Improving Load Balance in Fog Nodes by Reinforcement Learning Algorithm

Hongwei DING, Ying ZHANG*
Hebei Software Institute, Hebei, Baoding 071000, China

***Abstract*—Fog computing is a distributed computing concept that brings cloud services out to the network's edge. Real-time user queries and data streams are processed by cloud nodes. Tasks should be evenly divided among fog nodes in order to maximize speed and efficiency, optimize resource efficiency, and reaction time. Real-time user requests and data flow processing are done by cloud nodes. Nodes in a network must share responsibilities in a balanced manner in order to maximize speed and efficiency, resource efficiency, and reaction time, hence in this article, a novel approach is presented. When it comes to fog computing, load balancing essential suggested to be improved. According to the suggested algorithm, a task submitted to the fog node via a mobile device would be processed by the fog node using reinforcement learning before being passed on to another fog node. Neighbor or let the cloud handle it. According to the simulation findings, the suggested algorithm has achieved a reduced execution time than other compared approaches by properly allocating the work among the nodes. Consequently, the suggested technique has reduced the chance of incorrect job assignment by 24.02% and the response time to the user by 31.60% when compared to similar methods.**

***Keywords*—Fog computing; resource allocation; reinforcement learning; delay; load balancing; fog nodes**

I. Introduction

Load balancing is a fundamental concept utilized in cloud settings for allocating computing resources among servers and devices [1]. By balancing the use of hardware, network, and software resources, load balancing aims to maximize system performance, increase efficiency, and provide the best possible user experience [2]. Reinforcement learning algorithms are an artificial intelligence approach to load balancing that provides automatic and adaptive performance enhancement [3]. IoT devices usually assign task processing to the nearest fog node. In this case, it's possible that certain fog nodes take on more tasks than others and eventually become overburdened. In order to avoid this, load balancing techniques are used to spread tasks among fog nodes equitably [4]. Fog nodes are distributed throughout the environment [5]. Two distinct forms of load balancing are utilized in dispersed environments: static load balancing and dynamic load balancing. When choosing a load balancer, static load balancing ignores the target fog node's state [6]. On the other hand, dynamic load balancing chooses which fog node to route traffic based on its present status [7]. Dynamic load balancing is applied in this post. Load balancing among fog nodes reduces expenses, latency, and user response times while simultaneously enhancing resource productivity, efficiency, resource conservation, and real-time event detection [8]. Numerous load balancing techniques have been presented recently, and they are all effective in certain system situations. Meta-heuristic or hybrid load balancing algorithms may be taken into consideration, depending on the approach used [9]. There are two types of heuristic algorithms: static and dynamic. Initiatives entail limitations intended to determine the best course of action for a certain problem [10]. These algorithms have an advantage over meta-heuristic algorithms in that they are easily implemented and yield good results. Meta-heuristics require finality because of the enormous immensity of their solution space and the fact that they are completely random processes [11]. The type of problem, how it was initially set up, and the strategy employed to find a solution all have a big impact on how long it takes to resolve. In terms of execution time and cost, coupled algorithms—which are produced by combining numerous meta-heuristic heuristic algorithms—are more efficient than other algorithms [12]. Through the use of reinforcement learning techniques, the fog system can automatically and dynamically adapt to changes in compute load and service requirements [13]. Based on an assessment of the present status of the system and clients, the fog system can determine whether to add or subtract computing resources from a server, shift load from busy servers to freer servers, or assign resources to services and requests according to their importance. The system will be able to use the greatest computing resources and adapt dynamically to the different needs of users and services by employing this technique, which will significantly improve load balancing in fog computing.

Due to the distributed nature and dynamics of fog computing, however, conventional load balancing techniques become less effective, necessitating the development of an algorithm that can change with the context through time. To achieve this goal, a decision-making procedure based on reinforcement learning is suggested in this article to locate sparse fog nodes [14].

The agent chooses the right fog node based on the experiences it obtains from the environment in each scenario, which makes the proposed technique, the delay may be greatly reduced. Additional and time-consuming calculations are also removed in the proposed method of this article. Reinforcement learning for load balancing is superior to conventional approaches in that it not only simplifies the algorithm framework without taking any network model assumptions into account, but also converges to the best policy in polynomial time. The collected findings demonstrate that, when compared to the compared approaches, the suggested

load balancing method greatly decreases the lag and reaction time to the consumer. This article's conclusion is structured so that it is discussed in the section of current related research in the area of load balancing in fog computing.

Section III explains the concept of the proposed system, the reinforcement learning technique, and how to find the system's delay. The suggested load balancing strategy is presented in Section IV. The evaluation and comparison of the simulation results with earlier techniques are done in Section V. Section VI will conclude with recommendations for additional research.

II. Related Work

In fog computing, jobs are typically assigned to the nearest fog node by mobile users and IoTs devices. These devices are frequently mobile, therefore depending on where they are in the network, various important nodes may have varying loads. Due to this problem, some fog nodes may be overburdened while others may be idle or underloaded in terms of the distribution of work. Methods to solve the load balancing issue in the fog computing environment have been presented by some authors. These actions can be divided into various categories [15].

Here, prior research on task delegation in which nodes need to be aware of each other's computational capabilities will be examined. To discover the best loading decision in the presence of an uncertain reward model and transition probability, [16]. According to the resource capacity, fog nodes in the presented process can assign an ideal number of incoming jobs to a free neighboring fog node. This is done to cut down on processing time and potential overhead [17]. They looked into load balancing on several kinds of computing nodes before officially presenting the fog computing system's structure. Then, they developed a matching resource allocation strategy for fog environments, which combines static resource allocation with dynamic service transfer, to accomplish load balancing in fog computing systems. The min-min method was developed by [18] to take network resources into account. When sending a job to a cluster node that is overloaded, factors such the distance between the cluster and the node next to it, the amount of tasks that are waiting in each cluster's queue, and the distance between the cluster node and the closest cloud data center are taken into account. Researchers suggested the min-min method in [19] and put it into practice inside each cluster while taking network resources into account. In this method, a neural network is used to evaluate the fog node's current capacity. The Internet of Things gadget transmits its work to the cloud if it doesn't get the necessary resource. In this study, a four-layer architecture for load balancing and task scheduling is proposed. The Internet of Things is a component of the top layer, where a lot of data is generated and sent at once. The jobs are divided into two categories important and less important in the second layer via a dual fuzzy logic method. The user-proximate nodes with the lowest load are given priority for task execution [20].

Other works are predicated on the knowledge of the node's load or the prediction of its future load. This algorithm continuously gathers network traffic, server load information, and control information. By merging fog computing and software-based networks, [21] devised a load balancing technique based on reinforcement learning. In order to offer the greatest amount of access to the resources, this algorithm analyzes the behavior of the network and divides up the work by considering network's current load and forecasting its future load. The network is adaptable thanks to this architecture's dispersed nature. In this article, a threshold limit is taken into consideration to implement the load balancing method, and if the server load exceeds 75%, the load balancing algorithm is called. In order to achieve better load balance, [22]. It also applies reinforcement learning to handle the task loading problem. In this study, Deep Q-Learning is enhanced using an LSTM network. The quantity of input data needed for the sub-task, the downlink bandwidth, the amount of output data generated by the sub-task, and the load on each server make up the state space in this article. The action space is a vector with m + 2 zero- and one-dimensional dimensions. A cloud server and a mobile device are included in the m and 2 edge servers. Any of the same folders can be used to download data to the server. The three variables of load balance, cost, and energy consumption are taken into account by the reward function. In study [23], Berardi and colleagues address the issue of resource management by presenting two distributed load balancing algorithms, Sequential Forwarding and Adaptive Forwarding, which are intended to handle heterogeneity. They do this by assigning jobs to nearby nodes. According to the threshold limit and the maximum number of steps, M, a task is delivered at random to nearby fog nodes using the first approach, known as the Sequential Forwarding method, until it reaches the correct node. The second method, known as the Adaptive Forwarding method, is suggested since it is difficult to define the working parameters and M. This method automatically and conditionally updates these parameters. For a fog computing environment, the research in [24] presented a load balancing method that works well for medical applications.

