Computer & Robotics

A Reinforcement Learning Method for Joint Minimization of Energy Consumption and Delay in fog Computing

Reza Besharati[a] ,Mohammad Hossein Rezvani [a,*],Mohammad Mehdi Gilanian Sadeghi [a]

Department of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University,Qazvin. Iran

Received 11 July 2022; Accepted 27 October 2022

Abstract

In fog computing, optimal offloading is of crucial importance due to the limited energy of mobile devices. In this regard, using machine learning methods has recently attracted much attention. This paper presents a reinforcement learning-based approach to motivate users to offload their tasks. We propose a self-organizing algorithm for offloading based on Q-learning theory. Performance evaluation of the proposed method against traditional and state-of-the-art methods shows that it consumes less energy. It also reduces the execution time of tasks and results in less consumption of network resources.

Keywords: Fog Computing; Computation offloading; Optimization; reinforcement learning; Q-learning

1.Introduction

The use of fog computing technology in fields such as industry and vehicle traffic management is expanding rapidly [1, 2]. The basic idea is to transfer large amounts of data from the Internet of Things (IoT) devices to a remote cloud. Here, if an end device's computing capacity is insufficient, the task is offloaded to a near-cloud device. Similarly, if the resources on the cloud machine are inadequate, the task is offloaded to the remote cloud.

The literature review shows that most research has been focused on minimizing delay and energy consumption. Some of the essential methods used in the literature are game theory [3, 4], auction theory [2, 5], and meta-heuristics [1, 6]. Recently, the use of machine learning methods, especially Reinforcement Learning (RL) [7], for offloading optimization has been considered. Here, each agent learns to behave based on the reward/penalty of the previous rounds in a way that leads to the optimization of the goal in the following rounds. In other words, in RL, the agent tries to learn from the experiences gained from previous actions. This is the way that humans and animals learn in the real world. Various theories in RL have many diversities in goals and working methods. Their main difference is in the goals and type of modeling. The main contribution of this article is as follows:

- After modeling the system with queuing theory, we solve the joint minimization problem of delay and energy consumption. For this purpose, we perform

*Corresponding Author. Email: mhossein.rezvani@gmail.com

new modeling with the Q-learning method based on real-world requirements.

- The reward function we design is based on price. We consider a predetermined price for each joule of energy consumption. This paper complements one of the most recent research [2] in which each agent uses a *Second-price Sealed-bid auction (SPSB)* for task offloading. In this modeling, only the user who wins the auction has the right to use the resources of the fog machine to offload his/her tasks. Nevertheless, our proposed model does not ignore the chances of losing users in the following rounds. In this way, it tries to provide fairness to all users.

The remainder of this paper is organized as follows: Section 2 reviews the most critical studies on offloading optimization based on RL methods; Section 3 explains the system model; Section 4 describes the proposed method; Section 5 presents the evaluation results of the proposed method along with statistical analysis; Finally, Section 6 concludes the study.

2- Related Works

In recent years, critical research has been conducted on modeling offloading based on RL, most with Deep Reinforcement Learning (DRL). Interested readers can refer to [8] for a more in-depth study. Santos et al. [9] proposed a DRL method for joint energy consumption and cost optimization. They used embedded Service Function Chains (SFC). Mixed-integer Linear Programming (MILP) is used to solve the problem on a small scale. Their method can improve the acceptance ratio for user requests. In another research with a similar purpose, which was carried out by Gazori et al. [10], load balancing was also considered. Rahman et al. [11] proposed a DRL algorithm to minimize Fog Radio Access Networks (F-RANs) delay. It optimally allocates computing resources and power to users. Similar research [12] showed that the proposed method could reduce the delay, energy consumption, and network utilization compared to full local implementation and First-Fit (FF) plans. Most traditional fog networks use the FF algorithm for allocation, which is by no means optimal but has a high speed. In another research [13], the Deep Deterministic Policy Gradient (DDPG) algorithm was used. This method does not need to know the transfer probabilities in different states. The authors showed that their approach is superior to the Policy Gradient (PG), Deterministic Policy Gradient (DPG), and Actor-Critic (AC) methods. Baek et al. [14] proposed a combination of DRL and game theory to solve the offloading optimization problem. In this game, fog nodes work together to maximize local rewards. They do not need to know the states of other nodes and only work based on local observations. In this research, DRL is used to approximate reward functions. The technique used in this research results in a higher acceptance rate and minor overflow than state-of-the-art methods.

