Riad, "A machine learning model for improving healthcare services on cloud computing environment," Measurement, 119, 117–128, 2018.
[36] Z. Peng, J. Lin, D. Cui, Q. Li and J. He, "A multi-objective trade-off framework for cloud resource scheduling based on the Deep Q-network algorithm," Cluster Computing, Springer, 2020.
[37] J. Lin, D. Cui, Z. Peng, Q. Li and J. He, "A Two-Stage Framework for the Multi-User Multi-Data Center Job Scheduling and Resource Allocation," IEEE Access, Vol 8, pp 197863-74, 2020.
[38] H. Che, Z. Bai, R. Zuo and H. Li, "A Deep Reinforcement Learning Approach to the Optimization of Data Center Task Scheduling," Hindawi Complexity, Wiley, Vol 2020, Article ID 3046769, pp 1-12, 2020.
[39] T. Dong, F. Xue, C. Xiao and J. Li, "Task scheduling based on deep reinforcement learning in a cloud manufacturing environment," Concurrency and Computation: Practice and Experience. Wiley, pp 1-12, 2020.
[40] S. Mostafavi and V. Hakami, "A Stochastic Approximation Approach for Foresighted Task Scheduling in Cloud Computing," Wireless Personal Communications, Springer, 2020.
[41] Y. Wei, L. Pan, S. Liu, L. Wu, and X. Meng, "DRL-Scheduling: an intelligent QoS-aware job scheduling framework for applications in clouds," IEEE Access, vol. 6, pp. 55 112–55 125, 2018.
[42] J. Yan, Y. Huang, A. Gupta, A. Gupta, C. Liu, J. Li and L. Cheng, "Energy-aware systems for real-time job scheduling in cloud data centers: A deep reinforcement learning approach," Computers and Electrical Engineering 99 (2022) 107688, 2022.
[43] Z. Zhou, J. Chang, Z. Hu, J. Yu and F. Li, "A modified PSO algorithm for task scheduling optimization in cloud computing," Concurrency Computation Practice Experiment: e4970, Wiley, pp 1-11, 2018.
[44] Z. Chen, J. Hu, G. Min, C. Luo and T. El-Ghazawi, "Adaptive and Efficient Resource Allocation in Cloud Datacenters Using Actor-Critic Deep Reinforcement Learning," in IEEE Transactions on Parallel and Distributed Systems, vol. 33, no. 8, pp. 1911-1923, 2022.
[45] T. Renugadevi, K. Geetha, N. Prabaharan and P. Siano, "Carbon-Efficient Virtual Machine Placement Based on Dynamic Voltage Frequency Scaling in Geo-Distributed Cloud Data Centers," Applied Sciences, 10, 2701, 2020.
[46] V. Divya and S. R. Leena, "Intelligent Deep Reinforcement Learning based Resource Allocation in Fog network," 26th International Conference on High Performance Computing, Data, and Analytics Workshop (HiPCW), pp 18-22, 2019.
[47] F. Fu, Z. Zhang, F. R. Yu and Q. Yan, "An actor-critic reinforcement learning-based resource management in mobile edge computing systems," International Journal of Machine Learning and Cybernetics, Springer, 2020.
[48] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, p. 436-444, 2015.