The techniques described in earlier studies demonstrate that the majority of these techniques require knowledge about the nodes' capacity or load in order to make decisions [25]. This effort necessitates a number of time-consuming computations that add latency and raise network traffic burden shall be. Additionally, in the majority of these approaches, the load balancing operation and load distribution are often performed by a single node. In contrast to other works, this one uses a different decision-making procedure because, according to the suggested method, the fog node decides on processing and task assignment only after gathering information from the delay and reward during the learning period and after taking into account its own capacity and the positions of other nodes [26]. The proposed solution is intended to address a subset of difficulties, although its use is not constrained to a particular scenario. Additionally, the strategy suggested in this article is dynamic and adapts to the circumstances of the agent's goals.

III. System Model

The description of the suggested system and the reinforcement learning algorithm will be covered first in this

part. The load balancing problem formulations are then provided using reinforcement learning.

A. System Description

The paper takes into account a four-layer design for the suggested system, which includes Internet of Things, fog nodes, proxy servers, and cloud data centers, as shown in Fig. 1. The Internet of Things layer, which contains various end devices including wireless sensor nodes, mobile devices, etc., is the initial layer in this system. These gadgets can transmit data to nearby fog nodes because they are directly connected to them. The fog layer, which is the second layer, is made up of extremely intelligent equipment like routers, switches, and gateways that take in and process data from endpoints [27]. The cloud data center layer, which consists of numerous computers and data centers, is the fourth tier. This structure eliminates the requirement for data transfers to the central cloud by allowing data and information processing to take place locally in fog nodes [28]. Due of their limited computational power, mobile devices in the proposed system assign fog nodes within their range to process a portion of the work (a virtual reality game). Since there is no master node or controller in this system that keeps track of the fog nodes' status and the external conditions, it is up to the fog nodes to collect data and make decisions [29], [30].

The proposed system operates as follows: On the user's smartphone, an Android application called Tractor Beam2EEG (a type of game where players compete against each other) is running. This application demonstrates how the human brain and the computer interact. Each player must have a headset attached to his smartphone in order to play this game. This application continuously monitors the signals picked up by the headset, the processing, and the mental state of the user. Processing the software can need a lot of processing power. Therefore, in this article, the program is broken into multiple pieces known as subtask 3 in order to enhance the processing time, transfer time, and boost the usage rate of network resources. The following dependencies, depending on virtual reality game tasks, are presented in this article. The software is divided into five subtasks, as illustrated in Fig. 2: EEG, Client, Actuator, Concentration-Calculator, and Connector. These subtasks' data are interdependent.

The major processing modules in this application are the Client, Concentration-Calculator, and Connector modules. In order to receive the EEG signals, the Client module interfaces with the sensor. Once it has received the signals, it checks their levels and, if they are constant, passes them to the Concentration module. Calculator that assesses the user's mental state based on the signal it receives and computes their level of concentration [31]. The Client module is then informed of the computed concentration level by the Concentration-Calculator module. The Connector module connects the game amongst several participants who may be present in geographically dispersed areas by operating on a global scale. The Client module of each connected user receives a constant stream of information from the Connector about the game's current condition [32]. Numerous modules can be kept on mobile devices due to the fact that these sub-tasks require less computational complexity and data transfer, while those that demand greater computing resources can be assigned to the cloud provider's nodes. The loop that transforms the user's mental state into the game's state on the mobile device's screen is the most crucial control loop in this application. The mobile device and the device that houses the user's brain state calculation module must communicate in real-time for this to work. The user experience is significantly impacted by latency in this loop because it affects the entities that the user interacts with directly [33].

The computing modules should be as near the data sources as possible to minimize the latency in data transmission between units. The EEG, Client, and Actuator modules of this article's suggested design, as illustrated in Fig. 3, are connected to mobile devices. Each module in the loop processes the program, forwarding the processed data to the subsequent module, and so on, until the Actuator module in the mobile device receives the program's final results [34]. Each fog node has an unpredictable amount of mobile devices connected to it at any one time given time due to the dynamics of the environment, and it is possible that some fog nodes acquire more subtasks than others and eventually become overloaded. To prevent this, fog nodes are evenly divided into sub-tasks using load balancing techniques.

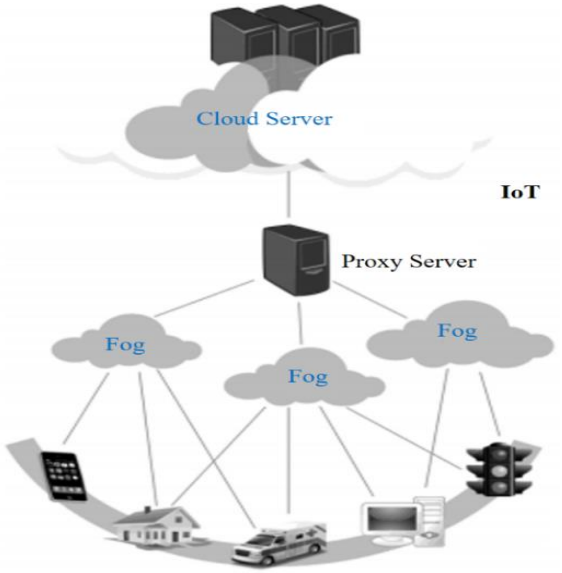

Fig. 1. Architecture of fog computing layers.

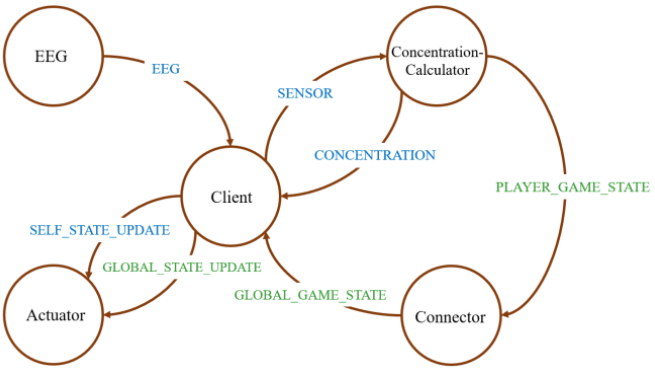

Fig. 2. Subtasks and their dependencies.

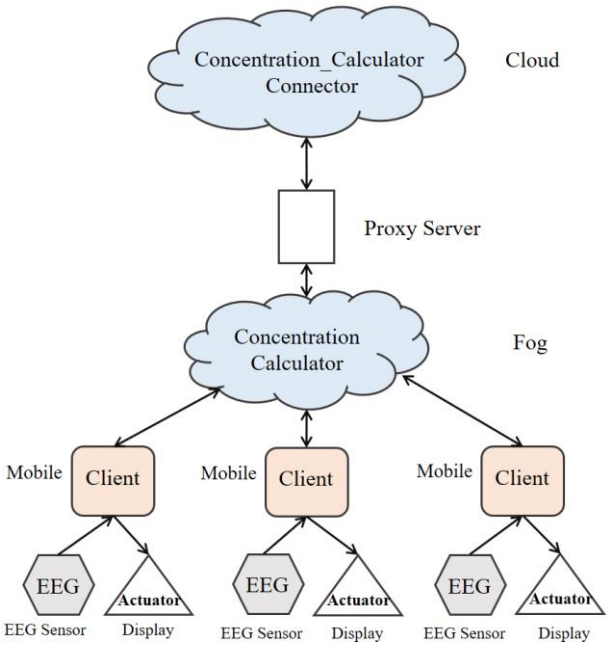

Fig. 3. Module placement in various devices.

The load balancing algorithm's primary goal in a fog computing setting is to increase the user reaction time by dispersing the system's total load so that it can continue to operate at its best under dynamic system settings. This article's goal is to reduce network resource loss, user response time, and delay by applying the reinforcement learning method to the fog nodes. When a fog node receives a subtask, it employs a reinforcement learning algorithm to decide whether to process it independently, pass it on to another node in the area, or send it to the cloud to be processed faster.

