

A Framework of an RL-Based Task Offloading Mechanism for Multi-users in Edge Computing

Shintaro Ide and Bernady O. Apduhan(✉)

Graduate School of Information Science, Kyushu Sangyo University, Fukuoka, Japan
bob@is.kyusan-u.ac.jp

Abstract. In recent years, the Internet of Things (IoT) has been growing rapidly, and new applications using the IoT includes autonomous driving systems, mobile health, smart homes, VR/AR technologies, and many more. While IoT applications plays important roles in enriching our lives, the size of each generated task is increasing compared to tasks generated by traditional mobile devices, which can cause high latency when deployed in traditional cloud computing. Edge computing has emerged as a method to mitigate this problem. Edge computing can provide services with lower latency than conventional cloud computing methods because the tasks are processed in the vicinity of the user's device. However, when the tasks are concentrated in one of the distributed servers, the low processing power of these servers becomes a bottleneck and high latency may occur.

In this paper, we proposed a deep reinforcement learning based offloading mechanism in an edge computing environment which can dynamically offload a task based on the performance and availability of nearby edge servers. Preliminary experiment results are promising and offered insights on related issues.

Keywords: Edge computing · Deep reinforcement learning · Task offloading

1 Introduction

Over the past decade, mobile communication systems have evolved from the 3rd and 4th generation (3G and 4G) to today's 5th generation (5G). 5G networks are characterized by extremely high speed, high capacity, low latency, and multiple simultaneous connections compared to previous generations of mobile communication systems. It is said that 5G networks will become a social infrastructure to further develop and promote the spread of existing IoT devices and applications. Examples of IoT devices or applications that are expected to develop in the future include VR/AR devices, mobile health, autonomous driving systems, and a wide range of other fields [1]. However, these intelligent devices and applications can only be realized by processing huge amounts of data in real time, which poses a great challenge for servers and network bandwidth.

In conventional cloud computing systems, data generated from edge devices such as IoT devices are processed by high-performance servers in a cloud data center. However, when the physical distance between the data source and the server is far, high latency is likely to occur [2]. Edge computing systems have emerged as a new technology to meet

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022 Workshops, LNCS 13379, pp. 103–114, 2022.
https://doi.org/10.1007/978-3-031-10545-6_8

the needs of real-time and low latency requirements. By distributing edge servers with a certain amount of computational power near the edge devices, edge computing can significantly reduce the response time for requested tasks and also reduce the load on cloud data centers. Edge computing is attracting attention as the next-generation computing paradigm. However, to operate it efficiently, it is necessary to consider task allocation and offloading. Task offloading here refers to uploading computationally intensive applications that cannot be processed by the local edge server and have the other server take over the processing. In general, the following cases can be considered as types of task offloading.

1. Local Execution
   It completes all tasks generated by the edge devices using only its own computational power.
2. Full Offloading
   All the generated tasks are offloaded to other edge servers, and the processing is completed by the computational power of these edge servers.
3. Partial Offloading
   Part of the task is processed locally, and the rest is offloaded to another edge server. This can be done only when the tasks can be divided.

In this paper, we study a dynamic task offloading mechanism for efficient edge computing system operation. To achieve dynamic task distribution, this paper employs reinforcement learning, which is one of the machine learning methods. Each edge device is modeled as a Markov Decision Process, and a task offloading algorithm based on multitasking and load balancing is proposed.

2 Related Work

For efficient task offloading, it is necessary to consider when, where, and to what extent tasks should be offloaded. Several studies have already been done on the optimal task offloading strategy for different scenarios.

Gao et al. developed a cooperative computing system consisting of mobile devices, edge cloud and central cloud and designed a Q-Learning based task offloading policy for optimal resource allocation and offloading based on it [3].