The use of RL is not limited to offloading. It has also been used in other fields, such as load balancing [15], spot price forecasting [16, 17], content distribution [18], and scheduling [19]. After reviewing the literature, the following points can be highlighted as the most important differences between our modeling and previous research:

1) Using queuing theory, we provide closed-form relations to model delay and energy for each user. None of the above studies have used such modeling to design the reward function. The formulation used in this research provides a more realistic description of the fog/cloud environment.
2) None of the above studies suggest a price-based model. We set a predetermined price for each joule of energy consumed. As we will see later in Eq. (13), this price is used in the reward function.

3.System Model

As shown in Fig. 1, the system consists of M mobile users, F fog devices, and a central cloud. Each mobile user can only connect to a fog node via the Base Station (BS). The set of mobile users is denoted by $\mathcal{U} = \{u_1, u_2, ..., u_M\}$. Also, the collection of fog nodes is denoted by $\mathcal{F} = \{f_1, f_2, ..., f_F\}$. Every mobile user has several tasks to be performed. These tasks are entered into the system continuously. If the user's computing resources are insufficient to perform the task, he/she offloads the task to a fog node. Similarly, if the computing resources of the fog node are inadequate, the task is offloaded to the central cloud. For each fog node f_j, we denote all users who are offloading to it by a set $\mathbf{C}(f_j)$. For example, in Fig. 1, $\mathbf{C}(f_1) = \{u_1, u_2, u_3, u_4\}$ indicates that users u_1, u_2, u_3, and u_4 are offloading their tasks to the node f_1. Similarly, in this figure, we can write $\mathbf{C}(f_2) = \{u_5, u_6\}$ and $\mathbf{C}(c_1) = \{f_1, f_2\}$.

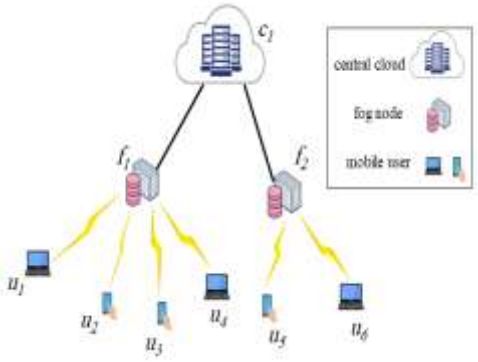

Fig. 1- An illustration of the system model

3.1.The Edge Layer Modelling

As with previous research [2], we assume that each mobile user $u_i \in \mathcal{U}$ can have multiple tasks to be performed. The number of these tasks follows the Poisson distribution with the mean arrival rate λ_{u_i}. The size of each task submitted to the user u_i is denoted by θ_i. Performing this task requires spending several processor cycles for each bit denoted by σ_i. The service rate of performing the task on the processor of the mobile device u_i, is calculated as follows:

$$\mu_{u_i} = \frac{C_{u_i}}{\sigma_i . \theta_i} \tag{1}$$

which C_{u_i} represents the computing capacity of the mobile user u_i in cycles/sec. This study uses an *M/M/1/K* queuing system to model the mobile user. In this model, the maximum number of buffer rooms is *K*. If the number of tasks entered is more than *K*, the extra ones will be dropped. Tasks are entered at a rate λ_{u_i} while the acceptance rate by the mobile device is $\lambda^a_{u_i}$. Let $\pi^K_{u_i}$ represent the task blocking ratio. The blocking probability (i.e. $k = K$) can be obtained as follows [2]:

$$\pi^K_{u_i} = \frac{(1-\rho_{u_i}).(\rho_{u_i})^K}{1-(\rho_{u_i})^{K+1}} \tag{2}$$

The average number of tasks, L_{u_i}, in the mobile user u_i is obtained as follows::

$$L_{u_i} = \frac{\rho_{u_i}}{1-\rho_{u_i}} - \frac{(K+1).(\rho_{u_i})^{K+1}}{1-(\rho_{u_i})^{K+1}} \tag{3}$$

Now, applying Little's law gives the local execution time for each task, $t^l_{u_i}$, as follows:

$$t^l_{u_i} = \frac{L_{u_i}}{\lambda^a_{u_i}} = \frac{1}{\lambda^a_{u_i}} . \left[\frac{\rho_{u_i}}{1-\rho_{u_i}} - \frac{(K+1).(\rho_{u_i})^{K+1}}{1-(\rho_{u_i})^{K+1}} \right] \tag{4}$$