B. Reinforcement Learning Algorithm

Supervised, unsupervised, semi-supervised, and reinforcement learning are the four main types of machine learning algorithms. Numerous labeled input data are needed for supervised learning in order to train the system. Unsupervised learning, as contrast to supervised learning, involves learning from unlabeled events in order to uncover hidden patterns in the data. Unlabeled data and labeled data are used in supervised fog learning to increase learning accuracy [35]. The agent can learn the best actions from the environment through reinforcement learning. Through environment exploration, trial and error, and use of the incentives provided by the environment, this learning is accomplished [36]. In this article, load balancing is accomplished by Q-L algorithm. The proposed approach formulates the load balancing problem as a Markov decision process (MDP), where the fog node takes a decision after learning the state of its surroundings and is rewarded by it. Experience is the end result of trial and error and is characterized by the four states of the present state, action, reward, and next state [37].

1) Policy: The agent's behavior can take one of two forms when it comes to policy: active policy, in which the agent learns the value function in accordance with the performance that the current policy has caused, or passive policy, in which the agent learns in accordance with the action that the policy has caused [38]. The function learns the value, the other is obtained, and it is defined. The Q-Learning algorithm is a passive algorithm with a greedy learning strategy for the Q value.

2) Value function: The learner function's reward function sets the objective, and the closer it gets to the target, the more reward it receives. The value function in reinforcement learning, on the other hand, derives a value as Eq. (1) for each state and has a long-term perspective. The closer to the objective this value is, the higher it is.

$$v*(s) = max \sum_{s',r} p(s', r|s, a)[r + \gamma v*(s')] \quad (1)$$

The discount factor, often known as $0 < \gamma < 1$, in this context determines the significance of potential benefits in the future and the decision made right now is more important than those made in the future.

In states, the agent takes action (A) to change the environment from its present state to a new state (b) and receives a reward (r) for his efforts; both factors influence his future decision-making.

3) Model: The reinforcement learning problem has a random model with non-deterministic states. Going from one state to another and taking any action are both possibilities [39].

Reinforcement learning has thus found various uses in optimization in the dynamic, unpredictable, and changing environments of fog and clouds. Additionally, it might be a good way to evenly distribute loads among fog nodes [40].

C. Formulation of the Problem

Discrete time stochastic control is what the MDP is. One method for solving MDP is reinforcement learning, which in turn makes use of dynamic programming. To achieve the target performance, the suggested load balancing problem is expressed as an MDP. For the suggested load balancing problem, the fours $< S, A, P, R >$ are defined below, which are typically included in MDP:

1) The state space $S = \{s\,(C, Q, N)\}$ represents the relationship between the fog node's capacity (C), the size of its upstream queue (Q), and the number of mobile devices (N) linked to the fog node.

The Q-L algorithm bases decisions on the system's present state. Many of the earlier techniques for defining the state space call for knowledge of the nearby nodes' capacity [41]. However, the state of the system is solely specified in the proposed method based on the state of the decision-making fog node, which forces decisions to be taken without knowledge of the states of the surrounding nodes.

2) In the action space $A = \{a = (n)\}$, n represents the choice of a fog or cloud node to be assigned to the subtask.

3) P: A number between zero and one represents the transition probability. In order to be in state s, the criterion is to have the probability distribution of the transition $P(s'|s, a)$ to the next state s', with the action choice.

4) R: The state's activity is directly linked to the reward. The primary objective is to minimize processing delay and overhead probability while maximizing long-term value in each system by selecting the appropriate action.

As was already noted, there are various subtasks within the task (virtual reality game). The sum of all linked sub-tasks' transmission and processing delays across all relevant devices is the task execution delay and is determined as follows:

$$T_{task} = t_{uot} - t_{in} \quad (2)$$

where, $t_{out} and\ t_{in}$ represent, respectively, the entry time and exit time of a task in the suggested system. Since the processing and transmission delays in mobile devices are relatively constant, this article solely calculates the processing delays of the Concentration-Calculator subtask in fog or cloud nodes and the subtask in the mobile device to compute the delay of task execution [42]. It is seen as having a constant value. The suggested method uses fog nodes to perform the Q-Learning algorithm, Additionally, the subtask's processing delay is equal to the negative of the reward function (R $R(s,a)$), which is allocated to the fog node. The longer the processing delay of the subtask, the better. As a result, $R(s,a)$ will be lower than expected. The calculation looks like this.

$$R(s,a) = -T_{Subtask} \quad (3)$$

This refers to the calculation subtask that the fog node is tasked with performing, where $T_{Subtask}$ is its processing delay. The symbols used to evaluate the system and the delay calculation formulas are listed in Table I.

TABLE I. SYMBOLS USED TO EVALUATE THE SYSTEM AND DELAY CALCULATION FORMULA

Parameters	Description	Value
W	How many smaller jobs ran on the node	-
L	Subtask data size	3500
B	Bandwidth per node	10000
$d_{i,j}$	How far apart are the nodes i and j?	-
β_1	Path loss constant between two nodes	10^{-3}
β_2	path loss power	5
P	Power transfer between nodes	20 dBm
N_0	Spectral density of noise power	175 dBm /Hz
I	Quantity of subtask-specific commands	200×10^6
Cycle	The amount of CPU cycles used by each instruction	5
f	Fog node cpu speed, cloud cpu speed	2800 44800
N	Just how many fog nodes	4
n	The frequency with which a state is displayed	-
α	Rate of learning	2/n
y	reduction element	0.8

The subtask's processing lag depends on whether its processing is done in the fog the node that the mobile device sent the subtask to $(FN-I)$ or whether it is handed over to the neighboring fog node $(FN-J)$ or the cloud.

- The following formula determines the execution delay of the subtask, which is equal to Subtask if the subtask is processed by $FN-I$.

$$T_{Subtask} = \frac{I\times Cycle\times W}{f} \quad (4)$$

- If the subtask is delegated to $FN-I$ or Cloud for processing, $T_{Subtask}$ is calculated as follows.

$$T_{subtask} = t_{W_i} + t_{c_{ij}} + t_{E_j} + t_{W_j} + t_{c_{ji}} \quad (5)$$

In this regard, t_{W_i} and t_{W_j} are the waiting delay of the subtask in the I-FN sending queue t and the J-FN sending queue or cloud is affected by the subtask's waiting delay. $t_{c_{ij}}$ denotes the time it takes for the subtask to be transmitted over the communication channel. Moving from I-FN to J-FN or the cloud. The subtask execution delay in J-FN or the cloud is represented by t_{E_j}, whereas the subtask result delay from J-FN or the cloud to I-FN is represented by $t_{c_{ij}}$. A node's (i) or cloud node's (j) waiting delay in the sending queue is determined in the following way.

$$t_w = t_o + t_i \quad (6)$$

where, t_o and t_i are respectively the entry Add the arrival time of subtask m to the node's queue and record the departure time of subtask m from the same queue. The latency of subtask transmission on the communication route between FN-I and FN-J or the cloud, and vice versa, is equivalent to:

$$t_c = \frac{L}{r_{i,j}} \quad (7)$$

where, $r_{i,j}$ The transmission rate between nodes i and j is denoted as.

$$r_{i,j} = Blog(1 + \frac{g_{i,j}\times p}{B\times N}) \quad (8)$$

where, $g_{i,j} = \beta_1 d_{i,j}^{-\beta_2}$ is the gain of the channel between two nodes i and j. The delay of execution of the subtask in J-FN or cloud t_{E_j} is equal to:

$$t_{E_j} = \frac{I \times Cycle \times W}{f} \quad (9)$$

Each fog node is a learning agent operating in an S-state space environment. Each time a new task is added to the system, the agent performs an action in the surrounding area and chooses one of the nodes to receive the new work. In the event that the environment state is updated, the reward for this allocation will be decided. The agent will receive the reward if the system is now operating with a load balance that is closer to ideal and the subtasks are processed more quickly than they would otherwise [43]. Each node progressively acquires the ability to optimize its decision-making process for handling subtasks based on the rewards it receives. Minimizing the delay of sub-task processing in the fog environment will reduce the overall delay and response time to the user, resulting in the network spending less time on task processing.

