Proc. of the Int. Conf. on Electrical, Computer and Energy Technologies (ICECET 2022)
20-22 July 2022, Prague-Czech Republic

2022 International Conference on Electrical, Computer and Energy Technologies (ICECET) | 978-1-6654-7087-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICECET55527.2022.9872763

A Load Balancing and Routing Strategy in Fog Computing using Deep Reinforcement Learning

Nacer Edine Belkout
Department of Computer Science
University of Science and Technology Houari Boumediene
Algiers, Algeria
naceredine.belkout@etu.usthb.dz

Khaled Zeraoulia
Department of Computer Science
University of Science and Technology Houari Boumediene
Algiers, Algeria
khaled.zeraoulia@etu.usthb.dz

Mohamed Nasir Shahzad
Department of Informatics
University of Leicester
Leicester, United Kingdom
mns14@leicester.ac.uk

Lu Liu
Department of Informatics
University of Leicester
Leicester, United Kingdom
l.liu@leicester.ac.uk

Bo Yuan
Department of Informatics
University of Leicester
Leicester, United Kingdom
b.yuan@leicester.ac.uk

***Abstract*— Fog computing is proposed to overcome cloud computing limitations by extending its services close to the network edge. However, fog systems are still facing many challenges due to their large-scale distributed architecture and small resource power. We investigate in this paper resource management in fog computing and propose a load balancing strategy using Deep Reinforcement Learning (DRL) combined with a link-state routing protocol based on the Dijkstra algorithm. The objective of the strategy is to simultaneously minimize the tasks' processing and communication delay. The proposed solution involves designing a Load Balancer Smart Controller (LBSC) which engages a smart DRL agent in a fog environment. The LBSC examines the nodes and links' states in order to make the optimal decision by selecting the most suitable node and path to process each task. The performance of the proposed approach is tested in a dynamic IoT environment with a high-rate workload generation scenario. Simulation results show that the average total latency including processing and communication delays of the proposed method is reduced by around 50% in comparison to the classic load balancing algorithms.**

Keywords*— *Fog computing, Resources management, Load balancing, Routing, Deep Reinforcement Learning, Dijkstra

I. INTRODUCTION

The fog computing paradigm [1] is proposed to complete the cloud computing model by offering the same kind of services (processing, storage, intelligence, etc.), with the same mechanisms such as virtualization and scalability, through large-scale distributed nodes, geographically located near to the edge of the network. The purpose of this architecture is to bring computation closer to the end devices in order to minimize direct communication with the cloud, hence producing less network congestion and fast processing time. Although the outstanding benefits brought by fog computing, the paradigm is not devoid of weaknesses. The capacities of the fog devices are very restricted compared to that of the cloud even though they share the same characteristics [2]: they have less computational resources due to smaller processors, limited storage, and restricted power supply such as battery-powered devices. Thus, addressing resource management is essential to optimize the performance of fog systems. Load Balancing (LB) is one of the major resource management challenges in fog computing [3], it plays a crucial role in reducing the service response time and ensuring a high quality of experience for the users. Indeed, an efficient LB strategy that optimally distributes the incoming workload among the fog devices can prevent the underloading and overloading of the system's resources. Nevertheless, developing an optimal LB policy is not always simple since the workload generation rate from the IoT layer is stochastic and unpredictable most of the time. Therefore, introducing intelligence in the LB strategy to dynamically adapt its algorithm according to the system state is a promising solution.

Embedding intelligence into the fog draws the attention of the research community, definitely due to its ability to solve complex issues by providing automated mechanisms that can significantly enhance the performance of the system [4]. Unlike the traditional supervised Machine Learning models, Reinforcement Learning (RL) offers a unique learning concept; In RL, an intelligent agent is used to interact with the environment and take action for each current state. The learning process of the RL involves using a reward system in which the agent learns the appropriate action to take in a specific state based on the received reward [5]. Another important factor that may substantially impact the efficiency of the LB is the task routing path. Even though it is supposed that the processing time is optimized using a reliable LB strategy, the latency may increase during the task communication phase. Accordingly, the communication latency may dominate the service latency, in this case, as the devices' response time becomes insignificant in comparison.