Now, the local energy consumption of the mobile user u_i is calculated as follows:

$$E^l_{u_i} = P^l_{u_i} t^l_{u_i} = \frac{P^l_{u_i}}{\lambda^a_{u_i}} . \left[\frac{\rho_{u_i}}{1-\rho_{u_i}} - \frac{(K+1).(\rho_{u_i})^{K+1}}{1-(\rho_{u_i})^{K+1}} \right] \quad (5)$$

which $P^l_{u_i}$ represents the local power of the mobile user u_i . Now, the offloading time is calculated as follows:

$$t^o_{u_i} = \frac{\pi^K_{u_i} . \lambda_{u_i} . \theta_i}{R_i} \quad (6)$$

where R_i denotes the offloading rate. Finally, the energy consumed during the task offloading from the mobile device u_i to the relevant fog device is calculated as follows:

$$E^o_{u_i} = P^o_{u_i} . t^o_{u_i} = P^o_{u_i} . \frac{\pi^K_{u_i} . \lambda_{u_i} . \theta_i}{R_i} \quad (7)$$

3.2.The Fog Layer Modelling

We represent the incoming traffic to each fog device with an M/M/c/K' model, in which there are c individual internal servers and a buffer of size K'. As mentioned in the previous section, if a task can not be admitted to the mobile device $u_i \in \mathbf{U}$, it will be offloaded to the relevant fog device f_j, for which $u_i \in \mathbf{C}(f_j)$. The size of a task that is submitted from the mobile device u_i to the fog device f_j is denoted by θ_i. Performing this task requires spending several processor cycles for each bit that is denoted by σ_i. The service rate to perform the task on the fog device f_j is calculated as follows:

$$\mu_{f_j} = \frac{C_{f_j}}{\sigma_i . \theta_i} \quad (8)$$

Similar to Eq. (4), using Little's law gives the local execution time for each task in the fog device f_j as follows:

$$t^l_{f_j} = \frac{L_{f_j}}{\lambda^a_{f_j}} \quad (9)$$

where L_{f_j} denotes the number of system tasks. Now, the local energy consumption of the fog device f_j is calculated as follows:

$$E^l_{f_j} = P^l_{f_j} . t^l_{f_j} = P^l_{f_j} . (t^q_{f_j} + \frac{1}{\mu_{f_j}}) \quad (10)$$

which $P^l_{f_j}$ represents the local power of the fog device f_j . If the computing resources in the fog device are insufficient to process a task, it is offloaded to the central cloud.

4- The proposed method

Q-learning is one of the model-free techniques in RL theory. It is a Finite Markov Decision Process (FMDP) [17]. Here, a *reward function* is the expected reward/penalty, r_{t+1}, for an action a_t performed in a given state s_t. The agent's previous experiences are weighted by the Q-value and then evaluated by the reward function. As shown in Fig. 2, after each action a_t, the agent is moved from the state s_t to a new state s_{t+1} and receives a reward r_{t+1}. This reward reflects the importance of the action and can be the basis for deciding what to do next. The sequences of states in which each agent enters over time can be represented by $(s_0, r_0, a_0, s_1, r_1, a_1, s_2, r_2, a_2, ..., s_t, r_t, a_t)$.

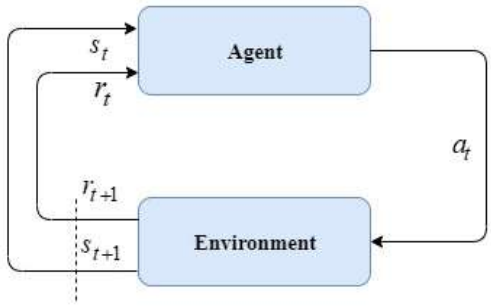

Fig. 2- The role of the environment in Q-learning [7]

Q-learning does not need any information from the environment in previous states to learn the value of an action. Therefore, it can find the optimal action selection policy in an almost random way. The most important elements of the proposed system are as follows:

Agent: For each fog device f_j, all mobile users u_i, which $i \in \mathbf{C}(f_j)$, are the operating agents in the proposed modeling.