IV. The Proposed Load Balancing Method

In order to address the issues with the prior approaches, the load balancing algorithm based on reinforcement learning is presented in this section. It is designed to distribute the load uniformly among the intermediate nodes of the fog. Dynamic programming can solve MDP when the system for every state-action combination has a transition function and a reward function, but typically the system is unable to anticipate the precise value of the transition function and reward for the majority of the states, which is required to solve the problem. It is suggested to use reinforcement learning to solve these issues. The Q-Learning algorithm, one of the reinforcement learning algorithms, is utilized in this article to locate the ideal action mode with the least amount of computing expense, making up for the absence of appropriate data through experimentation. The Q-Learning algorithm's model is a random model.

Distributes an agent involved in network learning is referred to as a fog node. The sub-fog node will choose to use reinforcement learning to process a new task after receiving it. Hence, the fog node designated for the Concentration-Calculator task acquires data from the mobile device. The fog node then assesses the environment and, in order to maximize the long-term reward, decides whether to complete the subtask independently or to delegate it to a nearby fog node for quicker completion based on its capacity and past experiences and rewards. If the Concentration-Calculator subtask takes longer to process in the fog nodes than it does in the cloud, the fog node chooses to transfer this subtask's processing to the cloud, which will speed up processing and lighten the load on the fog nodes. According to the system model, each fog node performs a response after observing the current states, a delay is made, and the new state s' is observed, and for that, it receives a reward (R, s, c) from the environment.

$$Q_{new}(s,a) = Q_{old}(s,a) + a[R(s,a) + \gamma \max_a Q(s',a') - Q_{old}(s,a)] \quad (10)$$

The learning rate, $0\ 1 < \alpha < 1$, strikes a balance between previously learned material and new observations. Using the available knowledge, the greedy method selects a course of action that yields the greatest reward in a single step. The Learning-Q method uses the likelihood of selecting an action ε, where ε −greedy is the policy with larger reward, in order to maximize the long-term value. The fog node is chosen to complete the subtask in order to maximize long-term value. In this manner, a random action with a fixed probability ε −greedy is chosen in each time step of the algorithm $0 \leq \varepsilon \leq 1$.

The benefit of utilizing 1−ε and the action with the highest value is that as the number of steps rises, every ε −greedy of the Q(s,a) algorithm exhibits an infinite action, ensuring that it will eventually converge to the best value. As a result, using the Learning-Q method, the fog node learns to choose the best node for handling the subtask. The suggested method calculates the suitable reward function using 3.

Since the function of infinity has been observed, it is certain that (Q, s, and a) will eventually converge to the ideal value. Consequently, using the Q-Learning method, the fog node learns to choose the best node for processing subtasks. The load balancing solution that has been suggested calculation of the suitable reward function is shown in Fig. 4 and it is done using Eq. (3). The fog node learns the complete network and the likelihood of loading to each node as it traverses the network using the learning method, enabling it to select the best node to transfer the task to. The algorithm begins by using a greedy approach to explore the network, and once it has a thorough understanding of the network's requirements, it performs optimal load balancing. The state space encompasses the capacity of the fog node, the length of the uplink queue within the fog node, and the quantity of mobile devices linked to the fog node. In order to minimize the processing delay for the subtask, select either a reward eyebrow or a fog node.

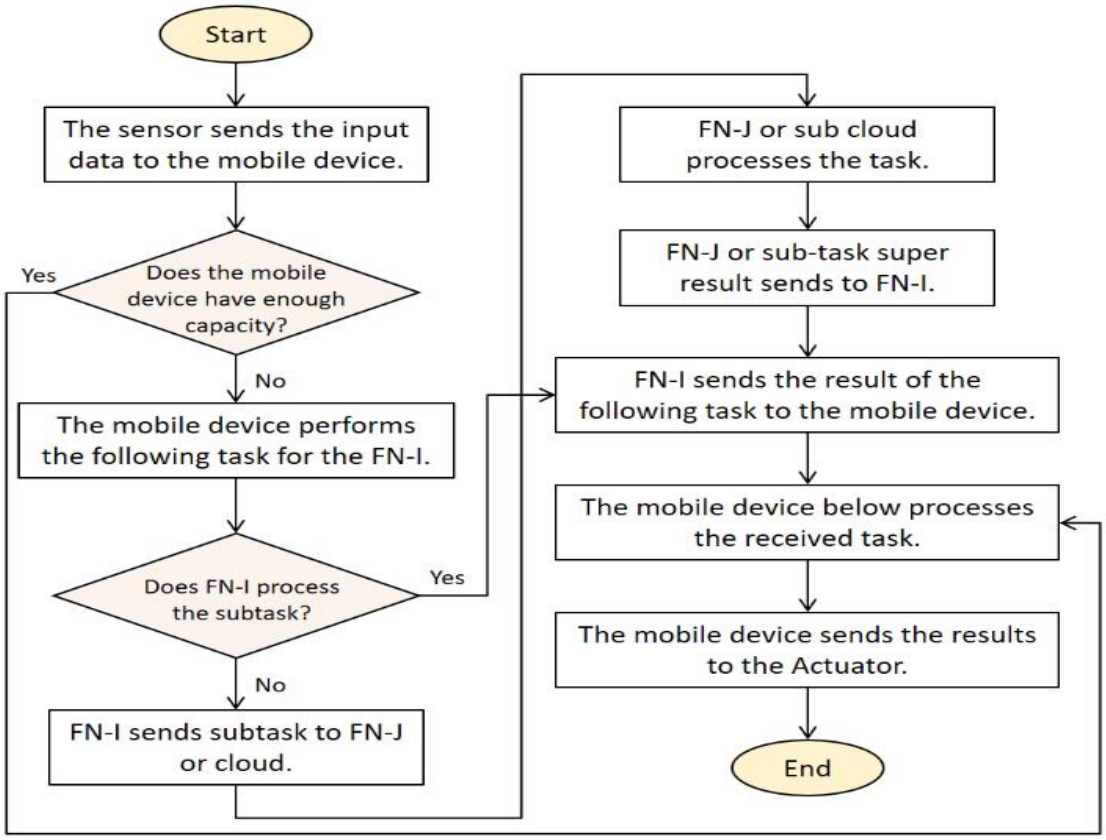

Fig. 4. Block diagram of the implementation steps of the proposed load balancing method.

V. Evaluation and Simulation

This article utilizes the iFogSim simulator [21] to model the load balancing problem using the reinforcement learning technique. This software was executed on a Sous computer equipped with seven Intel Core i processors and eight GB of RAM. The proposed system has N fog nodes and a variable number of mobile devices that establish random connections with neighboring fog nodes to broadcast subtasks. This paper proposes the utilization of the Q-Learning algorithm for load balancing in a fog environment. Initially, the fog node lacks any understanding of the network due to the fact that all the Q-table values in the Q-Learning algorithm are set to zero. The application of the avaricious approach to learning is implemented. Initially, the algorithm conducts an exhaustive search of the network, prioritizing immediate gains, as the number 0 is considered equivalent to 1. Over time, as the fog node's estimation reliability in the Q-table increased, its value changed to 0.3. Additionally, the received reward's value is equal to the concentration-calculator subtask's negative processing delay at the fog or cloud node.

The learning method is applied right away to minimize the generation of overhead in the nodes as it is expected that the task creation and task sending have already been completed in the simulation. The Q-L method allows the fog node to learn the best ways to interact with its surroundings. Through environment exploration, trial and error, and use of the incentives provided by the environment, this learning is accomplished. In general, each fog node assesses the state of the environment, selects a node to assign the work to, and then reaps the benefits. With each iteration of the method, the fog node's network experience grows, and over time it learns to assign the sub-task to a node with a lighter workload and faster processing speed. Contrary to the proposed way, alternative solutions (such as those in sources [6, 8]) conduct the load balancing algorithm before adding overhead to the fog node, which degrades the performance of the aforementioned systems and lengthens their latency. Another benefit is that, in the comparative methods for allocating sub-tasks to surrounding nodes, it is only necessary to be aware of this node's position and capacity, which may be determined by making a few numbers of laborious computations. However, in the suggested system of these computations, time-consuming and redundant tasks are eliminated, and the fog node simply behaves in accordance with the knowledge it has received from its surroundings. By doing this, the proposed system's latency will be as little as possible.

