Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 1533949, 11 pages
https://doi.org/10.1155/2022/1533949

Research Article

A Blind Load-Balancing Algorithm (BLBA) for Distributing Tasks in Fog Nodes

Niloofar Tahmasebi-Pouya, Mehdi-Agha Sarram, and Seyedakbar Mostafavi

Computer Engineering Department, Yazd University, Yazd, Iran

Correspondence should be addressed to Seyedakbar Mostafavi; a.mostafavi@yazd.ac.ir

Received 8 March 2022; Revised 28 June 2022; Accepted 20 July 2022; Published 11 August 2022

Academic Editor: Andrea Marin

In the distributed infrastructure of fog computing, fog nodes (FNs) can process user requests locally. In order to reduce the delay and response time of a user's requests, incoming requests must be evenly distributed among FNs. For this purpose, in this paper, we propose a blind load-balancing algorithm (BLBA) to improve the load distribution in the fog environment. In the proposed algorithm, the mobile device sends a task to a FN. Then, the FN decides to process that task using the Double-Q-learning algorithm. One of the critical advantages of BLBA is that decision-making on tasks is done without any knowledge of the state of neighbor nodes. The proposed system consists of four layers: (i) IoT layer, (ii) fog layer, (iii) proxy server layer, and (iv) cloud layer. The experimental results show that the proposed algorithm with proper distribution of tasks between nodes significantly reduces the delay and user response time compared to the existing methods.

1. Introduction

Fog computing is a distributed computing model that extends cloud services to the edge of the network to facilitate the management and scheduling of computing, networking, and storage services between data centers and end devices. Both fog computing and cloud computing provide computing, storage, and networking services to end-users, but fog is closer to the end-user, thus providing minimal delay for Internet of Things (IoT) applications. FNs are located in a layer between IoT and the cloud data center. FNs can process data stream and user requests in real time, reducing network delay and congestion [1–3].

IoT devices typically assign processing tasks to the nearest neighbor node. In this case, some FNs may receive more tasks than other FNs and be overloaded over time. To avoid this situation, load-balancing methods are suggested to distribute loads over the nodes. Load-balancing in FNs refers to the even distribution of input tasks across a group of processing FNs so that the capacity of FNs is fairly utilized and task processing speed is increased [4–8]. FNs can allocate their tasks to underloaded neighbor nodes or the cloud through the load-balancing approach and reduce overload and processing delay as much as possible. The load-balancing approaches in the fog environment can be categorized as static and dynamic. Static load-balancing algorithms perform load-balancing and apply fixed rules to distribute task requests. On the other side, in dynamic load-balancing, the tasks are assigned dynamically to the FNs based on a long-term knowledge of load distribution. In other words, dynamic load-balancing approaches update their load-balancing rules frequently based on the new knowledge of traffic loads [9, 10]. Dynamic load-balancing algorithms can be divided into two categories: (1) sender-initiated techniques where congested nodes look for lightly loaded nodes and offload their tasks to them and (2) receiver-initiated strategies where underloaded nodes search for overloaded nodes and steal their tasks [11, 12]. In this paper, we propose a dynamic load-balancing method based on sender-initiated strategies for task distribution over FNs.

The nature-inspired load-balancing algorithms can be classified into three different types: heuristic, metaheuristic, and hybrid. The purpose of designing heuristics is to achieve the optimal response in a specified period [13–15]. Metheuristic algorithms require more execution time to achieve the optimal response, and these algorithms have a more extensive response space than heuristics [16–19]. Hybrid algorithms combine heuristic or metaheuristic algorithms that reduce execution time and cost and provide more efficient results than other algorithms [20–24]. The use of typical load-balancing methods increases resource utilization and resource savings, as well as reduces delay and response time. However, these algorithms may lose their efficiency because of the time-varying dynamics of traffic load in fog computing. Therefore, we need an algorithm that can adapt to dynamics in environmental conditions. For this purpose, we introduce a decision-making process based on the Double-Q-learning algorithm to evenly distribute processing tasks among FNs. The main contributions of the proposed approach are summarized below:

(i) *Architecture*. The proposed system considers a four-layer architecture to handle load-balancing problems in the fog environment. Because of this architecture, tasks are processed locally at FNs, and there is no need to transfer data to the cloud.

(ii) *Algorithm*. This work proposes a decision-making process based on the Double-Q-learning algorithm to find a low-load FN. The FN using the Double-Q-learning algorithm selects an available neighbor FN or cloud to assign the task. This algorithm can be trained to maximize the long-term reward. In this algorithm, the agent makes decisions about the task processing without knowledge of the fog environment and only based on the observations and rewards. The results show that the load-balancing method based on the Double-Q-learning algorithm has significantly reduced the delay and response time than the compared approaches.

(iii) *Mechanism*. This work proposes a load-balancing method based on the Double-Q-learning algorithm to distribute the tasks evenly among FNs with the goal of reducing processing time. To our knowledge, most of the methods presented in previous research for decision-making require knowledge of the capacity of neighbor FNs and the cloud, which creates a traffic load on the network and also delays the decision-making process. In the BLBA, the FN just decides based on the information obtained during the learning period from delay and rewards based on its own condition and has no knowledge of the status of neighbor nodes. In this method, the FN learns to assign the received task to a low-load node for faster processing.

Our algorithm operates in such a way that the nodes have no initial knowledge of the position of other nodes in the fog environment, and during the learning period, they act only on the basis of experiences related to their conditions and have no knowledge of other neighbor nodes. We refer to such an algorithm as the blind algorithm for the proper distribution of tasks between FNs to stress the fact that there is no prior knowledge of the status of neighbor nodes. This algorithm can be implemented on other networks and run on the fly.

We organized this paper as follows: In Section 2, we offer the related works. In Section 3, we describe the proposed system architecture, the reinforcement learning algorithm, and how to compute the delay in this system. In Section 4, we introduce the proposed load-balancing method. In Section 5, the simulation results and our analysis of these results are presented. Finally, Section 6 offers a conclusion and suggestions for future work.

2. Related Work

In this section, we review the previous works on load-balancing using reinforcement learning. To minimize overload and reduce delay, it is critical to use an optimal load-balancing algorithm. In fog computing, IoT devices and mobile users typically assign their tasks to the nearest FN. Since these devices are often mobile, different FNs may have different loads depending on their position in the network. This causes an imbalance in the distribution of tasks between FNs, and some FNs may be overloaded, while other FNs are idle or low-load. In distributed environments, we can use reinforcement learning to design load-balancing algorithms that learn traffic patterns and automatically distribute the load evenly among the nodes. Some authors have used the benefits of reinforcement learning algorithms to solve the load-balancing problem. The existing studies can be classified from many perspectives. Here, we review several previous works that are aware of the capacity and load of the nodes.