In this paper, we design an intelligent resource management strategy based on Deep Reinforcement Learning (DRL) and Dijkstra algorithms, which tackle simultaneously load balancing and routing in fog computing. To the best of our knowledge, this is the first work that considers the task routing in LB schemes for fog computing platforms. The main contributions of this paper are summarized as follows:

- We propose a load balancing strategy based on a DRL model for a fog computing system. The objective of the strategy is to optimally distribute the generated workload among the computing entities in order to avoid the overload and underload of each one, minimizing consequently the processing delay.
- The proposed LB strategy is improved with a link-state routing protocol based on Dijkstra algorithms to deal with communication latency. The routing algorithm aims to transmit the tasks to the selected nodes through the least congested path in the network, hence reducing the communication delay.
- The LB and routing strategies are implemented in the Load Balancer Smart Controller (LBSC) which collects the required system information including the nodes and links states, selects the processing nodes using the DRL agent, and routes the task to the destination.
- The proposed strategy is simulated and tested in a dynamic environment in which the workload generated from IoT devices varies and can reach a high rate.

The remainder of this paper is organized as follows: Section II reviews the related work. Section III presents the preliminary knowledge regarding the RL algorithm. In Section IV we discuss the design of our system model, followed by a detailed description of the proposed algorithms in Section V. We illustrate in Section VI the obtained results. Finally, Section VII concludes the paper.

II. RELATED WORK

A lot of existing works in the literature take advantage of RL to optimize the performance of fog systems. The authors in [6] propose an LB strategy based on two software components using RL and genetic algorithms in the fog layer: A Load Balancer Agent (LBA) which is used to classify the fog servers based on their load status, and the second is the Resource Allocator (RA) which is responsible to select the suitable nodes for processing the tasks. Similarly, the authors in [7], represent the states of the environment using buffer occupation of the available fog nodes to design the RL algorithm, with the objective of minimizing processing latency. Other similar works involve the use of central controllers such as SDN to design the RL algorithm. To find an optimal offloading decision, the authors in [8] make use of an SDN controller to design a decision-making strategy based on RL. The evaluation results show that the proposed algorithm achieves higher performance compared to classical schemes in terms of traffic arrival rate and overload probabilities.

The existing LB schemes using RL are experimented in static fog systems since RL algorithms lack the ability to deal with complex environments which have high-dimensional state-action space such as fog systems. DRL on the other hand is designed to overcome those limitations. In DRL, the RL algorithm is assisted by a deep neural network to scale the decision-making using the properties of an approximation function [9]. In recent fog computing studies, there are only a few works that employ the DRL algorithm. The authors in [10] investigate offloading decision problems in Mobile Edge Computing (MEC). They formulate the latency and energy consumption as an optimization problem, then manage to solve it using a DRL SARSA algorithm. Besides, using DRL can be an asset for complex heterogeneous environments. Authors in [11] develop a Deep Deterministic Policy Gradient method which benefits from the scalability of the DRL algorithm to deal with tasks heterogeneity as well as the mobility of end devices in an edge computing environment.

In multi-layered distributed environments such as fog computing [12], the tasks may travel through multiple nodes to reach the destination services. Thus, the communication latency depends not only on the capacities of links and the traveling distance but also on the amount of circulating traffic. Although the above proposals effectively tackle task processing delay in LB, the network state is not considered in the task transmission phase. There are many existing solutions to solve link-state routing problems, the Dijkstra is one of the commonly used algorithms to calculate the shortest path between distant nodes based on the total cost of each possible path. One of its famous applications is the OSPF (Open Shortest Path First) routing protocol [13]. Authors in [14] adopt the Dijkstra algorithm to develop a routing protocol in a cloud-edge environment, considering the shortest path, latency, and energy consumption. The purpose of the protocol is to send the packets from source to destination through the most appropriate path while meeting the application requirements. Additionally, they designed a MapReduce approach to reduce path computation in large-scale architecture. Definitely, the above presented works do not consider task routing in load balancing schemes.

III. PRELIMINARY KNOWLEDGE

RL is a decision-making technique that provides a system with the intelligence to learn how to properly behave in the environment according to the observed state. From a mathematical perspective, the aim of the process is to optimize progressively a performance criterion [15]. The strategy consists of using an agent that interacts with the environment by observing its current state, then deciding to take a specific action based on the observation. Thereafter, the agent will be rewarded according to the quality of its action. The objective of the agent is to maximize long-term reward earning. Generally, RL problems are formulated as a Markov Decision Process (MDP) which consists of four elements <S, A, T, R> [16]:

- States: a set of possible states of the environment.
- Actions: a set of possible actions that can be taken in the environment.
- Transitions: a transition from one state to the next state based on a probability distribution.
- Reward function: which attributes a reward or penalty to the agent for taking an action in a given state of the environment.