A number of current load balancing techniques have been compared to the performance of the suggested method, and in this simulation, random and proportional SALB load balancing techniques have been employed as benchmarks [6, 8]. The SALB approach examines the adjacent nodes' capacities after the fog node is overloaded and delivers the subtask to the node with the highest capacity and at least 40% of its capacity. Sub-tasks are distributed at random to fog nodes in the random technique. The Proportional approach receives information on each neighbor's capacity and chooses the best node based on the size of the subtask.

Following that, the graphs created by applying the Q-L algorithm to the load balancing problem are provided. Finally, the results of applying all four algorithms are discussed, along with a comparison of how well they performed in terms of delay, user response time, and load balancing.

Fig. 5 displays the progressive augmentation of the cumulative reward with each repetition of the proposed technique. As mentioned earlier, the reward is equivalent to the reciprocal of the processing delay for the subtask assigned to the fog node. The awarded reward will drop as the subtask's processing time increases, depending on whether the cloud provisioning node handles it. The action that yields the highest Q-value is chosen in the decision to transfer the load. According to the effectiveness of the suggested incentive, the proposed technique uses Q-Learning-based load assignment decision to reduce processing duration and overhead likelihood. With an increase in task processing rate, cumulative reward rises. Due to the fact that many tasks are queued up in the nodes, the cumulative reward also continuously falls as the quantity of incoming tasks rises. In this method, the network delay and user response time are decreased as the fog node eventually learns to assign processing of subtasks to the node that causes the least amount of delay.

Fig. 6 demonstrates how the average execution time has decreased dramatically as a result of program repetition and increased learning. In this method, the network delay and user response time are decreased as the fog node figures out which node is the least delay-prone and starts to delegate subtask execution to it. Furthermore, as seen in Fig. 7, the standard deviation of the nodes' load decreases as learning grows.

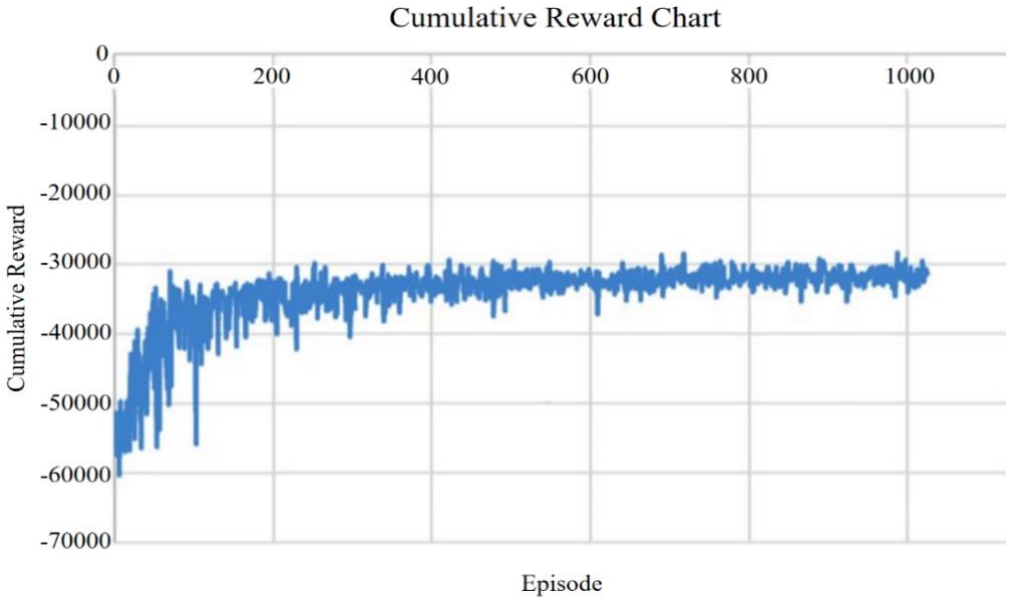

Fig. 5. Increasing payout with each cycle of the suggested method.

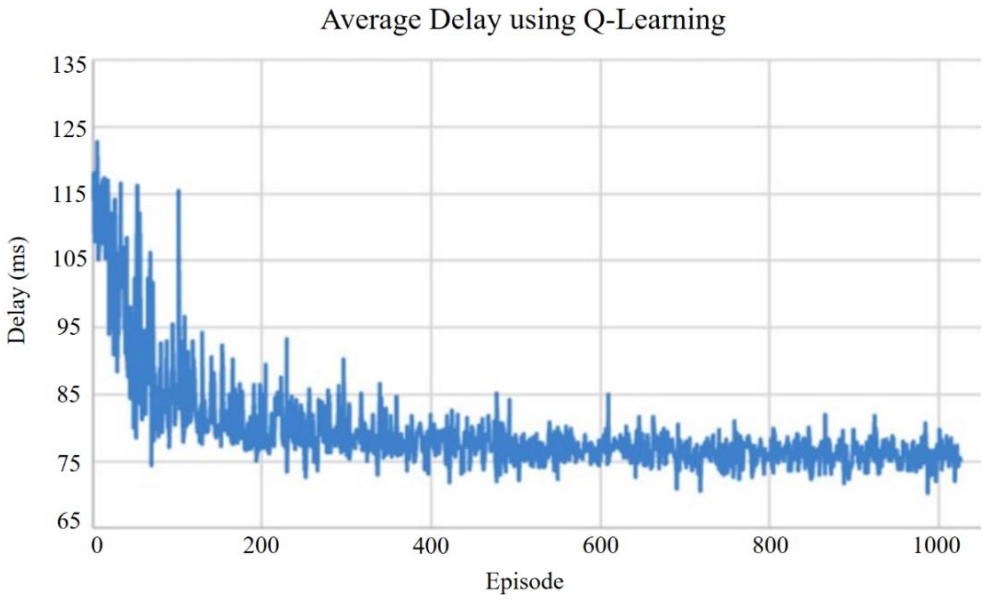

Fig. 6. Average task execution delay in Q-Learning method.

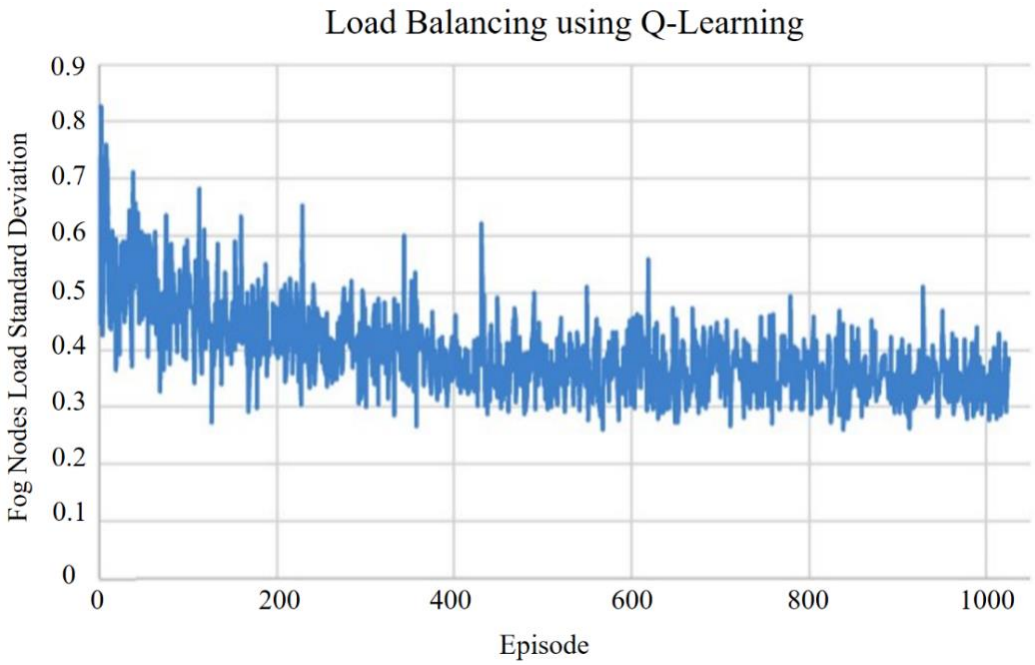

Fig. 7. Standard deviation of load on nodes in Q-Learning algorithm.