2.1. Literature Review. Many of the previous studies are founded based on the assumption of knowledge of node capacity. Baek et al. [10] proposed a decision-making process based on reinforcement learning to find the optimal offloading decision with unknown reward and transition functions. In this method, FNs can send some tasks to an available neighbor FN. The purpose of this is to minimize overload probability and processing time. Xu et al. [25] introduced a dynamic resource allocation method for load-balancing in the fog environment. Technically, they presented a system framework for fog computing and the load-balancing analysis for various types of computing nodes. Then, they designed a corresponding resource allocation method in the fog environment through static resource allocation and dynamic service migration to achieve load-balancing for fog computing systems. Moon et al. [26] defined a computational task migration problem for balancing loads of vehicular edge computing servers (VECSs) and minimizing migration costs. To solve this problem, they adopt a reinforcement learning algorithm in a cooperative VECS group environment that can collaborate with VECSs in the group. The objective of this study is to optimize load-balancing and migration cost while satisfying the delay constraints of the computation task of vehicles. Wu et al. [27] proposed a reinforcement learning-based metadata dynamic load-balancing mechanism. This method can control the load dynamically according to the performance of the metadata servers, and it has good adaptability in the case of a sudden change in data volume.

Other studies are based on the assumption of knowledge of node loads or future load predictions. Razaq et al. [28] proposed a Q-learning-based algorithm for load-balancing in the fog environment, in which a task is divided into several pieces based on security requirements to help in privacy preservation. In this algorithm, the agent assigns a task piece to a node with an equal or higher security reputation than the security level of a task piece that can provide service to avoid overload on the nodes. Xu et al. [29] proposed a work donation algorithm based on reinforcement learning for distributed-memory systems to optimize load-balancing with minimized communication costs and dynamically adapt to flow behaviors and available network bandwidth. Then, they designed a high-order load estimation model to predict blockwise particle advection loads and used a linear transmission model to estimate interprocess communications' costs. Mai et al. [30] suggested a reinforcement learning-based method that uses evolution strategies to assign tasks between fog servers to minimize processing latency in the long term. Talaat et al. [31] introduced a load-balancing and optimization strategy using a dynamic resource allocation method based on reinforcement learning and genetic algorithm. This method collects the load information for each server, handles the incoming requests, and distributes them between the servers evenly. Divya and Sri [32] proposed a reinforcement learning-based load-balancing method by combining software-defined networks and fog computing. The proposed method understands the network behavior and balances the loads to provide the maximum possible availability of the resources. Lu et al. [33] used improved deep reinforcement learning based on LSTM and candidate networks to solve tasks offloading in mobile edge computing. Li et al. [34] suggested a load-balancing method using an online reinforcement learning algorithm for load distribution in vehicular networks. This algorithm achieves a suitable association solution through continuous learning from the dynamic vehicular environment. Lin et al. [35] introduced a reinforcement learning-based approach aimed at load-balancing for data center networks. This approach employs reinforcement learning to learn a network and control it based on the learned experience. Li et al. [36] suggested an algorithm based on machine learning which is aimed at generating intelligent adaptive strategies related to load-balancing of collaborative servers and dynamic scheduling of sequential tasks. Based on the proposed algorithm and software-defined networking technology, the tasks can be executed cooperatively by the user device and the servers in the mobile fog computing network. Rikhtegar et al. [37] proposed a load-balancing method based on deep reinforcement learning for software-defined networking-based data center networks. This method uses the deep deterministic policy gradient algorithm to adaptively learn the link-weight values by observing the traffic flow characteristics. Kim and Kim [38] proposed an agent that uses a deep reinforcement learning algorithm to distribute requests between gaming servers. The agent has done this by measuring network loads and analyzing a large amount of user data.

2.2. Research Gap and Motivation. To our knowledge, most of the methods presented in previous research for decision-making require knowledge of the capacity of neighbor FNs and the cloud (e.g., [10, 26, 27]), which creates a traffic load on the network and also delays the decision-making process. Our work in this paper differs from previous works as in our method, the FN just decides based on the information obtained during the learning period from delay and reward based on its own condition and has no knowledge of the status of neighbor nodes. Our work in this paper enables load-balancing in a dynamic fog environment where the nodes have no information about each other. The application of the proposed scheme is not limited to a specific scenario, but its purpose is a subclass of problems.

3. System Model

In this section, we describe the proposed system architecture, the reinforcement learning algorithm, and how to compute the delay in this system.

3.1. Proposed System Architecture. In this paper, as shown in Figure 1, a four-layer architecture is considered for the proposed system. The first layer includes IoT devices that connect directly to FNs and send data to these nodes locally. The second layer is the fog layer. Fog servers can be located in different geographical locations and process data received from IoT devices in real time. The third layer comprises a proxy server that receives data from FNs and then sends this data to the cloud. The last layer in this structure is the cloud data center layer, which includes several servers and data centers. Because of this structure, data and information are processed locally at FNs, and there is no need to transfer data to the cloud.

FNs can allocate their tasks to low-load neighbor nodes or the cloud through the load-balancing method provided for the dynamic fog environment in this paper and reduce overload and processing delay as much as possible. Because of the dynamic of the fog environment, a variable number of mobile devices may be connected to each FN at any moment. The FN to which more mobile devices are connected receives more tasks than other FNs and will be overloaded. Load-balancing methods are used to evenly distribute tasks among FNs to avoid overload. The primary purpose of the load-balancing algorithm in the fog computing environment is to improve the response time so that it operates optimally even in dynamic conditions of the system. For this purpose, in this paper, the Double-Q-learning algorithm is applied in FNs to improve delay, response time, and resource loss in the network. After receiving a task, each FN decides to use the Double-Q-learning algorithm to process it or send it to a neighbor FN or cloud for faster processing.