Mochizuki et al. focused on mobile data offloading, which is an effort by telecommunication carriers to meet the increasing demand for mobile data communications. They propose a mobile data offloading method based on distributed deep reinforcement learning [4], in contrast to the temporal offloading of existing methods (Mobile Data Offloading Protocol) to maximize bandwidth utilization efficiency.

Shino et al. proposed an autonomous distributed task offloading method in which applications autonomously determine the optimal execution node and offload tasks to provide distributed processing that can cope with the increasing number of moving applications and topology changes among nodes in edge computing environments [5].

These studies are like our work in that they adapt reinforcement learning-based algorithms for optimal task assignment and task offloading for user devices in edge

computing environments. On the other hand, they only make discrete behavioral decisions for task offloading and do not support handling continuous variables that allow detailed parameter decisions. In this study, we propose a method to solve not only the discrete problem of deciding which server to offload to and determining task allocation, but also the continuous problem of making detailed decisions on migration bandwidth and other parameters during offloading.

3 System Model

In this paper, the tasks are generated from multiple user devices e = {1, 2, 3…}. Here, we assume a general edge computing scenario where tasks are generated by multiple user devices e = {1, 2, 3…, n} and processed by multiple edge servers E = {1, 2, 3…, N} as shown in Fig. 1. Each user device ranges from common computing devices such as desktop and laptop computers to web cameras and wearable devices used for mobile health monitoring. Each of these devices generates a different size of demanding tasks.

In addition, the computing power of user devices themselves varies. If a user device chooses to offload a task to an edge server, each user device thinks only of its own benefit and tries to offload the task to the edge server with the highest computational power. However, even a high-performance edge server cannot process all these tasks in real time due to its limited computing power. It may not be able to allocate the necessary resources in time, which may cause a large delay.

Therefore, task offloading in edge computing environments is important to balance the distribution of tasks and to increase the utilization rate of each edge server to a certain value.

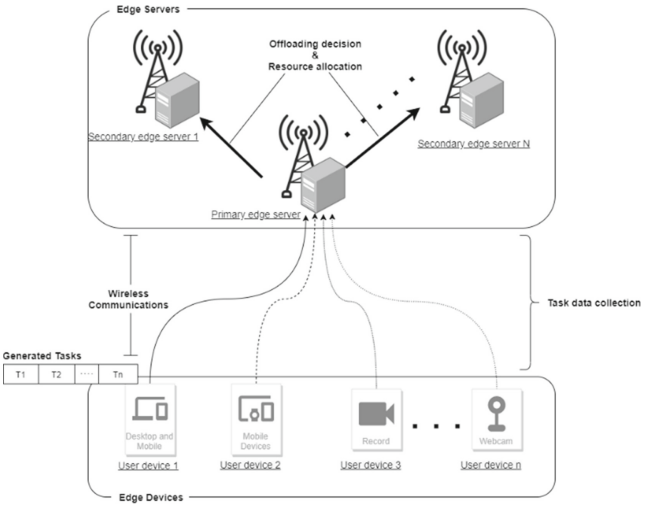

Fig. 1. An RL-based task offloading mechanism in edge computing

The Proposed Method

Deep Q-Network (DQN), a deep reinforcement learning algorithm, is one of the most effective methods for task offloading in edge computing environments. DQN is a useful method for complex resource allocation problems and high-dimensional state spaces, but it does not address the problem of continuous action spaces, and action decisions must always be made using discrete values.

In this paper, we proposed a task offloading mechanism that makes continuous action decisions in a single-agent scenario with a primary edge server as an agent. Deep Deterministic Policy Gradient (DDPG), a deep reinforcement learning algorithm based on the Actor-Critic model, is employed to realize complex decision making by the agent.

4 Reinforcement Learning – A Recap

Reinforcement learning [6, 7] is a machine learning method. It is characterized by the fact that the system itself achieves optimal system control through trial and error. It differs from supervised learning in that it does not require artificial labeling of the teacher data (correct answer data). In reinforcement learning, the computer learns behaviors to maximize the reward set as the objective in a certain environment. After a certain number of trials, it is possible to learn a policy that maximizes the long-term reward for the current environment. Furthermore, by combining this method with deep learning, it is possible to adapt to high-dimensional states and behaviors, which greatly improves the learning efficiency.