One of the approaches to solving some MDP problems and finding the optimal behavior is measuring the quality of each state-action using a value function. In this matter, Q-learning is one of the most popular algorithms. Q-Learning is a model-free RL algorithm which consists of building a Q-table containing a set of $Q(s,a)$ values for each state-action pair. The $Q(s,a)$ values are regularly updated after each training using the obtained reward $R(s,a)$ for the state-action pair, the discount factor γ, and the max Q-value $max[Q(s',a^{*})]$ of all possible actions of the next state s', as described in equation (1) [17]:

$$Q(s,a) = R(s,a) + \gamma\, max[Q(s',a^{*})] \qquad (1)$$

In a typical RL scenario, the interaction of the agent with the environment is managed using an exploration-exploitation trade-off [16]. In the beginning of the training, the agent does not possess any prior knowledge about the system. Therefore, it has to explore the environment by performing random actions and perceive their consequences through the obtained rewards. Progressively, the agent exploits the accumulated knowledge to maximize the long-term reward.

IV. SYSTEM MODEL

We consider in this work a three-layered architecture, the different elements of the architecture are shown in Figure 1

1) *The IoT layer:* it represents the edge of the network, it is essentially composed of smart devices provided with sensing capability, and end-users who eventually exploit the applications. The latency of the results messages received by the users is neglected in this work [18]. The generated data is routed to the next layers through the LBSC for processing.

2) *The Fog layer:* consists of randomly distributed and linked fog nodes, they represent the computing devices which could be gateways, routers and switches [19]. They provide similar computation and storage mechanisms as the cloud, whereas their hardware capacity is less powerful.

3) *The Cloud layer:* it relies on a cloud data-center model which provides a powerful computation and storage capacity for IoT applications. The communication with this entity is done through a gateway with the fog layer.

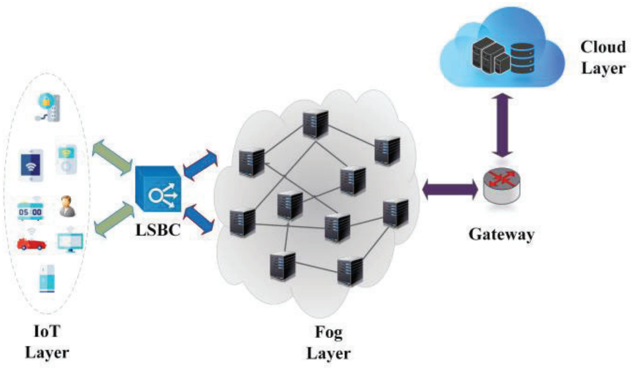

Fig. 1. The architecture of the cloud - fog – IoT system

A. Load Balancer Smart Controller framework

The main components of LBSC are highlighted in Figure 2. The LBSC is responsible for distributing the tasks among the nodes and routing them to the destination. The LBSC is always kept updated in real-time about the state of the environment by receiving the nodes and links state through the control interface. For each task received, the task management module sends a selection request to the AI manager submodule which commands the selection module, and the source node ID to the routing manager submodule commanding the routing module. Using the nodes' state information, the DRL agent deployed in the selection module chooses the destination node and then sends its ID to the routing module. Thereafter, the Dijkstra algorithm implemented in the routing module uses the source and destination nodes ID to obtain all possible paths from the routes database, then it chooses the most suitable one based on the links' state. In order to route the task to the destination, the selected path is included in the task data using the packet constructor submodule and transmitted through the routing interface. The selection and routing modules are covered in more detail in the next sections.