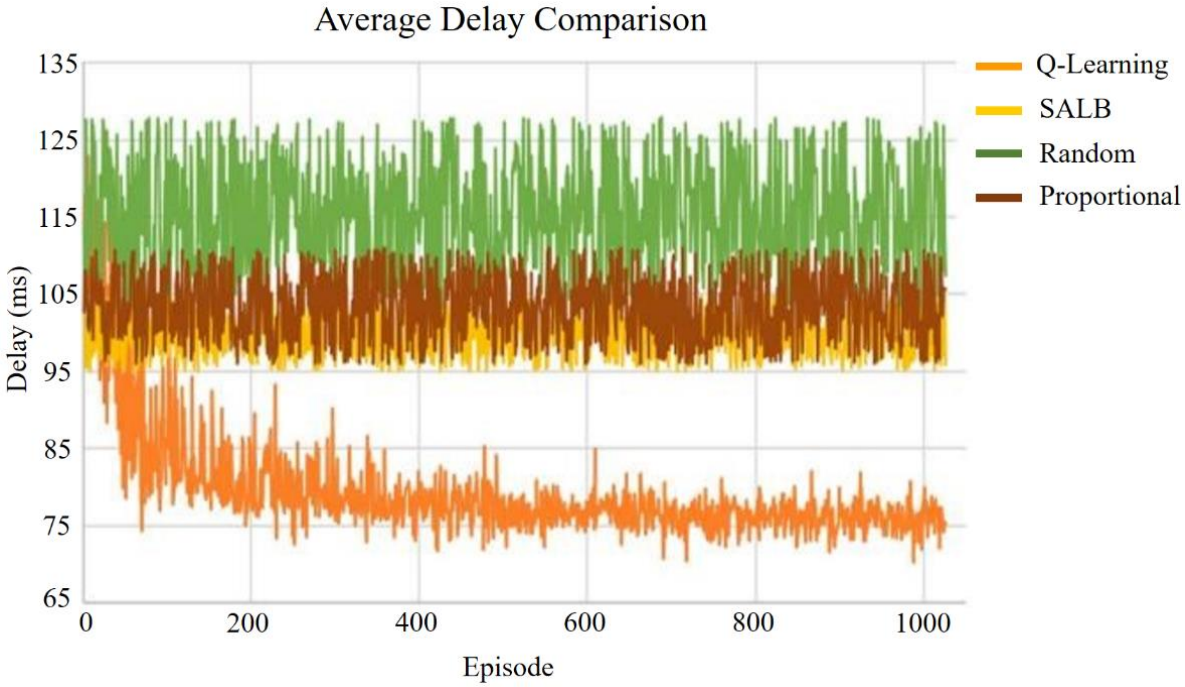

Fig. 8. Average task execution delay.

As a result, the network's job distribution and processing may be guaranteed to be balanced, and load balancing is enhanced. Just as mentioned before, the fog node detects which node causes the least delay and starts sending the subtask processing to that node. If the sub-task is sent to a node with more capacity and can therefore handle the incoming sub-task more quickly, the processing latency of the sub-task is decreased. By giving the subtask to the fog node with the greatest capacity, overhead with minimal strain on other nodes are avoided.

The pressure on the nodes' standard deviation diminishes as learning progresses. As a result, the load balance is enhanced and it is possible to guarantee that the tasks are dispersed and carried out in the network in a balanced manner. As previously noted, the fog node eventually learns to assign the subtask's processing to the node that causes the least delay. If a subtask is assigned to a node that has more capacity and can handle the incoming subtask more quickly, the processing latency is decreased. By giving the work to the fog node with the greatest capacity, overhead and underloading of other

nodes are avoided. The outcomes of the execution of all four algorithms are now reviewed, along with a performance comparison. It is anticipated that the Q-Learning algorithm will considerably enhance the network's load balancing capabilities. According to the evaluation's findings, Fig. 8 illustrates how choosing a task based on Q-Learning minimizes task execution latency in accordance with the suggested reward function. The proposed load balancing technique uses reinforcement learning to evenly spread the load across the nodes, enabling the nodes to complete sub-tasks faster.

The cumulative reward drops as the task arrival rate rises because fewer tasks can be processed by fog nodes due to the relatively high amount of subtasks that are queued at them. However, the suggested load balancing method produces a beneficial reward in that the delay is also reduced in the same proportion. This is because the load is distributed properly across the fog nodes. Additionally, unlike the approaches that were examined, no time-consuming computations were required in the suggested load balancing method to determine the capacity and location of surrounding nodes. As a result, as shown in this figure, the time it takes for the suggested approach to work is when contrasted with alternative methods, it is greatly decreased. When the standard deviation of the four methods for node load is compared, the average task execution delay is checked. Fig. 9 shows that the dispersion of the nodes' loads is initially lower in the SALB algorithm than in other approaches, but that it has dramatically decreased within the suggested approach as an agent learning has increased. This demonstrates that the suggested strategy evenly distributes the duties around the network, minimizing the likelihood of overhead in the nodes.

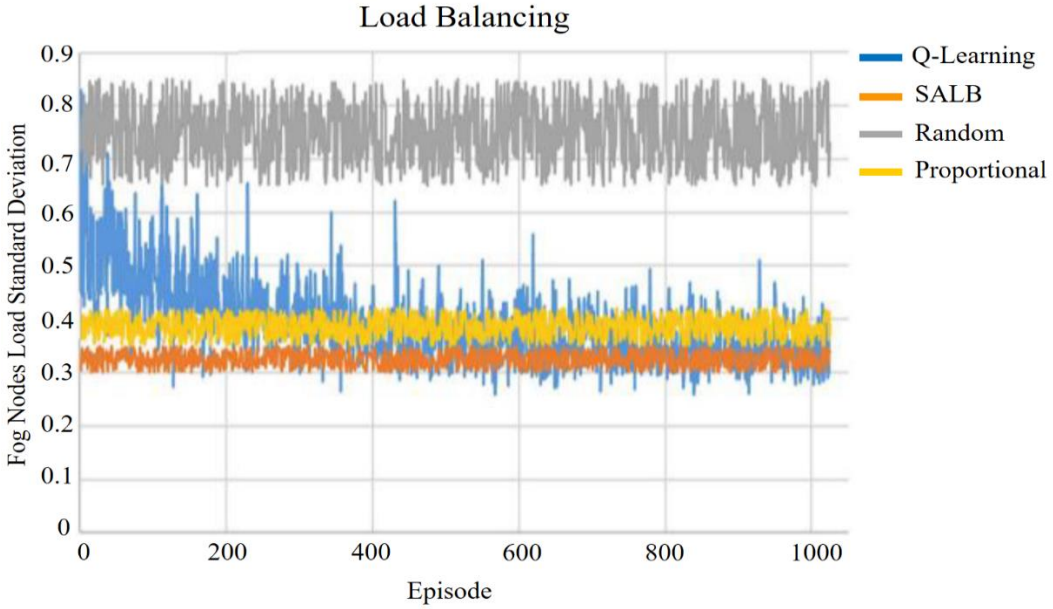

Fig. 9. Dispersion of node loads averaged out.