In this paper, we use deep reinforcement learning for efficient resource allocation for optimal task offloading in edge computing environments. In reinforcement learning, the dynamics of the environment are typically modeled by a Markov Decision Process (MDP), and the algorithm is analyzed. Likewise, we establish the MDP and define the following three elements to adapt the deep reinforcement learning algorithm to the edge computing environment.

- **State**
  The current state S_t is defined as the system load rate on each edge server. Each edge server queues up a predetermined number of tasks in its own data structure and processes them in turn.
- **Action**
  In the current state S_t, the action A_t taken by the system refers to the decision of the primary edge server on the task offload vector, task allocation, and migration bandwidth during offload. The task offload vector and task allocation decisions are discrete values, while the migration bandwidth decision is treated as a continuous value.
- **Reward**
  The system-wide reward R_t is defined as the total number of tasks processed by each edge server for each trial episode in reinforcement learning.

4.1 Markov Decision Process (MDP)

MDP is a stochastic model of a dynamic system in which state transitions occur stochastically, and in which state transitions satisfy Markovianity. Markovianity means that the transition to the next state depends only on the current state and action but is independent of earlier states and actions. By modeling the representation of time-series state transitions in reinforcement learning with MDPs, we can maximize the cumulative reward by simply finding the value function applied by the MDP [8].

4.2 Value Function

In reinforcement learning, the reward is a measure of the immediate goodness or badness of an action chosen from the current state, but the decision to act must consider the reward that will come later. In reinforcement learning, the cumulative reward obtained over a period is called the revenue. There are several types of revenue, but the most used revenue is the discounted cumulative reward which is expressed by discounting the value of the reward to be obtained later, as shown in Eq. (1).

$$G_t = \sum_{\tau=0}^{\infty} \gamma^{\tau} R_{t+1+\tau} 0 < \gamma < 1 \tag{1}$$

The sum of the discounted rewards is the sum of the rewards for a given interval, and the content of the interaction is determined stochastically depending on the state at the start of the interval. As a result, the revenue also becomes a value that varies stochastically. Therefore, the expected value of the revenue is obtained conditionally on the state, and the expected value is called the state value, which is used as an index for defining a good strategy.

The state value is the sum of the rewards for continuing to take actions according to the strategy π, starting from the state S_t, and can be expressed as in Eq. (2).

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[G_{t+1} | S_t = s\right] \tag{2}$$

The best strategy is called the optimal strategy π^* and is given by Eq. (3).

$$V^*(s) = V^{\pi^*}(s) = maxV^{\pi}(s) \tag{3}$$

Here, V^* (s) is called the optimal state value function. To improve convenience when making action decisions, we use a function called the action value function, which adds actions to the conditions in Eq. (3) above. The action value function represents the value of each action A_t in each state S_t, as shown in Eq. (4).

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[G_{t+1} | S_t = s, A_t = a\right] \tag{4}$$

4.3 Deterministic Policy Gradient (DPG)

Deterministic policy gradient is a reinforcement learning algorithm presented by Silver et al. in 2014 [9]. The DPG algorithm is a policy gradient-based algorithm, which models the policy with parameter θ and performs direct optimization. The DPG algorithm uses a reinforcement learning approach called Actor-Critic, which is described later.

4.4 Actor-Critic

The DPG algorithm is a reinforcement learning algorithm based on the Actor-Critic model, which has a separate structure for representing policies, independent of the value function, as shown in Fig. 2. The structure that represents the policy is called an Actor, because it is used to select actions, and is expressed as Eq. (5), because it is a function of deterministic measures whose actions are deterministic for a given state.

$$\mu(s|\theta^{\mu}) \tag{5}$$

Here, θ^{μ} is a parameter of the measure function μ.