3.2. Preliminaries on the Reinforcement Learning Algorithm. In this part, we review the background on reinforcement learning and Double-Q-learning algorithm:

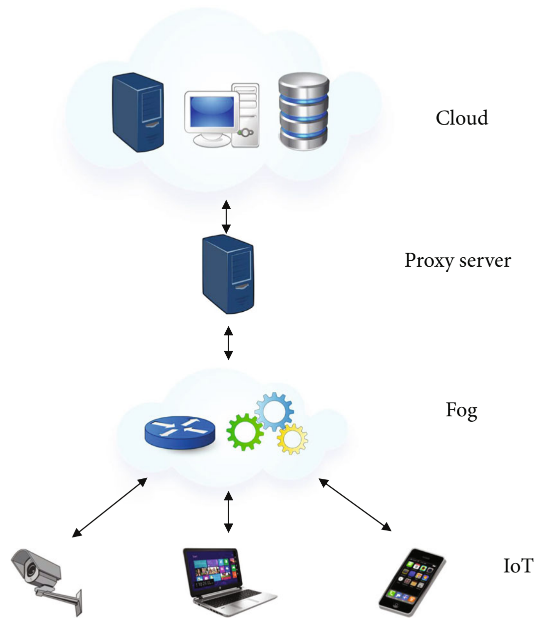

Figure 1: The four-layer architecture of fog computing.

(i) *Reinforcement Learning Algorithm*. In this paper, we formulate the load-balancing approach with the reinforcement learning algorithm. Specifically, the reinforcement learning algorithm maximizes the cumulative reward by selecting optimal action in each state of the environment [39, 40]. The proposed method formulates the load-balancing problem as a Markov decision process (MDP) for the dynamic fog environment. The MDP comprises a decision-making agent that continuously observes the current state s of the system, selects an action a from the allowed actions in that state $(a \in A(s))$, and then transitions to a new state s' and receives a reward r for that action, which influences future decisions [41].

(ii) *Off-Policy Learning*. The policy is a mapping from one state to one action, which determines how to deal with each action and how to make a decision in each of the different situations and is defined in the two forms of On-policy and Off-policy. In On-policy, the same policy is used for both optimization and action selection purposes. However, in Off-policy, two separate policies are used for optimization and select action [42]. In this paper, among the reinforcement learning algorithms, the Double-Q-learning algorithm is used. The Double-Q-learning algorithm is an Off-policy algorithm.

(iii) *Value Function*. In the reinforcement learning algorithm, the value function is defined as the received long-term expected cumulative rewards, which has a long-term view, and for each state, a value is determined as follows:

$$v^*(s) = \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v^*\left(s'\right)\right], \tag{1}$$

where $0 < \gamma < 1$ is called the discount factor, which determines the importance of future rewards and shows that the current decision has more value than future decisions.

(iv) *Model*. The model of the Double-Q-learning algorithm is random, and its states are indefinite.

In a reinforcement learning problem, the agent explores the environment and learns to select the optimal action to maximize long-term reward. Hence, reinforcement learning in dynamic environments has many applications for optimization. In addition, it can be an excellent way to evenly distribute tasks between FNs.

3.3. Problem Formulation. We have formulated the proposed load-balancing problem as the MDP to achieve the desired performance. An MDP usually consists of $<S, A, P, R>$, which are defined for the proposed load-balancing problem as follows:

(i) $S = \{s = (C, Q, N)\}$. It is the state space, where C represents the capacity of FN, Q represents the forward queue size in FN, and N represents the number of mobile devices connected to FN. Decision-making in the Double-Q-learning algorithm is made based on the current state of the system. Most of the previous methods to define the state space require knowledge about the capacity of neighbor FNs. However, in the BLBA, the state of the system is defined only based on the status of the decision-maker FN, and this causes the decision-making to be done without any knowledge of the status of the neighbor nodes.

(ii) $A = \{a = (n)\}$. It is the action space, where n represents the selected FN or cloud to assign the task.

(iii) P. The transition probability is a value between $[0, 1]$. The transition probability distribution $P(s' \mid s, a)$ to the next state s' by selecting action a, if it is in the state s.

(iv) R. It is the reward for selecting the action a in the current state. The primary goal is to select the optimal action in each system so that the long-term value is maximized and the processing time and overload probability are minimized.

The task processing time is equal to the sum of the transmission delay and processing delay of that task in different devices. These delays are calculated as follows.

3.3.1. Task Transmission Delay. The transmission delay between two nodes is obtained from the sum of the waiting delay in the forward queue of the source node and the send delay on the communication channel between the two nodes, which is calculated as follows:

$$\delta = t_W + t_S, \quad (2)$$

where t_W is the waiting delay in the forward queue of the source node and is calculated as follows:

$$t_W = t_{\text{out}} - t_{\text{in}}, \quad (3)$$

where t_{in} represents the arrival time of the task m in the queue of a node and t_{out} represents the exit time of the task m from that node. The parameters needed to calculate the delay are given in Table 1.

In (2), t_S is the send delay on the communication channel between the two nodes and is calculated as

$$t_S = \frac{L_{\text{Task}}}{\text{BW}_N \cdot \log_2\left(1 + \beta_1 D_{i,j}^{-\beta_2} \cdot P_t/\sigma \cdot \text{BW}_N\right)}, \quad (4)$$

where $D_{i,j}$ represents the distance between two nodes.

3.3.2. Reward Function. In the BLBA, the Double-Q-learning algorithm runs on FNs, in which we defined the reward function $R(s, a)$ as the negative of the processing delay. If the task processing delay is longer, as a result, less reward is received. In this paper, $R(s, a)$ is calculated as follows:

$$R(s, a) = -\theta, \quad (5)$$

where θ represents the processing delay of the task assigned to the FN. θ is calculated in one of the following two ways:

(i) If the node itself (FN-I) that received the task from the mobile device processes it, θ is calculated as

$$\theta = t_{E_i}, \quad (6)$$

where t_{E_i} represents the task execution time in FN-I

(ii) If the task is assigned to the neighbor node (FN-J) or the cloud for processing, θ is calculated as

$$\theta = \delta_{ij} + t_{E_j} + \delta_{ji}, \quad (7)$$

where δ_{ij} represents the task transmission delay from FN-I to FN-J or cloud, t_{E_j} represents the task execution time in FN-J or cloud, and δ_{ji} represents the transmission delay the result of the task from FN-J or cloud to FN-I

3.3.3. Task Execution Time. After the node receives the task, that node allocates part of its capacity to execute this task. The task execution time in the FN-I, FN-J, or cloud is calculated as follows, where I represents the number of task instructions:

$$t_E = \frac{I \cdot C_{\text{CPU}}}{f_{\text{CPU}}}. \quad (8)$$

3.3.4. Total Delay. Depending on which node will process the task, the task processing time is calculated as follows:

$$T_{\text{task}} = \delta_{mi} + \theta + \delta_{im}, \quad (9)$$

where δ_{mi} represents the task transmission delay from the mobile device to FN-I and δ_{im} represents the transmission delay of the result of the task from FN-I to the mobile device.