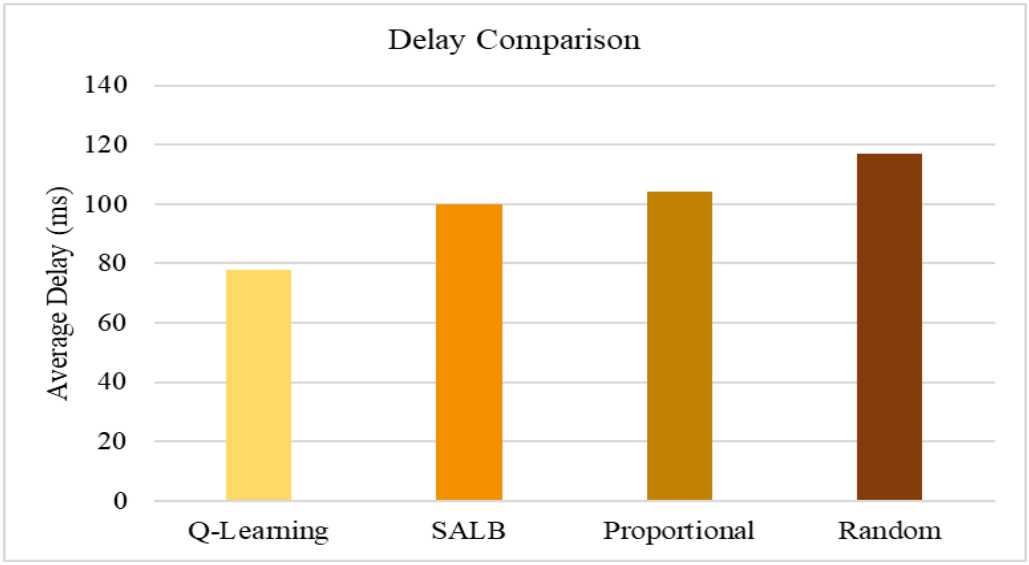

Fig. 10. Comparison of total delay for different methods.

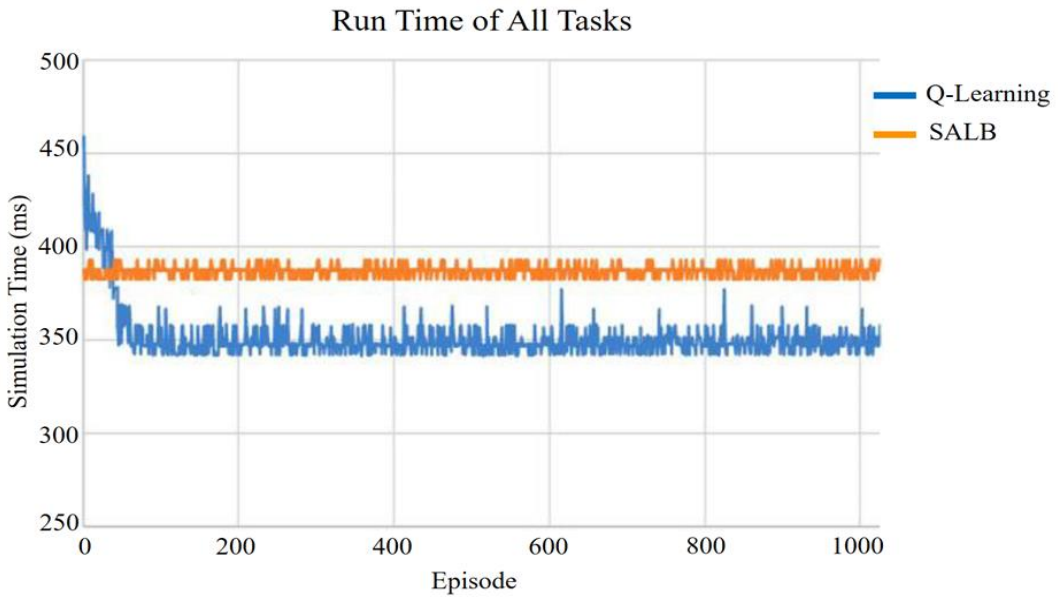

Fig. 11. Time to execute all tasks.

The overall execution time for the jobs using all four approaches is shown in Fig. 10. The average execution time of the input jobs from the first to the last algorithm iteration is used to calculate the overall delay. The examination reveals that the Q-Learning approach has the least overall delay when compared to other methods, which indicates that this method performs at a higher level of excellence.

Fig. 11 compares the execution times of both the suggested approach and the SALB method for each task that is entered into the system. The suggested solution significantly outperformed the SALB method in terms of the amount of time it took to execute all of the jobs that were input into the system during the simulation. Due to this, the suggested approach and system perform better than alternative methods that were also considered. The results show that the fog node considers the queue states, capacities, and the number of mobile devices linked to each node while deciding whether to disperse the load using the Q-Learning method. So, the unsuccessful allocation can be reduced by the proposed approach. Based on the results of the evaluation mentioned earlier, the proposed load balancing method is more stable than current load balancing approaches, and it significantly reduces both the network delay and the user response time.

VI. Conclusion

This article's goal is to outline a strategy for enhancing load balancing in fog nodes. Task distribution and load balancing are difficult problems in fog computing because of unique characteristics including topology dynamics and resource heterogeneity, and the adoption of conventional approaches to address these difficulties is inefficient. The application of machine learning algorithms, including reinforcement learning, is one of the cutting-edge methods for tackling complicated issues. In this article, the Q-Learning algorithm is used to demonstrate load balancing in a fog environment. By utilizing the experience that the learning agent acquires through interacting with the environment, this algorithm generates a long-term optimal strategy. As an agent, each fog node in the proposed technique searches the fog environment for low-load nodes suitable for allocating sub-tasks to reduce processing time and overhead with the use of the Markov decision process. The proposed solution has been tried with various numbers of mobile devices and fog nodes in the network, and it has produced successful results. According on simulation results, the suggested algorithm greatly reduces processing delay, user response time, and the likelihood of task assignment failure when compared to existing approaches. Some of the limitations of this research can include the following:

1) Hypothetical system model: This research assumes a system model and specific features for fog nodes, task distribution and network communication. These assumptions may not always hold true in practical deployments and potentially limit the generalizability of the findings.

2) Limited scalability testing: The scalability of the proposed algorithm may not have been extensively tested across a wide range of network sizes and configurations. Performance evaluation at different scales can provide valuable insight into algorithm robustness.

According to the study, the following can be considered for future research:

1) Dynamic adaptation: Enhancing the algorithm to dynamically adapt to changes in network conditions, such as node failures, varying workloads, or the mobility of fog nodes, to ensure robustness and scalability.

2) Optimization techniques: Investigate advanced optimization techniques to improve the efficiency and convergence speed of the reinforcement learning algorithm, potentially including deep reinforcement learning or other advanced methods.

3) Security and privacy considerations: Review the security and privacy implications of the proposed load

balancing approach, including potential vulnerabilities and mitigation strategies to protect sensitive data and ensure system integrity.

4) Integration with edge devices: extending the algorithm to combine edge devices and optimize the allocation of work in both fog nodes and edge devices, taking into account factors such as device capabilities, energy constraints, and communication protocols.

REFERENCES

[1] M. H. Kashani and E. Mahdipour, "Load Balancing Algorithms in Fog Computing," IEEE Trans Serv Comput, vol. 16, no. 2, pp. 1505–1521, 2022.

[2] I. Martinez, A. S. Hafid, and A. Jarray, "Design, resource management, and evaluation of fog computing systems: a survey," IEEE Internet Things J, vol. 8, no. 4, pp. 2494–2516, 2020.

[3] M. H. Kashani, A. Ahmadzadeh, and E. Mahdipour, "Load balancing mechanisms in fog computing: A systematic review," arXiv preprint arXiv:2011.14706, 2020.

[4] A. Yousefpour et al., "All one needs to know about fog computing and related edge computing paradigms: A complete survey," Journal of Systems Architecture, vol. 98, pp. 289–330, 2019.

[5] D. Puthal, R. Ranjan, A. Nanda, P. Nanda, P. P. Jayaraman, and A. Y. Zomaya, "Secure authentication and load balancing of distributed edge datacenters," J Parallel Distrib Comput, vol. 124, pp. 60–69, 2019.

[6] S. Gupta and N. Singh, "Fog-GMFA-DRL: Enhanced deep reinforcement learning with hybrid grey wolf and modified moth flame optimization to enhance the load balancing in the fog-IoT environment," Advances in Engineering Software, vol. 174, p. 103295, 2022.

[7] N. Khattar, J. Sidhu, and J. Singh, "Toward energy-efficient cloud computing: a survey of dynamic power management and heuristics-based optimization techniques," J Supercomput, vol. 75, pp. 4750–4810, 2019.

[8] S. Malik et al., "Intelligent load-balancing framework for fog-enabled communication in healthcare," Electronics (Basel), vol. 11, no. 4, p. 566, 2022.

[9] Sajadi, S. M., Kadir, D. H., Balaky, S. M., & Perot, E. M. (2021). An Eco-friendly nanocatalyst for removal of some poisonous environmental pollutions and statistically evaluation of its performance. Surfaces and Interfaces, 23, 100908.