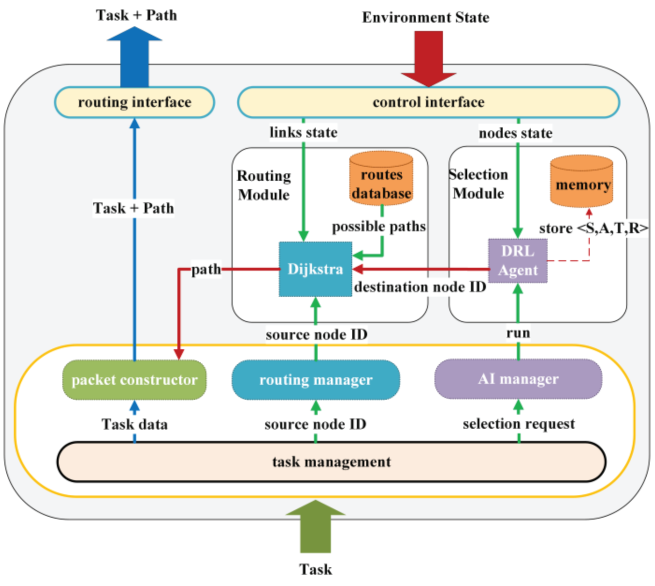

Fig. 2. Load Balancer Smart Controller (LBSC) framework

B. Communication mode

The smart devices in the IoT layer can generate multiple tasks simultaneously. Each task $i, i \in \{1,2,\ldots,I\}$, has a data size θ_i and required CPU instructions λ_i. Each task i is transmitted through the LBSC to a processing entity $n, n \in \{1,2,\ldots,N\}$, located in the top layers, which can be either a fog node or cloud server hosting the computing services. It is assumed that a transmission link $k, k \in \{1,2,\ldots,K\}$, can only send one task simultaneously, and the bandwidth β_k is the same in both uplink and downlink. The task may pass across multiple nodes and links to travel from the source to the destination. The link latency $lt_{i,k}$ represents the necessary time to transmit a task i in link k, expressed by equation (2) [20]:

$$lt_{i,k} = \frac{\theta_i}{\beta_k} \tag{2}$$

The waiting time $wt_{i,k}^{link}$ of a task i in link k is expressed by the sum of the link latency $lt_{z,k}$ of all waiting tasks $z, z \in \{1,2,\ldots,Z\}$, as defined in equation (3):

$$wt_{i,k}^{link} = \sum_{z=1}^{Z} lt_{z,k} \tag{3}$$

Using equation (2) and (3), the communication latency ct_i to transmit a task i to the destination is formulated in equation (4):

$$ct_i = \sum_{k=1}^{K} wt_{i,k}^{link} + lt_{i,k} \tag{4}$$

C. Computing model

Periodically, the LBSC receives lightweight information packets containing the load states of the computing nodes. For simplicity, the latency generated from those packets is not considered in this work. Based on the obtained information, the LBSC is in charge of making decisions of selecting the right node where each task should be processed and routing it in the right path. We consider in this paper a classic computation unit in each computing node n, in which the tasks are processed following a first-in-first-out queue. The waiting time $wt_{i,n}^{node}$ in the node is depending on the number of tasks Y in the queue. The execution time $et_{i,n}$ and the waiting time $wt_{i,n}^{node}$ of a task i, in the computing node, are measured using the equations equation (5) and (6) respectively [19].

$$et_{i,n} = \frac{\lambda_i}{\varphi_n} \tag{5}$$

$$wt_{i,n}^{node} = \sum_{y=1}^{Y} et_{y,n} \tag{6}$$

Where φ_n is the CPU frequency of the computing node. Hence, the processing time pt_i is expressed by equation (7):

$$pt_i = et_{i,n} + wt_{i,n}^{node} \tag{7}$$

The quality of service to transmit and process a task is represented by the total latency T_i, described by equation (8):

$$T_i = pt_i + ct_i \tag{8}$$

V. THE PROPOSED LOAD BALANCING STRATEGY

The main difference between RL and DRL lies in decision-making. In the classic Q-learning, the optimal Q-value of each state-action pair is acquired from the Q-table. Whereas in the DRL model, it is predicted using a neural network [5]. In order to reach our goal, the discussed LB problem is first formulated as MDP which involves placing a DRL agent in a fog computing environment. The agent is in charge of developing an optimal workload distribution strategy for the purpose of balancing the load between computing entities. Then, the strategy is afterward enhanced with a Dijkstra-based routing protocol that chooses the optimal path to route the task. The DRL and Dijkstra algorithm are implemented respectively in the selection and routing modules of the LBSC.

A. DRL agent design

The MDP elements ⟨S, A, T, R⟩ are defined as follows:

- State-space: $S = \{s_1 = [L_1^1, L_2^1, \ldots, L_N^1], s_2, \ldots, s_T\}$, where:
  - s_t: the current state of the environment.
  - L_n^t: the load index of the node « n »in a state « t ».
- Action space: $A = \{a_1, a_2, a_3 \ldots, a_n\}$, where a_n is the chosen node for processing the next task.
- Transition: the next state of the environment is depending on the current task distribution including the impact of the chosen action.
- Reward: the reward system depends on the chosen action in which the agent is rewarded if the index load of the selected node meets the desirable performances, otherwise, it is penalized.