Because the proxy server only sends the task to the cloud and does not perform any processing, it is assumed that the send delay on the communication channel from FN-I to the cloud and vice versa is calculated directly and without considering the proxy server.

Each FN is an agent that is learning in the network. Any new task in the system causes the FN to perform an action in the environment and select one node to assign the new task.

The reward of the selected action will be specified when updating the state of the environment. If the current state of the system is closer to the load-balancing and the tasks are processed faster, the reward will be given to the agent; otherwise, no reward will be awarded to that. Through the rewards received, each node learns to make the best decision for processing a task.

4. Blind Load-Balancing Algorithm (BLBA)

In this section, a Double-Q-learning-based load-balancing algorithm for proper distribution of the load between the FNs is presented to solve the problems of previous methods. The Double-Q-learning algorithm is used to find the optimal state-action with the least computational cost, which obtains enough information through experience. The model of the Double-Q-learning algorithm is random, and its states are indefinite. In a learning problem, the agent explores the environment and learns to select the optimal action to maximize long-term reward. Hence, the Double-Q-learning algorithm in dynamic environments has many applications for optimization. In addition, it can be an excellent way to evenly distribute tasks between FNs.

The Double-Q-learning algorithm uses two estimation functions instead of one estimation function: Q_1 and Q_2. This way, it uses two Q-tables for estimates that stored the value of all actions. The difference between the two tables is that when we update the value of one of the tables, we use the maximum value present in the other table. Assume that the action $a^* = \arg \max_a Q_1(s', a)$ is the most valuable action in the state s', according to the value function Q_1. We use the value $Q_2(s', a^*)$ to update Q_1. In a similar way, the action $b^* = \arg \max_a Q_2(s', a)$ is the most valuable action in the state s', according to the value function Q_2. We use b^* and Q_1 to update Q_2. In the Double-Q-learning algorithm, each time an update is performed, it is decided with equal probability that the value of which table is updated and which table is used to consider the maximum value. In this algorithm, an agent performs an action a after receiving the state s of the environment and then transitions to the next state s' and receives a reward $R(s, a)$ from the environment in return.

Table 1: Parameter values to calculate the delay.

Parameters	Definition	Value
BW_N	Bandwidth per node	10000 kB
	Cloud bandwidth	2000 kB
L_{Task}	Task data length	3800
β_1	The path loss constant	10^{-3}
β_2	The path loss exponent	4
P_t	The transmission power of node	20 dBm
σ	The noise power spectral density	174 dBm/Hz
C_{CPU}	The number of CPU cycles required to compute any instruction	5
f_{CPU}	The CPU speeds of the FN	2800
	The CPU speeds of the cloud	44800

The value function for state s and action a in the Double-Q -learning algorithm is estimated as follows:

$$Q_1(s,a) = Q_1(s,a) + \alpha\left[R(s,a) + \gamma Q_2\left(s', \arg\ \max_a Q_1\left(s',a\right)\right) - Q_1(s,a)\right], \tag{10}$$

$$Q_2(s,a) = Q_2(s,a) + \alpha\left[R(s,a) + \gamma Q_1\left(s', \arg\ \max_a Q_2\left(s',a\right)\right) - Q_2(s,a)\right], \tag{11}$$

where $(0 < \alpha < 1)$ is the learning rate, which balances between new observations and what has been learned. The Double-Q-learning algorithm uses the ε-greedy policy to maximize long-term value, in which ε indicates that the next action is randomly selected (with a constant probability of $0 \le \varepsilon \le 1$) or selected from among the best in the table (with probability $1-\varepsilon$). First, the algorithm does not have any information about the network, so it is in the form of greedy exploring the network. Once enough information is obtained from the network, load-balancing is performed optimally. In the ε-greedy algorithm, if we observe each action infinite times, we can be ensured that $Q(s,a)$ converges to the optimal value. Therefore, the FN learns through the Double-Q -learning algorithm to select the most suitable node to assign the task.

By applying load-balancing on FNs, the load is evenly distributed between these nodes. The FN is considered an agent and is busy learning in the network. After the FN receives a new task via the mobile device, it observes the current state of the environment. Then, to maximize the long-term reward, based on the experiences and rewards it has received so far and without any knowledge of the capacity of the other nodes and only according to its own capacity, it decides the task processing. If the FN has enough capacity, it processes the task itself; otherwise, it assigns the task processing to the neighbor FN or cloud. If the task processing delay in the FN itself and the neighbor FN is greater than the processing delay of that task in the cloud, then the FN assigns this task to the cloud for faster processing and reduces the load on other nodes. Figure 2 shows the flowchart of the proposed load-balancing method.

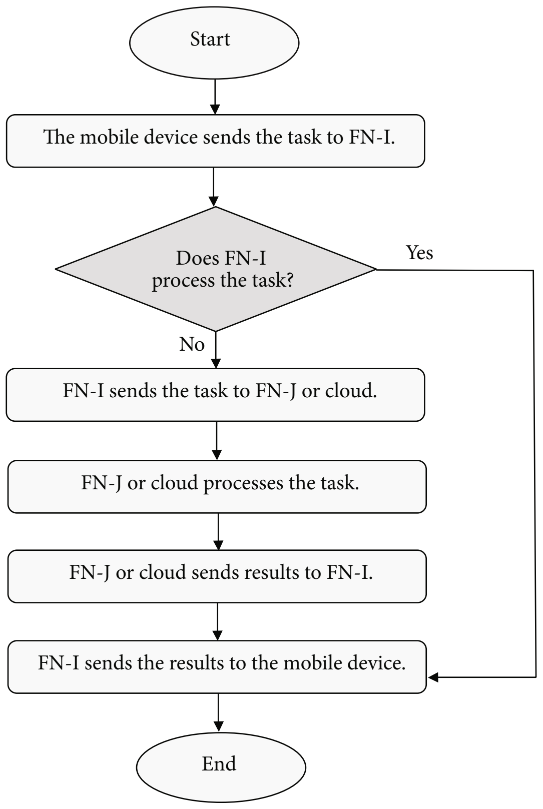
Start
The mobile device sends the task to FN-I.
Does FN-I process the task?
Yes
No
FN-I sends the task to FN-J or cloud.
FN-J or cloud processes the task.
FN-J or cloud sends results to FN-I.
FN-I sends the results to the mobile device.
End

Figure 2: Flowchart of the proposed load-balancing method.