State: The state of the mobile user u_i at a time t is denoted by $s_t^i \in \mathbf{S}$, which $\mathbf{S}$ represents the set of all possible states as follows:

$$\mathbf{S} = \{\text{"offload\_pending"}, \text{"local\_pending"}, \text{"running"}\} \quad (11)$$

"local_pending" indicates the state where the task is executed locally on the end device. If the user intends to offload the task to the fog node, he/she goes to the "offload_pending" state. According to the *SPSP* method [2], each user can set his/her own suggested price for each joule of energy consumption (US$/J). The user then sends his/her bid value to the fog machine. After the bid values of all users are sent to the fog machine, the winning user u_w is determined. The fog machine should now allocate computational resources to the winning user. At this moment, the user's state changes from "offload_pending" to "running". The proposed method is pretty fair to loser users, u_{-w}. In other words, they are still allowed to try their chance in the later rounds of competing for resources. Any losing user may remain in his/her current state "offload_pending" to participate in the next auction round. He/she may also refuse to offload. In this case, the state of the user changes from "offload_pending" to "local_pending". Let us denote the number of end devices connected to the fog device f_j by N_{f_j}. Therefore, the state of all users can be represented by a vector $\mathbf{S}_t$. For example, if four users are connected to a fog node, and the third user wins the auction, the state vector of the system is represented by:

$$\mathbf{S}_t = (\text{"offload\_pending"}, \text{"offload\_pending"}, \text{"running"}, \text{"local\_pending"})$$

Now, if the capacity of the fog node is still sufficient, users whose state is "offload_pending" can try again to offload their tasks. For example, if at the next time $t+1$, the second user wins the auction, the state vector changes to

$$\mathbf{S}_{t+1} = (\text{"offload\_pending"}, \text{"running"}, \text{"running"}, \text{"local\_pending"})$$

As is common in FMDP, at least one of the states must be defined as the terminal state $\mathcal{T}$. For a user u_i, if his/her state at the time t, namely s_t^i, belongs to the set $\mathcal{T}$, then its value will not change with any future action. In our problem, the state "running" is terminal, i.e., $\mathcal{T} = \{\text{"running"}\}$. In other words, if a user's task is in "running" state (that is $s_t^i \in \mathcal{T}$), it stays in that state until the task is completed.

Action: The action performed by the mobile user u_i at a time t is denoted by $a_t^i \in \mathcal{A}$. Here, $\mathcal{A}$ is a set of all possible actions as follows:

$$\mathcal{A} = \{\text{"local\_request"}, \text{"offload\_request"}\} \quad (12)$$

The simplest way is to select an action based on a greedy strategy. In this strategy, the action is selected to have the highest value, $a_t^i \doteq \arg\max_{a^i} Q_t\left(s_t^i, a^i\right)$.

Transition Function: It is the probability that an agent u_i will move from the state $s_t^i = s$ to a new state $s_{t+1}^i = s'$ when it acts $a_t^i = a$. It is denoted by a matrix $P(s', r \mid s, a)$, which $P : \mathbf{S} \times \mathbf{A} \times \mathbf{S} \to [0,1]$ and $\sum_{s'} \sum_{r} P(s', r \mid s, a) = 1$.

Reward Function: Depending on what action the agent chooses, it will receive a reward as follows:

$$r_{t+1}^i = \begin{cases} -p_{u_i}.E_{u_i}^l & \text{if } s_{t+1}^i = local\_pending \\ p_{u_i}.E_{u_i}^l - p_{u_i}.E_{u_i}^o - b_{u_i}^{f_j}.E_{f_j}^l & \text{else if } s_{t+1}^i = running \\ 0 & \text{else if } s_{t+1}^i = offloading\_pending \end{cases} \quad (13)$$

As stated in the above relation, if after acting a_t^i, the mobile user u_i is moved to a new state $local\_pending$, he/she will receive a reward $-p_{u_i}.E_{u_i}^l$. Note that the goal of the system here is to minimize the energy consumption of mobile devices. In other words, the proposed algorithm encourages users to offload their tasks to the fog devices instead of performing them locally on their own devices. If the user performs the task locally, he/she will receive a penalty $p_{u_i}.E_{u_i}^l$. Simply speaking, if the user offloaded the task instead of running it locally, he/she could save $p_{u_i}.E_{u_i}^l$ dollars by not consuming valuable local resources. The second condition in the above relation states that if the user is moved to a state "running", he/she will receive a reward $p_{u_i}.E_{u_i}^l - p_{u_i}.E_{u_i}^o - b_{u_i}^{f_j}.E_{f_j}^l$. Here, the first term, $p_{u_i}.E_{u_i}^l$, is the amount of money the user has saved by not performing the task locally. Simply put, if the user wanted to run the task locally instead of offloading it, he/she would have to pay $p_{u_i}.E_{u_i}^l$ dollars for local resource usage. The second term, $-p_{u_i}.E_{u_i}^o$, is the amount of money that must be paid to transmit the task to the BS. The third term, $-b_{u_i}^{f_j}.E_{f_j}^l$, is the amount of money that must be paid to the fog device to perform the task. Note that $b_{u_i}^{f_j}$ is the bid price previously offered by the user u_i to the fog device f_j. As mentioned earlier, in this research we adopt the SPSB auction mechanism proposed by Besharti et al. [2]. When all users submit their bids to the fog device, only one of them who has offered the highest bid (here, the user u_i), will win the auction. Then, the winner user u_i has to pay $b_{u_i}^{f_j}$ dollars for each joule of energy consumption to the fog device. Finally, the last condition in Eq. (13) states that if the user is moved to a state "offloading_pending", he/she will not receive any reward. Although this user tends to offload the task, his/her bid price has not been the highest compared to other competitors. Therefore, the user is temporarily moved to the state "offloading_pending" so that he/she may win the auction in the next time slot.