[10] Wang, G., Wu, J., & Trik, M. (2023). A novel approach to reduce video traffic based on understanding user demand and D2D communication in 5G networks. IETE Journal of Research, 1-17.

[11] Sai Huang, Guangdeng Zong, Ning Zhao, Xudong Zhao, Adil M. Ahmad. Performance Recovery-Based Fuzzy Robust Control of Networked Nonlinear Systems against Actuator Fault: A Deferred Actuator-Switching Method, Fuzzy Sets and Systems, doi: 10.1016/j.fss.2024.108858, 2024.

[12] Kadir, D. H. (2021). Statistical evaluation of main extraction parameters in twenty plant extracts for obtaining their optimum total phenolic content and its relation to antioxidant and antibacterial activities. Food Science & Nutrition, 9(7), 3491-3499.

[13] Khezri, E., Yahya, R. O., Hassanzadeh, H., Mohaidat, M., Ahmadi, S., & Trik, M. (2024). DLJSF: Data-Locality Aware Job Scheduling IoT tasks in fog-cloud computing environments. Results in Engineering, 21, 101780.

[14] Zoraghchian, A. A., Asghari, A., & Trik, M. (2014). Thermal Control Methods for Reducing Heat in 3D ICs-TSV (Through-Silicon-Via).

[15] E. Khezri, E. Zeinali, and H. Sargolzaey, "SGHRP: Secure Greedy Highway Routing Protocol with authentication and increased privacy in vehicular ad hoc networks," PLoS One, vol. 18, no. 4, p. e0282031, 2023.

[16] Trik, M., Jabbehdari, S., Darvani, F. M., & Shojaei, A. (2015). Studying security protocol architecture based on cryptography algorithms. International Journal of Innovative Science, Engineering & Technology, 2(4).

[17] M. Trik, A. M. N. G. Molk, F. Ghasemi, and P. Pouryeganeh, "A hybrid selection strategy based on traffic analysis for improving performance in networks on chip," J Sens, vol. 2022, 2022.

[18] 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. 02, p. 2150008, 2021.

[19] Trik, M., Pour Mozafari, S., & Bidgoli, A. M. (2021). An adaptive routing strategy to reduce energy consumption in network on chip. Journal of Advances in Computer Research, 12(3), 13-26.

[20] Ding, X., Yao, R., & Khezri, E. (2023). An efficient algorithm for optimal route node sensing in smart tourism Urban traffic based on priority constraints. Wireless Networks, 1-18.

[21] Khosravi, M., Trik, M., & Ansari, A. (2024). Diagnosis and classification of disturbances in the power distribution network by phasor measurement unit based on fuzzy intelligent system. The Journal of Engineering, 2024(1), e12322.

[22] Blbas, H., & Kadir, D. H. (2019). An application of factor analysis to identify the most effective reasons that university students hate to read books. International Journal of Innovation, Creativity and Change, 6(2), 251-265.

[23] Cao Y, Niu B, Wang H, Zhao X. Event - based adaptive resilient control for networked nonlinear systems against unknown deception attacks and actuator saturation. International Journal of Robust and Nonlinear Control . doi: 10.1002/rnc.7231, 2024.

[24] M. Samiei, A. Hassani, S. Sarspy, I. E. Komari, M. Trik, and F. Hassanpour, "Classification of skin cancer stages using a AHP fuzzy technique within the context of big data healthcare," J Cancer Res Clin Oncol, pp. 1–15, 2023.

[25] J. Sun, Y. Zhang, and M. Trik, "PBPHS: a profile-based predictive handover strategy for 5G networks," Cybern Syst, pp. 1–22, 2022.

[26] M. Trik, H. Akhavan, A. M. Bidgoli, A. M. N. G. Molk, H. Vashani, and S. P. Mozaffari, "A new adaptive selection strategy for reducing latency in networks on chip," Integration, vol. 89, pp. 9–24, 2023.

[27] Omer, A. W., Blbas, H. T., & Kadir, D. H. (2021). A Comparison between Brown's and Holt's Double Exponential Smoothing for Forecasting Applied Generation Electrical Energies in Kurdistan Region.

[28] Sai Huang, Guangdeng Zong, Ning Zhao, Xudong Zhao, Adil M. Ahmad. Performance Recovery-Based Fuzzy Robust Control of Networked Nonlinear Systems against Actuator Fault: A Deferred Actuator-Switching Method, Fuzzy Sets and Systems, doi: 10.1016/j.fss.2024.108858, 2024.

[29] Haoyu Zhang, Quan Zou, Ying Ju, Chenggang Song, Dong Chen. Distance-based Support Vector Machine to Predict DNA N6-methyladine Modification. Current Bioinformatics. 2022, 17(5): 473-482.

[30] Xiao, L., Cao, Y., Gai, Y., Khezri, E., Liu, J., & Yang, M. (2023). Recognizing sports activities from video frames using deformable convolution and adaptive multiscale features. Journal of Cloud Computing, 12(1), 1-20.

[31] Hu, H., Luo, P., Kadir, D. H., & Hassanvand, A. (2023). Assessing the impact of aneurysm morphology on the risk of internal carotid artery aneurysm rupture: A statistical and computational analysis of endovascular coiling. Physics of Fluids, 35(10).

[32] Hai, T., Kadir, D. H., & Ghanbari, A. (2023). Modeling the emission characteristics of the hydrogen-enriched natural gas engines by multi-output least-squares support vector regression: Comprehensive statistical and operating analyses. Energy, 276, 127515.

[33] Kadir, D. H., & Rahi, A. R. K. (2023). Applying the Bayesian technique in designing a single sampling plan. Cihan University-Erbil Scientific Journal, 7(2), 17-25.

[34] Saidabad, M. Y., Hassanzadeh, H., Ebrahimi, S. H. S., Khezri, E., Rahimi, M. R., & Trik, M. (2024). An efficient approach for multi-label classification based on Advanced Kernel-Based Learning System. Intelligent Systems with Applications, 200332.

[35] Mahmood, N. H., Kadir, D. H., & Alzawbaee, O. M. M. (2024). Building a Statistical Model to Forecast Traffic Accidents for Death and

Injuries by Using Bivariate Time Series Analysis. Zanco Journal of Human Sciences, 28(1), 278-289.

[36] Saleh, D. M., Kadir, D. H., & Jamil, D. I. (2023). A Comparison between Some Penalized Methods for Estimating Parameters: Simulation Study. QALAAI ZANIST JOURNAL, 8(1), 1122-1134.

[37] Fakhri, P. S., Asghari, O., Sarspy, S., Marand, M. B., Moshaver, P., & Trik, M. (2023). A fuzzy decision-making system for video tracking with multiple objects in non-stationary conditions. Heliyon, 9(11).

[38] M. Trik, S. P. Mozaffari, and A. M. Bidgoli, "Providing an adaptive routing along with a hybrid selection strategy to increase efficiency in NoC-based neuromorphic systems," Comput Intell Neurosci, vol. 2021, 2021.

[39] Chen Cao, Jianhua Wang, Devin Kwok, Zilong Zhang, Feifei Cui, Da Zhao, Mulin Jun Li, Quan Zou. webTWAS: a resource for disease candidate susceptibility genes identified by transcriptome-wide association study. Nucleic Acids Research.2022, 50(D1): D1123-D1130.

[40] Hu, H., Luo, P., Kadir, D. H., & Hassanvand, A. (2023). Assessing the impact of aneurysm morphology on the risk of internal carotid artery aneurysm rupture: A statistical and computational analysis of endovascular coiling. Physics of Fluids, 35 (10).

[41] Li, Y., Wang, H., & Trik, M. (2024). Design and simulation of a new current mirror circuit with low power consumption and high performance and output impedance. Analog Integrated Circuits and Signal Processing, 1-13.

[42] D. Mokhlesi Ghanevati, E. Khorami, B. Boukani, and M. Trik, "Improve replica placement in content distribution networks with hybrid technique," Journal of Advances in Computer Research, vol. 11, no. 1, pp. 87–99, 2020.

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