The state space is equal to the capacity of the FN, the forward queue size in the FN, and the number of mobile devices connected to the FN; the action of selecting a FN or cloud to assign the task and the reward is a function to minimizing task processing delay.

5. Performance Evaluation

The performance of the proposed load-balancing problem based on the Double-Q-learning algorithm is evaluated using the iFogSim simulation environment [43]. We ran this program on an Asus computer with an Intel Core i7 processor and 8 GB RAM. The proposed system includes N FNs

Table 2: Parameter values in the simulation.

Parameters	Definition	Value
N	Number of FNs	4, 10
n	The number of times a state has been observed	—
α	Learning rate	$1/n$
γ	Discount factor	0.9

and a variable number of mobile devices. Mobile devices randomly connect to their neighbor FNs and assign their task processing to these nodes. First, in the Double-Q-learning algorithm, all Q-table values are zero, and the FN has no information about the network. In order to learn, the ε-greedy method is used, in which the value ε is initially considered equal to 1, and the algorithm in the form of greedy explores the network. After that, the FN's confidence increases in estimating Q-values; its value changes to 0.3.

The amount of reward received is equal to the negative of the task processing delay in one of the FNs or the cloud. In the simulation, it is assumed that mobile devices now send tasks to FNs, and the Double-Q-learning algorithm is executed simultaneously with received tasks by FNs. This will prevent overload in the nodes as much as possible. Each node using the Double-Q-learning algorithm selects a node to assign the task after examining the current state of the environment and receives a reward from the environment in return. Over time, the experience of the FNs from the network increases, and each node learns to assign the task to a low-load node that can process the task faster and receive a reward from the environment in return. However, in other methods, unlike the proposed method, the load-balancing algorithm is executed after creating an overload in the FN. This leads to reduced performance and increases the delay in these systems. The parameters used for system evaluation are given in Table 2.

In the following, the performance of BLBA is compared with SSLB [9], random, and proportional [44] load-balancing methods. In the random method, a node offloads tasks to a randomly picked neighbor. That is, when the FN overloads, it randomly selects a neighbor node and sends its load to it for faster processing. In the proportional method, the capacity information of the neighbors is received and selects the optimal one to offload a task. In the SSLB method, after a FN is overloaded, it compares the capacity of the other neighbor nodes and sends the task to the node that has at least 40% of its capacity empty and has the highest capacity.

In this section, we first consider the number of FNs as 4. Then, we increase the number of nodes to 10 and check the performance of the proposed algorithm in both conditions. Figure 3 shows the increase in cumulative reward at each time iteration of the proposed algorithm. In this paper, the reward is equal to the negative of the processing delay of the task assigned to the FN. Given that each task is processed by which node, reducing the processing delay of a task increases the reward received for processing that task. Increasing the number of tasks assigned to nodes leads to

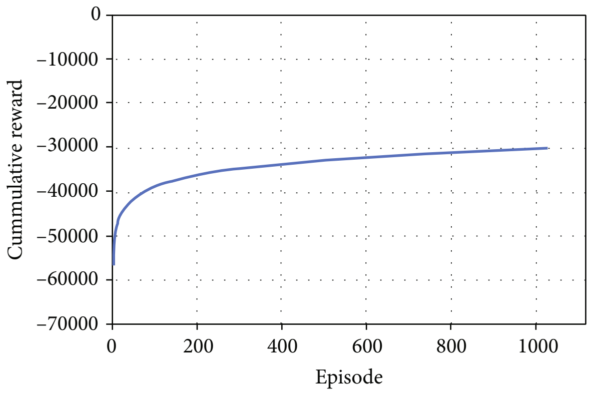

Figure 3: The cumulative reward for each time iteration of the proposed algorithm.

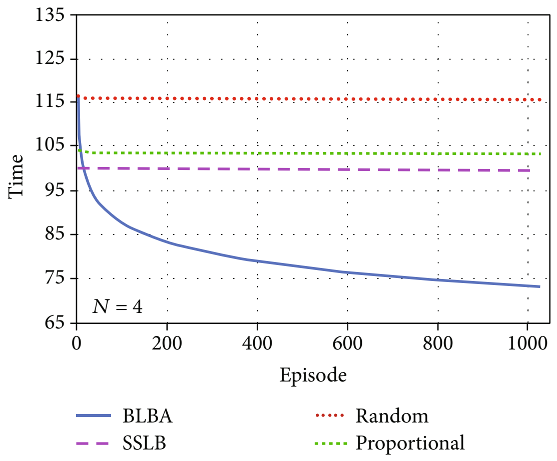

Figure 4: Average processing time.

reducing cumulative rewards. Because less processing capacity is allocated to each task, as a result, processing each of them has more delay. As shown in this figure, the assigned decision of the task based on the Double-Q-learning algorithm, with the suitable distribution of tasks between nodes, has gradually increased the cumulative reward.

It is expected that by using the Double-Q-learning algorithm, the load-balancing performance in the network will be significantly improved. As shown in Figure 4, the SSLB method has partly reduced the average processing time than the random and proportional methods. However, the assignment of tasks based on the Double-Q-learning algorithm, with the proper distribution of tasks among FNs, has

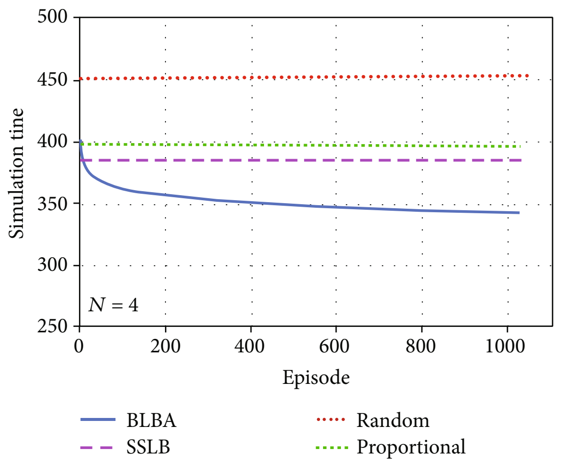

FIGURE 5: The run time of all tasks.

enabled the nodes to process the tasks faster, and as can be seen, the average processing time in the BLBA than compared methods has improved dramatically.