Based on performing a specific action, the agent is transferred from one state to another. The basic form of the Q-learning algorithm for the agent u_i is $\left(s_t^i, a_t^i, r_{t+1}^i, s_{t+1}^i, a_{t+1}^i\right)$, meaning that the agent was in the state s_t^i, performed the action a_t^i, received the reward r_{t+1}^i, and finally ended up in the state s_{t+1}^i, from where it decided to act a_{t+1}^i. By doing so, it provides a new iteration to update $Q\left(s_t^i, a_t^i\right)$.

The Q-learning algorithm is one of the off-policy RL methods. We adopt the simplest form of it, which is called single-step Q-learning. It is defined as follows:

$$Q\left(s^i\ a^i\right) \quad Q\left(s^i\ a^i\right)$$
$$\lfloor {}_t, {}_t \leftarrow {}_b \quad {}_t, {}_t + \quad \rfloor$$
$$\alpha\lceil r_{t+1}^i + \gamma \max_i Q\left(s_{t+1}^i, b^i\right) - Q\left(s_t^i, a_t^i\right)\rceil \quad (14)$$

where Q and α are the learned action-value function and learning rate, respectively. The α value strikes a balance between the agent's findings from the environment and what he/she has learned. Also, γ denotes the discount factor, which satisfies $0 \le \gamma \le 1$. A lower discount factor encourages the agent to take action sooner rather than postponing it indefinitely. In other words, it determines how much importance should be given to immediate rewards and future rewards. This helps us to avoid infinity as a reward. A value of $\gamma = 0$ means that more importance is given to immediate rewards, and a value of $\gamma = 1$ means that more importance is given to future rewards. In practice, the discount factor of $\gamma = 0$ is never learned, because it only considers immediate rewards, and the discount factor of $\gamma = 1$ continues for future rewards, which may lead to infinity. Therefore, the optimal value for the discount factor, γ, is in the range (0, 1).

After determining the transition function P and the received reward r_{t+1}^{i} by the parent node (controller), the MDP problem can be easily solved using dynamic programming algorithms. Here, the core idea is to use the value function $V(s)$ to find the optimal action a_*^i. The optimal action in any state s_t^i is the action that brings the most reward to the agent. For this purpose, the state value function must be expressed in the following form, which is known as the Bellman equation:

$$V_*(s) = \max \mathrm{E}(r_{t+1}^{i} + \gamma . V_*(s_{t+1}^{i}) \mid s_t^i = s,\ a_t^i = a) \tag{15}$$

The above formula, after simplification, can be rewritten as follows:

$$V_*(s) = \max_a \sum_{s',r} P(s', r \mid s, a)[r + \gamma . V_*(s')] \tag{16}$$

One of the most common ways to solve the Bellman equation is to rewrite it in the following recursive form:

$$V_{t+1}(s) = \max_a \sum_{s',r} P(s', r \mid s, a)[r + \gamma . V_t(s')] \tag{17}$$

Algorithm 1 Pseudo-code of the Q-learning for offloading

Input: The fog node identifier f_j, the set of states $\mathcal{S}$, the set of actions , the set of terminal states $\mathcal{T}$