The neural network architecture shown in Figure 3 is designed based on the MDP formulation:

- The input vector corresponds to the state-space s_t. The states of the nodes are collected by the LBSC. Hence, the vector size is the same as the number of processing nodes N in the environment.
- The output vector corresponds to the Q-value of each possible action in A. Thus, the vector size is also the same as the number of nodes N.

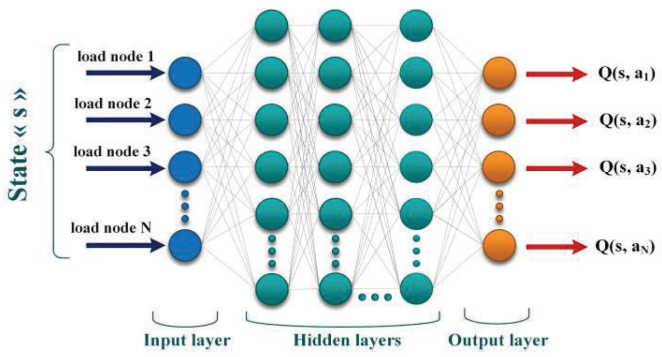

Fig. 3. The proposed neural network architecture

The training process of the neural network is carried out using the classical Mean Square Error loss function between the $Q(s,a)$ and its target $\hat{Q}(s,a)$, where α is the learning rate as shown in equation (9) [5]:

$$\Delta w = \alpha\left[\left(R + \gamma\, max_a Q(s',a)\right) - \hat{Q}(s,a)\right]^2 \quad (9)$$

Furthermore, the ReLU activation function is used [21]. The DRL algorithm is usually designed with an experience replay technique [5] in which each experience instance tuples <S, A, T, R> are stored in a memory buffer. The memory buffer is regularly updated with new experiences instances through a first-in-first-out queue, this means that the older experiences are deleted and only the newest are kept in the memory. Accordingly, samples from the memory buffer are used to train the agent at the end of each scenario. Thereby, the agent is always kept updated with the newest events, which provides the algorithm with great flexibility to adapt to the new changes in the environment. The operating process of the DRL agent is described in a diagram in Figure 4.

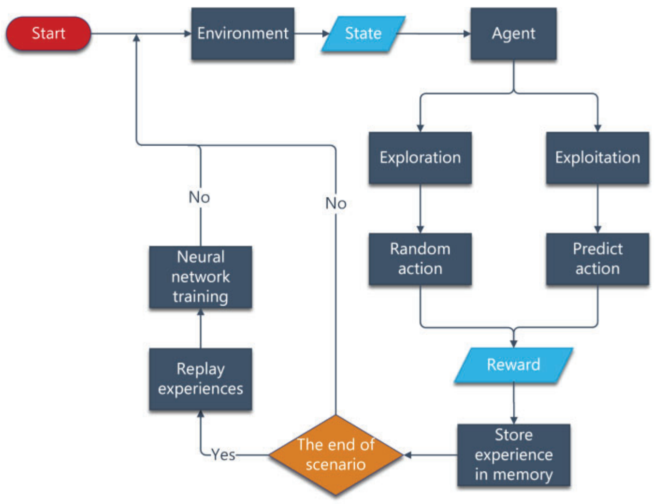

Fig. 4. DRL agent operating process

The IoT sensors in the bottom layer generate and then transmit the tasks to LBSC. The DRL agent receives by interaction the actual state of the environment including the load index of each processing node in the system. The load index < L> is related to the number of tasks in the waiting queue and the processing power of the node. Using the collected information, the DRL agent makes a decision by selecting a node to forward the next task. The decision is then evaluated by the algorithm; the agent is rewarded only if the selected node possesses a low load index which satisfies the performance requirement of the service level agreement (SLA). After enough iterations, the agent finds the best LB strategy that optimizes the computing capacity of the system.

B. *Routing Strategy*

Our strategy relies on a link-state protocol to route the data from the source to the destination. Initially, the LBSC builds a routing database containing the different paths to reach every fog node and the cloud, by exchanging the topology's information messages (nodes locations and connections between nodes) from the available nodes. Thus, the LBSC can adapt the routing table in case any changes occur in the architecture, such as newly added elements or nodes failure. Moreover, the LBSC periodically receives the current cost of each link. The routing strategy implies the use of the Dijkstra algorithm to find the optimal path as described in Algorithm 1. After receiving the current cost LC of all links in the topology, the algorithm calculates the total cost TC for each possible path in the path list PL. Finally, the path P with the minimum cost is returned.