In Figure 5, the run time of all the tasks that enter the system during the simulation time is compared in the BLBA with other methods. As shown in this figure, the proposed BLBA compared to other methods has significantly reduced the run time of these tasks. This makes the proposed system perform better than other compared methods.

Then, in Figure 6, the total delay in all four methods is reviewed. Total delay is obtained through the average processing time of input tasks from the first to the last iteration of the algorithm implementation. As can be seen, the SSLB method has a lower total delay than the random and proportional methods. However, the proposed BLBA achieves less total delay than the other three methods, which means that the proposed method works better.

Finally, the standard deviation of load on nodes is compared in all four methods. According to Figure 7, at first, in the SSLB method, the standard deviation of load on nodes is less than in other methods. However, with increasing agent learning, the standard deviation of load on nodes in the proposed BLBA is significantly reduced. This indicates that in the proposed BLBA, the tasks are evenly distributed in the network, and the overload and underload possibility in the nodes is reduced. In addition, in this paper, the aim is to improve the load-balancing and reduce the delay, which in the proposed method both the delay and the load-balancing are optimized. In other methods, load-balancing may be improved, but that does not mean that delay is minimized.

Then, we consider the number of nodes as 10 and check the performance of these algorithms in these conditions. As shown in Figure 8, as the number of FNs and mobile devices increases, the proposed algorithm spends more time learning. However, even in this situation, the assignment of tasks based on the Double-Q-learning algorithm, with proper distribution of tasks among the nodes, has enabled the nodes to process tasks faster, and as can be seen, the average processing time of tasks in the proposed method is still improved over other methods.

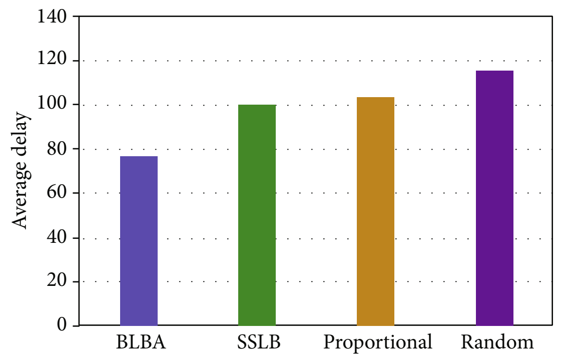

FIGURE 6: Total delay.

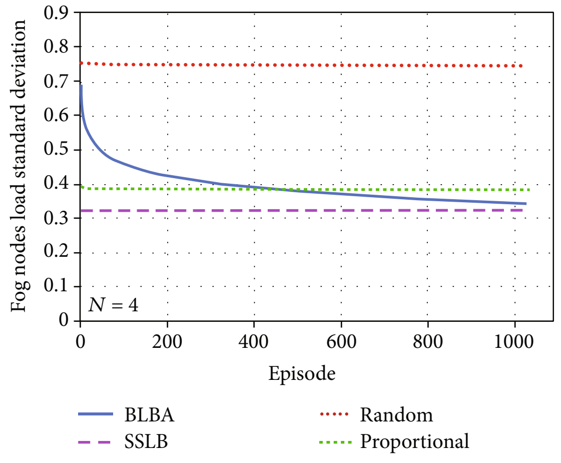

FIGURE 7: The standard deviation of load on nodes.

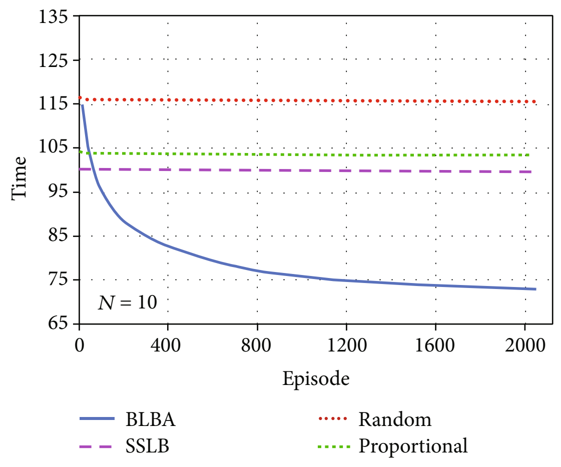

FIGURE 8: Average processing time.

In Figure 9, we can see that as the number of nodes increases, although the BLBA algorithm spends more time learning, finally the run time of all tasks that enter the system during the simulation time, the load-balancing method based on the Double-Q-learning algorithm in these conditions is also significantly reduced compared to other

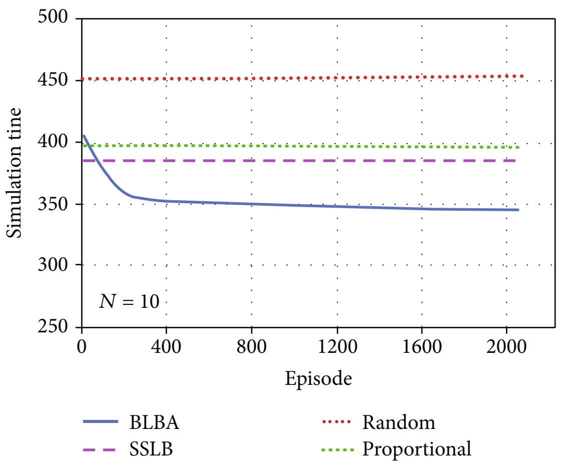

Figure 9: The run time of all tasks.

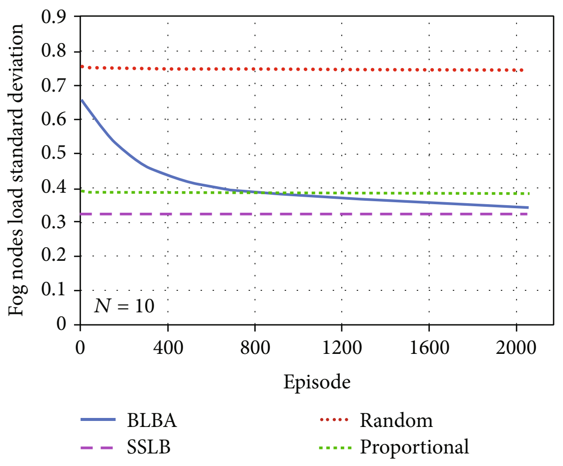

Figure 10: The standard deviation of load on nodes.