Output: the optimal action-value function vector $\mathbf{Q}_*$

Set $t = 0$
for $i \in \mathbf{C}(f_j)$ ***do***
  for $s_t^i \in \mathcal{S}$ ***do***
    for $a_t^i \in \mathcal{A}$ ***do***
      Initialize $Q(s_t^i, a_t^i)$ arbitrarily
      Initialize terminal state value, $Q(\mathcal{T}^i, .) = 0$
    end for
  end for
end for
repeat
  Initialize $\mathbf{S}_t$
  repeat (for each step of the episode)
    Choose $\mathbf{A}_t$ from $\mathbf{S}_t$ using policy derived from $\mathbf{Q}_t$ (e.g., $\mathcal{E}$-greedy)
    Take action $\mathbf{A}_t$, observe $\mathbf{R}_{t+1}$, $\mathbf{S}_{t+1}$
  $\mathbf{Q}(\mathbf{S}_t, \mathbf{A}_t) \leftarrow \mathbf{Q}(\mathbf{S}_t, \mathbf{A}_t) + \alpha\left[\mathbf{R}_{t+1} + \gamma \max_{\mathbf{B}} \mathbf{Q}(\mathbf{S}_{t+1}, \mathbf{B}) - \mathbf{Q}(\mathbf{S}_t, \mathbf{A}_t)\right]$ using Eq. (14)
  $\mathbf{S}_t \leftarrow \mathbf{S}_{t+1}$
  until $\mathbf{S}_t \in \mathcal{T}$
  $t \leftarrow t + 1$
until *Max_Num_Episodes*
$\mathbf{Q}_* \leftarrow \mathbf{Q}(\mathbf{S}_t, \mathbf{A}_t)$ using Eq. (18)
return $\mathbf{Q}_*$

Thus, the value of all states can be obtained. Given that $0 \le \gamma \le 1$, it can be proved that the Bellman equation will converge and, therefore, the solution is as follows:

$$V_*(s) = \lim_{t \to \infty} V_t(s) \tag{18}$$

The critical advantage of Q-learning over other RL methods is that it converges very quickly. The main reason is that Q can directly approximate the optimal

action-value function, q_*. Here, the policy determines which state-action pairs to visit and update. Like most RL methods, the prerequisite for convergence here is that all pairs continue to be updated. Under this assumption and other common indefinite approximation conditions in the sequence of size-step parameters, it is found that Q converges with a probability of 1 to q_*. Another advantage of Q-learning is that it does not require a specific environment model. In RL terminology, it is a model-free method and does not require a predetermined policy to find any optimal state-action pair. The pseudo-code of the Q-learning for the offloading problem is shown in *Algorithm 1*.

5.Performance Evaluation

We use the *iFogSim* [20] simulator to evaluate the efficiency of the proposed method, *SPSB_Auction_RL*. We compare the *SPSB_Auction_RL* with *FCFS* method [2]. It is the base method used in *iFogSim*. Here, the task is offloaded to the parent fog node once the buffer becomes full. As described in the previous section, *SPSB_Auction_RL* uses a second-price sealed-bid auction to offload tasks. The inter-arrival time of tasks has an exponential distribution with a rate of 50 tasks per minute. Also, the queue capacity of each fog node is 20. All experiments were repeated 40 times, and their average was calculated for each offloading criterion.

Fig. 3 shows the total energy consumption. It is equal to the sum of the energy consumed in the local execution and the energy consumed for offloading. The local and offloading energy is calculated using Eq. (10) and Eq. (7), respectively. As shown in Fig. 3, energy consumption increases in both methods when the number of fog devices increases. However, *SPSB_Auction_RL* has lower energy consumption compared to *FCFS*.

Fig. 4 shows the execution time for different numbers of fog nodes. Note that the number of fog nodes is insignificant in a real-world environment. Previous research has also considered the number of fog nodes low [4]. Also, note that the computing capacity of a typical fog device is much less than a dedicated cloud server. As seen in Fig. 4, the average execution time of both methods increases with the number of nodes. However, the growth of execution time in *SPSB_Auction_RL* is always linear, while the growth of time in *FCFS* is exponential. This presents an interesting implication for network designers in terms of scalability.

Fig. 5 shows the network usage for different numbers of fog nodes. This is one of the essential metrics produced in *iFogSim* reports. It is the amount of data transferred to offload all tasks in the network. As shown in Fig. 5, network usage increases with the number of fog devices. However, the network usage in the *SPSB_Auction_RL* method is significantly lower than that of the FCFS.

Fig. 6 shows the buffering cost of executing commands on the cloud node. The buffering cost in the proposed method is slightly higher than in the FCFS. The reason lies behind the execution of auction operations between each node and its children in discrete time intervals. Reinforcement learning algorithms impose more overhead on the fog operating environment. Of course, this amount of overhead is negligible and can never be a reason for the impracticality of the proposed method. It is still far superior to FCFS in energy consumption and latency.