Algorithm 1: Dijkstra routing

```
     Input: source node $S$; destination node $D$; current
     links costs $LC$;
     Output: the selected path $P$;
1    Initialization: $TC \leftarrow 0$; $destPath \leftarrow \emptyset$
2    $PL \leftarrow getAllPossiblePaths(S, D)$
3    for each $path \in PL$ do
4        if $TC = 0$ then
5            $TC \leftarrow calculPathTotalCost(path, LC)$
6            $destPath \leftarrow path$
7        else
8            $pathCost \leftarrow calculPathTotalCost(path, LC)$
9            if $pathCost < TC$ then
10               $TC \leftarrow pathCost$
11               $destPath \leftarrow path$
12           end if
13       end if
14   end for
15   $P \leftarrow destPath$
```

VI. IMPLEMENTATION AND RESULTS

The proposed system is simulated using Yet Another Fog Simulator (YAFS) [22] which is a Python-based library developed upon discrete-event simulation and complex network theory. YAFS is a particularly flexible and robust fog computing simulator, it provides total control of the scenario's events and topology's elements, which ease the implementation of the desired strategies such as routing, mobility, nodes failure, and dynamic control of workload generation. However, we had to extend the YAFS libraries to fit our needs. We are going to present in this section the simulation parameters used in our application, then analyze the obtained results and evaluate the performance of the proposed smart load balancing scheme.

A. *Simulation parameters*

For the sake of the evaluation of the system and to better expose the impact of our solution, we assume that the latency generated by the information packets of the topology is neglected and the system architecture is static during the simulation. We conducted three independent experiments using multiple numbers of fog nodes ($N = 10, 20 \text{ and } 30$) with various computing capabilities. The nodes are randomly distributed and connected with different link capacities. We also chose to apply a high-rate task generation from only 4 IoT devices rather than deploying a large number of IoTs. The simulation is carried out in a dynamic environment wherein the IoTs' task generation rate varies during the simulation and can reach up to $I = 3000$ tasks for each IoT device. The simulation setup is presented in Table I.

Furthermore, we have simulated for benchmarking three classical LB algorithms in the same experiment conditions:

- The Random LB (RLB), which randomly distributes the workload among fog nodes and the cloud.
- The Round Robin (RR) algorithm which equally distributes the workload between the processing nodes.
- The Weighted Round Robin (WRR) in which the workload is distributed according to the weight index attributed to each processing node, corresponding to its computing power.

The simulation is conducted on a P2. xlarge AWS instance running on Intel Xeon E5-2686 v4 and NVIDIA K80 GPU which is efficient to accelerate the training phase of the DRL neural network.

TABLE I. Setup parameters

Parameter	***Value***
number of Cloud	1
number of Fog	[10 to 30]
number of IoT	4
number of Tasks	[500 to 3000] Task/IoT devices
cloud RAM	40 GB
cloud CPU frequency	25 GHz
fog RAM	2 GB
fog CPU frequency	[0.5 to 8] GHz
links bandwidth	[5 to 10] Mbps
task size	2 KB
task instructions requirement	5 Mega instructions

B. *Performance evaluation*

During the simulation, we measured the processing latency, the communication latency, and the total latency for each event. Thereafter, we calculated the average latency for the entire simulation's events. The obtained results are shown in Figure 5, Figure 6, and Figure 7.

The average processing latency is shown in Figure 5. We notice at first that increasing the number of nodes from $N = 10$ to $N = 30$ will reduce the average processing time for all algorithms since it brings more processing power to the system which reduces the nodes' overload probability. Obviously, the highest processing latency is manifested in the RLB algorithm « 615ms – 335ms », the workload, in this case, is randomly distributed between fog nodes and the cloud, this may lead to an unbalanced state in which the weaker nodes get overloaded, and the powerful one in the other hand are less solicited. The WRR outperforms the RR algorithm by more than 30%, it offers a more balanced workload distribution than the RR due to its weight-based mechanism which forwards more tasks to powerful nodes than the weaker ones, thus better exploiting the fog resources. Nevertheless, this resource exploitation is not optimal since the weights attributed to each node do not accurately match its computing power. In contrast, the LB-DRL algorithm examines the current load state of all nodes in the system, including their processing power and the number of tasks in their waiting queues. Thereby, the LB-DRL has successfully developed an optimal strategy that exploits in the best possible way the computing potential of the system, outperforming the WRR algorithm and reducing the processing latency by 30%.