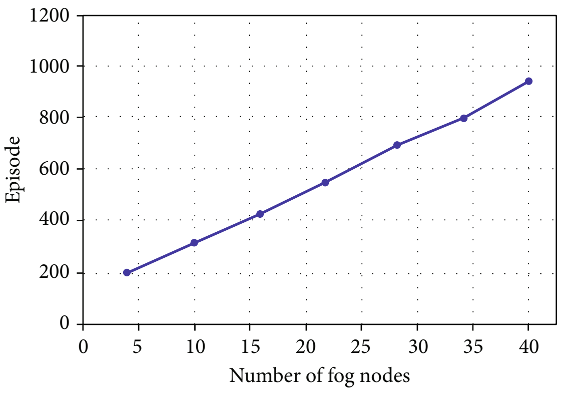

Figure 11: Approximate number of iterations required for convergence of the proposed algorithm.

methods. Then, the standard deviation of load on nodes in all four methods is compared for a situation where the number of nodes is 10. According to Figure 10, with increasing agent learning, the standard deviation of load on nodes in the proposed method is significantly reduced. This indicates that in the proposed method, the tasks are evenly distributed among the nodes.

Finally, Figure 11 shows the approximate number of iterations for convergence to the optimal run time per number of different nodes. As you can see, as the number of FNs increases, the number of possible actions increases, and thus, the response space becomes larger. Therefore, the time required to learn and converge to optimal policy also increases. However, although the number of nodes increases, in all implementations, the proposed algorithm eventually converges to the minimum point.

The results show that when a node decides to assign the load using the Double-Q-learning algorithm, it only considers the forward queue state, capacity, and the number of mobile devices connected to itself, and no information from other nodes. From the above evaluation, we conclude that the proposed BLBA is more stable than other load-balancing methods and significantly reduces network delay and response time. In addition, as the number of FNs increases, although the nodes spend more time learning, the results of the proposed method are better than the other methods over time, and we can be sure that the algorithm works well in any situation.

6. Conclusion and Future Work

The purpose of this paper is to provide a method to improve load-balancing in FNs. In this paper, the Double-Q-learning algorithm is used for load-balancing in the fog environment. The Double-Q-learning algorithm achieves an optimal policy using the experience that the agent gains from interacting with the environment. In the proposed BLBA, each FN as the agent explores the fog environment and seeks to find a low-load node for assigning tasks, to minimize processing time and the overload possibility. In this paper, the system state is defined only based on the state of the decision-maker FN, and the decision-making is done without any knowledge of the state of neighbor FNs. The BLBA has been tested for a different number of FNs and mobile devices within the network and has had good efficiency. The simulation results show that our proposed method significantly reduces processing time and response time than existing methods. According to the network structure, the utilization of the Double-Q-learning algorithm in any IoT device to further improve load-balancing and reduce delay is one of the future research directions that this paper opens for researchers. In addition, in the future, we intend to examine the performance of the proposed load-balancing algorithm in mobile edge computing.

Data Availability

The data used to support the findings of this article can be accessed by request.

Disclosure

A preliminary version of this manuscript has been published in the proceeding of 2021 11th International Conference on Computer and Knowledge Engineering (ICCKE), https://ieeexplore.ieee.org/document/9721449.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

(i) Niloofar Tahmasebi Pouya worked on validation, formal analysis, software, data curation, and writing—original draft. (ii) Seyedakbar Mostafavi worked on methodology, proofing—original draft, and project administration. (iii) Mehdi Agha Sarram worked on review and editing, data curation, and supervision.

References

[1] D. Baburao, T. Pavankumar, and C. S. R. Prabhu, "A novel application framework for resource optimization, service migration, and load balancing in fog computing environment," *Nano*, pp. 1–14, 2022.

[2] P. H. Vilela, J. J. Rodrigues, R. D. Righi, S. Kozlov, and V. F. Rodrigues, "Looking at fog computing for E-health through the lens of deployment challenges and applications," *Sensors*, vol. 20, no. 9, p. 2553, 2020.

[3] R. Beraldi, C. Canali, R. Lancellotti, and G. P. Mattia, "Distributed load balancing for heterogeneous fog computing infrastructures in smart cities," *Pervasive and Mobile Computing*, vol. 67, article 101221, 2020.

[4] Z. Duan, C. Tian, N. Zhang et al., "A novel load balancing scheme for mobile edge computing," *Journal of Systems and Software*, vol. 186, article 111195, 2022.

[5] M. H. Kashani and E. Mahdipour, "Load balancing algorithms in fog computing: a systematic review," *IEEE Transactions on Services Computing*, 2022.

[6] A. R. Hameed, S. ul Islam, I. Ahmad, and K. Munir, "Energy- and performance-aware load-balancing in vehicular fog computing," *Sustainable Computing: Informatics and Systems*, vol. 30, article 100454, 2021.

[7] M. Kaur and R. Aron, "A systematic study of load balancing approaches in the fog computing environment," *The Journal of Supercomputing*, vol. 77, no. 8, pp. 9202–9247, 2021.

[8] T. Hellemans and B. Van Houdt, "Performance analysis of load balancing policies with memory," *Performance Evaluation*, vol. 153, article 102259, 2022.

[9] D. Puthal, R. Ranjan, A. Nanda, P. Nanda, P. P. Jayaraman, and A. Y. Zomaya, "Secure authentication and load balancing of distributed edge datacenters," *Journal of Parallel and Distributed Computing*, vol. 124, pp. 60–69, 2019.

[10] J. Y. Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel, "Managing fog networks using reinforcement learning based load balancing algorithm," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, Marrakesh, Morocco, 2019.

[11] A. Marin, S. Balsamo, and J. M. Fourneau, "LB-networks: a model for dynamic load balancing in queueing networks," *Performance Evaluation*, vol. 115, pp. 38–53, 2017.

[12] N. Sonenberg, G. Kielanski, and B. Van Houdt, "Performance analysis of work stealing in large-scale multithreaded computing," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 6, no. 2, pp. 1–28, 2021.

[13] N. Khattar, J. Sidhu, and J. Singh, "Toward energy-efficient cloud computing: a survey of dynamic power management and heuristics-based optimization techniques," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 4750–4810, 2019.

[14] M. Adhikari and T. Amgoth, "Heuristic-based load-balancing algorithm for IaaS cloud," *Future Generation Computer Systems*, vol. 81, pp. 156–165, 2018.