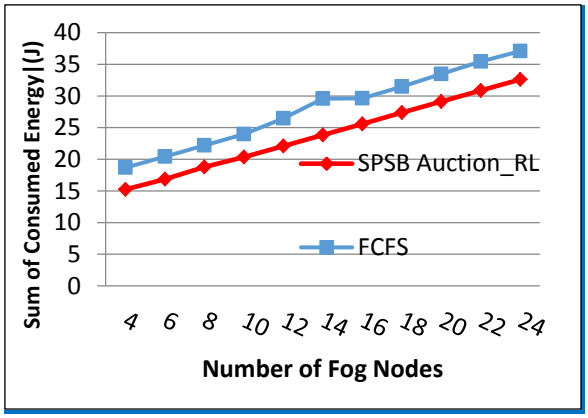

Fig. 3- Energy consumption for different numbers of fog nodes

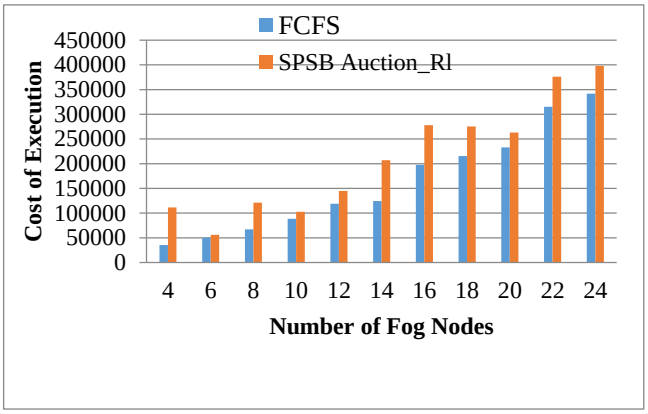

Fig. 6 Cost of execution for different numbers of fog nodes

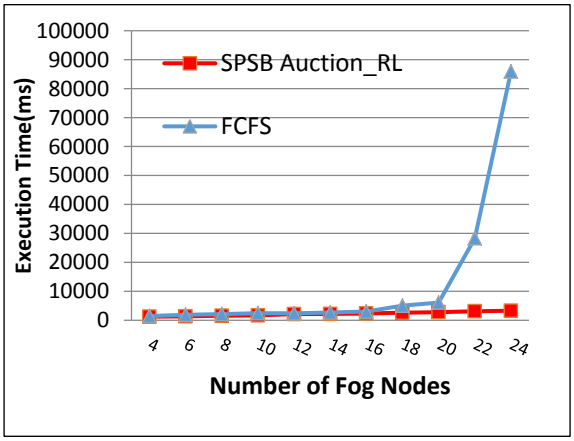

Fig.4 Execution time for different numbers of fog nodes

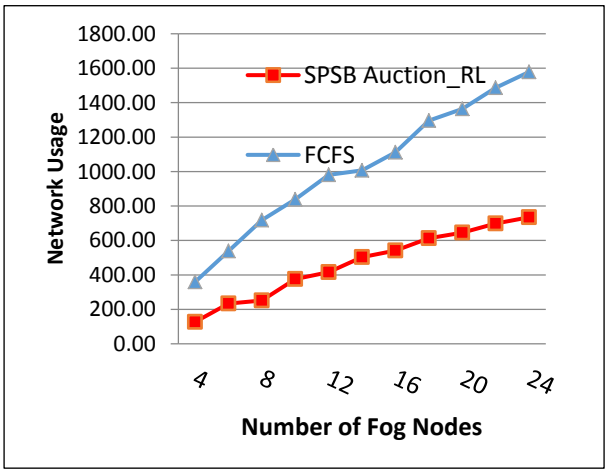

Fig. 5 Network usage for different numbers of fog nodes

6- Conclusion

This paper proposed a Q-learning-based method to solve the offloading optimization problem in fog computing. After modeling the system with queuing theory, the joint energy and delay minimization problem was formulated. Our price-based Q-learning technique motivates users to participate in offloading operations. It also does not ignore the chances of users who lose in the auction participating in subsequent rounds.

The performance of the proposed method was evaluated against the baseline method, namely *FCFS*. The simulation results showed that the proposed method significantly reduces the execution time compared to the *FCFS* method. Also, energy consumption and network usage show a significant amount of improvement. The proposed method is more stable than *FCFS* due to the lower variance of energy consumption.

One of the future research trends is to consider user mobility and handoff. Also, using other reinforcement learning methods in combination with multi-objective meta-heuristics such as NSGA may significantly affect convergence.

References

[1] Keshavarznejad, M., Rezvani, M.H. and Adabi, S.,: Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms. *Cluster Computing*, pp.1-29 (2021)

[2] Besharati, R., Rezvani, M.H. and Sadeghi, M.M.G., 2021. An Incentive-Compatible Offloading Mechanism in Fog-Cloud Environments Using Second-Price Sealed-Bid Auction. *Journal of Grid Computing*, *19*(3), pp.1-29.