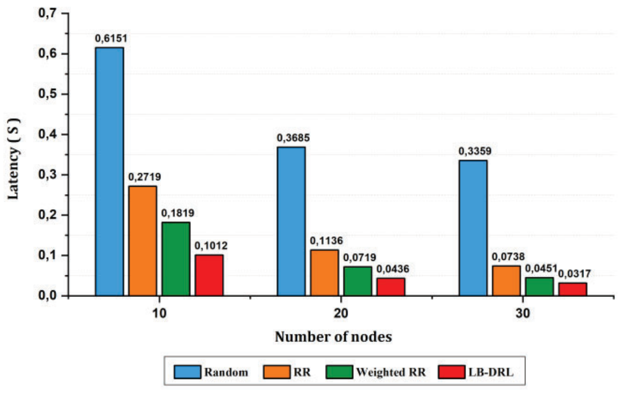

Fig. 5. The average processing latency

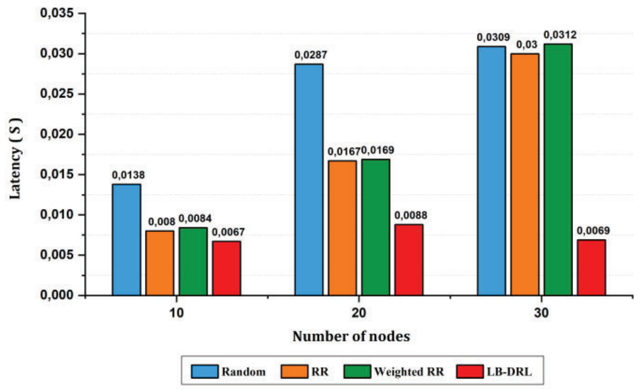

Fig. 6. The average communication latency

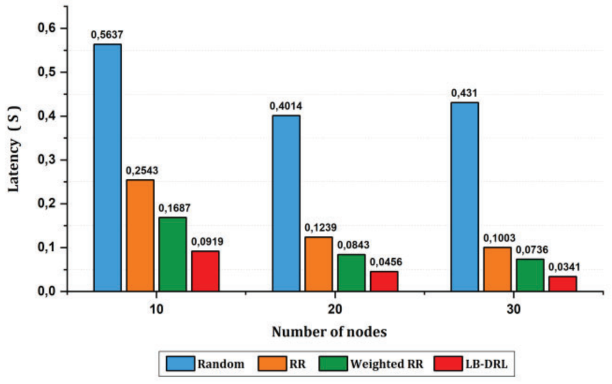

Fig. 7. The average total latency.

Figure 6 depicts the average communication latency for all algorithms. We can clearly notice that without the proposed routing strategy, the communication latency increases in the case of the three classic LB algorithms, exceeding the 30ms in the case of $N = 30$ fog nodes. Whereas in the proposed LB-DRL algorithm, it is relatively stable and does not exceed 10ms, which is three times less than the others. The reason behind the obtained results is that the classic LB algorithms are not supported by default with an efficient routing strategy, the tasks, in this case, are routed using the default paths (minimum hops) which may raise the number of waiting tasks, leading to the congestion of the links. Additionally, adding more fog nodes to the environment will consequently add more links to the network, hence producing more latency to reach the far located nodes, which explains the rise of communication latency. In the LB-DRL however, the LBSC exploits dynamically the link states information to route the task through the least congested paths using the Dijkstra algorithm. Moreover, adding more nodes to the system does not impact the communication latency since the Dijkstra algorithm can effectively find the optimal path to send the tasks. Therefore, the results in Figure 6 definitely demonstrate the prominence of the routing strategy for LB in fog computing, as well as the outstanding performance of the Dijkstra-based routing protocol.

The average total latency shown in Figure 7 represents the quality of service of our proposed strategies. Overall, although the WRR algorithm provides quite acceptable performances compared to RLB and RR. Our LB-DRL solution outclasses all algorithms, particularly WRR by nearly 50%.

VII. Conclusion

We proposed in this paper a load balancing solution based on a Deep Reinforcement Learning algorithm in order to exploit at best, the restricted fog computing resources and manage the high-rate workload generation to deliver the best quality of service for IoT applications' users. The numerical results demonstrate that the DRL agent was able to establish an optimal workload distribution strategy among the computing nodes based on their current load states. Our DRL policy achieves a lower processing delay in comparison to classic load balancing algorithms. Moreover, we demonstrate in this work the importance of developing an efficient routing strategy to improve load balancing. Indeed, the system performance is enhanced using the Dijkstra-based routing protocol which considerably reduces the communication delay. In future work, we will investigate the parameters affecting energy consumption in fog computing environments, in order to find an optimal trade-off between latency and energy consumption

References

[1] Z. Mahmood, "Fog Computing: Concepts, Frameworks and Technologies," *Springer Int. Publishing*, New York, USA, pp. 4-13, 2018.