[15] S. Rasheed, N. Javaid, S. Rehman, K. Hassan, F. Zafar, and M. Naeem, "A cloud-fog based smart grid model using max-min scheduling algorithm for efficient resource allocation," in *International Conference on Network-Based Information Systems*, Cham, 2019.

[16] L. Abualigah, A. H. Gandomi, M. A. Elaziz et al., "Advances in meta-heuristic optimization algorithms in big data text clustering," *Electronics*, vol. 10, no. 2, p. 101, 2021.

[17] M. Adhikari, S. Nandy, and T. Amgoth, "Meta heuristic-based task deployment mechanism for load balancing in IaaS cloud," *Journal of Network and Computer Applications*, vol. 128, pp. 64–77, 2019.

[18] S. T. Milan, L. Rajabion, H. Ranjbar, and N. J. Navimipour, "Nature inspired meta-heuristic algorithms for solving the load-balancing problem in cloud environments," *Computers & Operations Research*, vol. 110, pp. 159–187, 2019.

[19] S. Sefati, M. Mousavinasab, and R. Zareh 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, 2022.

[20] B. Kruekaew and W. Kimpan, "Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning," *IEEE Access*, vol. 10, pp. 17803–17818, 2022.

[21] S. R. Deshmukh, S. K. Yadav, and D. N. Kyatanvar, "Load balancing in cloud environs: optimal task scheduling via hybrid algorithm," *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 12, no. 2, p. 2150008, 2021.

[22] M. M. Golchi, S. Saraeian, and M. Heydari, "A hybrid of firefly and improved particle swarm optimization algorithms for load balancing in cloud environments: performance evaluation," *Computer Networks*, vol. 162, article 106860, 2019.

[23] A. M. Manasrah and H. B. Ali, "Workflow scheduling using hybrid GA-PSO algorithm in cloud computing," *Wireless Communications and Mobile Computing*, vol. 2018, 16 pages, 2018.

[24] A. Thakur and M. S. Goraya, "RAFL: a hybrid metaheuristic based resource allocation framework for load balancing in cloud computing environment," *Simulation Modelling Practice and Theory*, vol. 116, article 102485, 2022.

[25] X. Xu, S. Fu, Q. Cai et al., "Dynamic resource allocation for load balancing in fog environment," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 6421607, 15 pages, 2018.

[26] S. Moon, J. Park, and Y. Lim, "Task migration based on reinforcement learning in vehicular edge computing," *Wireless Communications and Mobile Computing*, vol. 2021, 10 pages, 2021.

[27] Z. Wu, J. Wei, F. Zhang, W. Guo, and G. Xie, "MDLB: a metadata dynamic load balancing mechanism based on reinforcement learning," *Journal of Zhejiang University Science C*, vol. 21, no. 7, pp. 1034–1046, 2020.

[28] M. M. Razaq, S. Rahim, B. Tak, and L. Peng, "Fragmented task scheduling for load-balanced fog computing based on Q-learning," *Wireless Communications and Mobile Computing*, vol. 2022, 9 pages, 2022.

[29] J. Xu, H. Guo, H. W. Shen, M. Raj, S. W. Wurster, and T. Peterka, "Reinforcement learning for load-balanced parallel particle tracing," *IEEE Transactions on Visualization and Computer Graphics*, vol. PP, p. 1, 2022.

[30] L. Mai, N. N. Dao, and M. Park, "Real-time task assignment approach leveraging reinforcement learning with evolution strategies for long-term latency minimization in fog computing," *Sensors*, vol. 18, no. 9, p. 2830, 2018.

[31] F. M. Talaat, M. S. Saraya, A. I. Saleh, H. A. Ali, and S. H. Ali, "A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment," *Journal of Ambient Intelligence and Humanized Computing*, vol. 11, no. 11, pp. 4951–4966, 2020.

[32] V. Divya and R. L. Sri, "ReTra: reinforcement based traffic load balancer in fog based network," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, Kanpur, India, 2019.

[33] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 102, pp. 847–861, 2020.

[34] Z. Li, C. Wang, and C. J. Jiang, "User association for load balancing in vehicular networks: an online reinforcement learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 8, pp. 2217–2228, 2017.

[35] Q. Lin, Z. Gong, Q. Wang, and J. Li, "RILNET: a reinforcement learning based load balancing approach for datacenter networks," in *International Conference on Machine Learning for Networking*, pp. 44–55, Cham, 2019.

[36] X. Li, Y. Qin, H. Zhou, D. Chen, S. Yang, and Z. Zhang, "An intelligent adaptive algorithm for servers balancing and tasks scheduling over mobile fog computing networks," *Wireless Communications and Mobile Computing*, vol. 2020, 16 pages, 2020.

[37] N. Rikhtegar, O. Bushehrian, and M. Keshtgari, "DeepRLB: a deep reinforcement learning-based load balancing in data center networks," *International Journal of Communication Systems*, vol. 34, no. 15, 2021.

[38] H. Y. Kim and J. Kim, "A load balancing scheme for gaming server applying reinforcement learning in IoT," *Computer Science and Information Systems*, vol. 17, no. 3, pp. 891–906, 2020.

[39] H. Ye, L. Liang, G. Y. Li, J. Kim, L. Lu, and M. Wu, "Machine learning for vehicular networks: recent advances and application examples," *IEEE Vehicular Technology Magazine*, vol. 13, no. 2, pp. 94–101, 2018.

[40] A. Mebrek, M. Esseghir, and L. Merghem-Boulahia, "Energy-efficient solution based on reinforcement learning approach in fog networks," in *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Tangier, Morocco, 2019.

[41] N. C. Luong, D. T. Hoang, S. Gong et al., "Applications of deep reinforcement learning in communications and networking: a survey," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[42] Y. Xu, W. Xu, Z. Wang, J. Lin, and S. Cui, "Load balancing for ultradense networks: a deep reinforcement learning-based approach," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 9399–9412, 2019.

[43] R. Mahmud and R. Buyya, "Modeling and simulation of fog and edge computing environments using iFogSim toolkit," *Fog and edge computing: Principles and paradigms*, pp. 433–465, 2019.

[44] I. Tellioglu and H. A. Mantar, "A proportional load balancing for wireless sensor networks," in *2009 Third International Conference on Sensor Technologies and Applications*, pp. 514–519, Athens, Greece, 2009.