[3] Li, Q., Zhao, J., Gong, Y. and Zhang, Q.,: Energy-efficient computation offloading and resource allocation in fog computing for internet of everything. *China Communications*, *16*(3), pp.32-41 (2019)

[4] Khoobkar, M.H., Dehghan Takht Fooladi, M., Rezvani, M.H. and Gilanian Sadeghi, M.M., 2022. Partial offloading with stable equilibrium in fog-cloud environments using replicator dynamics of evolutionary game theory. *Cluster Computing*, *25*(2), pp.1393-1420.

[5] Besharati, R. and Rezvani, M.H., : February. A prototype auction-based mechanism for computation offloading in fog-cloud environments. In *2019 5th conference on knowledge based engineering and innovation (KBEI)* (pp. 542-547). IEEE (2019)

[6] Jafari, V and Rezvani, M.H., Joint Optimization of Energy Consumption and Time Delay in IoT-Fog-Cloud Computing Environments using NSGA-II Metaheuristic Algorithm. *Journal of Ambient Intelligence and Humanized Computing, in press,(2021)*

[7] Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.

[8] Lei, L., Tan, Y., Zheng, K., Liu, S., Zhang, K. and Shen, X., 2020. Deep reinforcement learning for autonomous internet of things: Model, applications and challenges. *IEEE Communications Surveys & Tutorials*, *22*(3), pp.1722-1760.

[9] Santos, J., Wauters, T., Volckaert, B. and De Turck, F., 2021. Resource provisioning in fog computing through deep reinforcement learning.

[10] Gazori, P., Rahbari, D. and Nickray, M., 2020. Saving time and cost on the scheduling of fog-based IoT applications using deep reinforcement learning approach. *Future Generation Computer Systems*, *110*, pp.1098-1115.

[11] Rahman, G.S., Dang, T. and Ahmed, M., 2020. Deep reinforcement learning based computation offloading and resource allocation for low-latency fog radio access networks. *Intelligent and Converged Networks*, *1*(3), pp.243-257.

[12] Jazayeri, F., Shahidinejad, A. and Ghobaei-Arani, M., 2021. Autonomous computation offloading and auto-scaling the in the mobile fog computing: a deep reinforcement learning-based approach. *Journal of Ambient Intelligence and Humanized Computing*, *12*(8), pp.8265-8284.

[13] Chen, M., Wang, T., Zhang, S. and Liu, A., 2021. Deep reinforcement learning for computation offloading in mobile edge computing environment. *Computer Communications*, *175*, pp.1-12.

[14] Baek, J. and Kaddoum, G., 2020. Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks. *IEEE Internet of Things Journal*, *8*(2), pp.1041-1056.

[15] Talaat, F.M., Saraya, M.S., Saleh, A.I., Ali, H.A. and Ali, S.H., 2020. A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment. *Journal of Ambient Intelligence and Humanized Computing*, pp.1-16.

[16] Naghdehforoushha, M., Dehghan Takht Fooladi, M., Rezvani, M.H., Gilanian Sadeghi,M.M.,, 2022, BLMDP: A New Bi-level Markov Decision Process Approach to Joint Bidding and Task-Scheduling in Cloud Spot Market, Turk J Elec Eng & Comp Sci, DOI: 10.3906/elk-2108-89

[17] Naghdehforoushha, M., Fooladi, M.D.T., Rezvani, M.H. and Sadeghi, M.M.G., 2022. BLMDP: A new bi-level Markov decision process approach to joint bidding andtask-scheduling in cloud spot market. *Turkish Journal of Electrical Engineering and Computer Sciences*, *30*(4), pp.1419-1438.

[18] Fang, C., Xu, H., Yang, Y., Hu, Z., Tu, S., Ota, K., Yang, Z., Dong, M., Han, Z., Yu, F.R. and Liu, Y., 2022. Deep-Reinforcement-Learning-Based Resource Allocation for Content Distribution in Fog Radio Access Networks. *IEEE Internet of Things Journal*, *9*(18), pp.16874-16883.

[19] Shruthi, G., Mundada, M.R., Sowmya, B.J. and Supreeth, S., 2022. Mayfly taylor optimisation-based scheduling algorithm with deep reinforcement learning for dynamic scheduling in fog-cloud computing. *Applied Computational Intelligence and Soft Computing*, *2022*.

[20] Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K. and Buyya, R.,: iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience*, *47*(9), pp. 1275-1296 (2017)