[2] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys*, vol. 52, no. 5, pp. 1–37, 2019.

[3] M. Ghobaei-Arani, A. Souri and A.A. Rahmanian, "Resource Management Approaches in Fog Computing: A Comprehensive Review," *Journal of Grid Computing*, vol. 18, no. 1, pp. 1–42, 2020.

[4] E. Hormozi, H. Hormozi, M. K. Akbari and M. S. Javan, "Using of Machine Learning into Cloud Environment (A Survey): Managing and Scheduling of Resources in Cloud Systems," in *2012 Seventh Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing*, Victoria, BC, 2012, pp. 363-368.

[5] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang and X. Shen, "Deep Reinforcement Learning for Autonomous Internet of Things: Model, Applications and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1722-1760, 2020.

[6] F.M. Talaat, M.S. Saraya, A.I. Saleh, A.A. Hesham and H.A. Shereen, "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.

[7] L. Mai, Nhu-Ngoc Dao, and M. Park, "Real-Time Task Assignment Approach Leveraging Reinforcement Learning with Evolution Strategies for Long-Term Latency Minimization in Fog Computing", *Sensors*, MDPI, vol. 18, no. 2830, Aug. 2018.

[8] J. 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 Conf. (WCNC)*, Marrakesh, Morocco, 2019, pp. 1-7

[9] K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26-38, Nov. 2017

[10] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio and G. Fortino, "Task Offloading and Resource Allocation for Mobile Edge Computing by Deep Reinforcement Learning Based on SARSA," *IEEE Access*, vol. 8, pp. 54074-54084, 2020

[11] Y. Li, F. Qi, Z. Wang, X. Yu and S. Shao, "Distributed Edge Computing Offloading Algorithm Based on Deep Reinforcement Learning," *IEEE Access*, vol. 8, pp. 85204-85215, 2020

[12] F. Y. Okay and S. Ozdemir, "Routing in Fog-Enabled IoT Platforms: A Survey and an SDN-Based Solution," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4871-4889, Dec. 2018.

[13] N. Gihan, and W.G. Ali, "Network Routing Protocol using Genetic Algorithms", *Int. Journal of Electrical & Computer Sciences IJECS-IJENS*, vol. 10, no. 2, pp. 40-44, 2010.

[14] Buzachis A., Galletta A., Celesti A., Villari M. "An innovative MapReduce-based approach of Dijkstra's algorithm for SDN routing in hybrid cloud, edge and IoT scenarios," in *European Conf. on service-oriented and cloud computing*. Springer, Cham, 2018, pp. 185–198.

[15] C. Szepesvari, "Algorithms for reinforcement learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no 1, pp. 1-103, 2010.

[16] M. Wiering and M. Otterlo, "Reinforcement Learning and Markov Decision Processes," *Reinforcement Learning: State-of-the-Art*, Springer-Verlag, Germany, pp. 3-13, 2012.

[17] D. Pandey and P. Pandey, "Approximate Q-Learning: An Introduction," in *2010 Second Int. Conf. on Machine Learning and Computing*, Bangalore, 2010, pp. 317-320.

[18] G. Zhang, F. Shen, Z. Liu, Y. Yang, K. Wang and M. Zhou, "FEMTO: Fair and Energy-Minimized Task Offloading for Fog-Enabled IoT Networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4388-4400, Jun. 2019.

[19] H. Sun, H. Yu, G. Fan, and L. Chen, "Energy and time efficient task offloading and resource allocation on the generic IoT-fog-cloud architecture, " *Peer-to-Peer Networking and Applications*, vol. 13, no. 2, pp. 548–563, Mar. 2020.

[20] Q. Zhu, B. Si, F. Yang and Y. Ma, "Task offloading decision in fog computing system," *China Communications*, vol. 14, no. 11, pp. 59-68, Nov. 2017.

[21] G. Wang, G. B. Giannakis and J. Chen, "Learning ReLU Networks on Linearly Separable Data: Algorithm, Optimality, and Generalization," *IEEE Transactions on Signal Processing*, vol. 67, no. 9, pp. 2357-2370, May. 2019.

[22] I. Lera, C. Guerrero and C. Juiz, "YAFS: A Simulator for IoT Scenarios in Fog Computing," *IEEE Access*, vol. 7, pp. 91745-91758, 2019.