The part that predicts the value function is called Critic because it critiques the actions chosen by the Actor [10]. Critic is synonymous with the action value function and is therefore expressed as in Eq. (6).

$$\mathrm{Q}(\mathrm{s}, \mathrm{a}\left|\theta^{\mathrm{Q}}\right.) \tag{6}$$

where θ^{Q} is a parameter of the action value function Q.

Since the strategy μ only needs to learn to maximize the current value, we can optimize θ^{μ} using the gradient shown in Eq. (7).

$$\begin{aligned} \nabla_{\theta^{\mu}} J &\approx \mathbb{E}\left[\nabla_{\theta^{\mu}} Q\left(s, a|\theta^{Q}\right)|_{s=s_t, a=\mu(s|\theta^{\mu})}\right] \\ &= \mathbb{E}\left[\nabla_{\theta^{\mu}} Q\left(s, a|\theta^{Q}\right)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}\right] \end{aligned} \tag{7}$$

In addition, the action value function Q can be defined as the error function in Eq. (8) from the Bellman equation [11], which holds for action values.

$$\begin{aligned} L\left(\theta^{Q}\right) &= \mathbb{E}\left[\left(Q\left(s, a|\theta^{Q}\right) - y_t\right)^2\right] \\ \textit{where } y_t &= r(s_t, a_t) + \gamma Q\left(s_{t+1}, \mu(s_{t+1})|\theta^{Q}\right) \end{aligned} \tag{8}$$

The behavioral value function Q can be optimized by learning to minimize the error function calculated in Eq. (4) above.

As described above, in the DPG algorithm, the optimal strategy function μ^* can be obtained by simultaneously optimizing the parameter θ^{μ} of the strategy function μ and the parameter θ^{Q} of the action value function Q during the learning process.

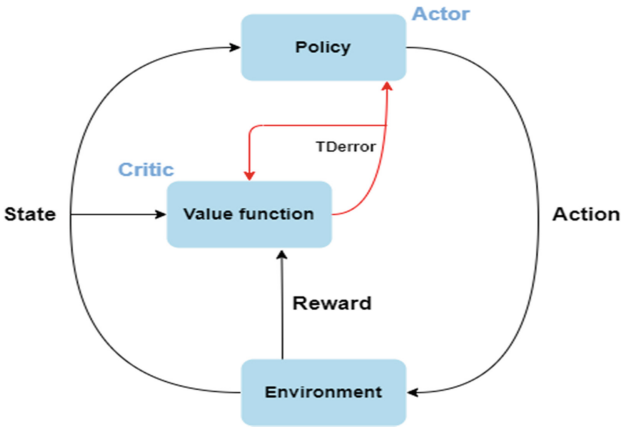

Fig. 2. Overview of actor-critic

5 Deep Reinforcement Learning

Deep reinforcement learning refers to the use of neural networks for function approximation of the value function and the policy function in reinforcement learning. In conventional reinforcement learning, linear functions are used to approximate the functions of value and policy functions. In addition, the design of the features requires human intervention. In deep reinforcement learning, however, the neural network designs the features, eliminating the need for human intervention. In addition, it was confirmed that deep reinforcement learning can significantly outperform conventional reinforcement learning simply by providing the environment directly as input [12, 13].

5.1 Deep Deterministic Policy Gradient (DDPG)

The first innovation is a method called Experience Replay [15], proposed by L. Lin in 1992. Experience Replay is a method of learning by memorizing the past actions of an agent and randomly selecting multiple memorized actions at regular intervals. This method is called Experience Replay. The number of randomly selected actions is called the batch size. By learning randomly in this way, it is possible to control the bias of the samples and to learn the same training data repeatedly, thus improving the efficiency of sample usage.

The second innovation is the fixed target Q-network. In a strategy-gradient type reinforcement learning algorithm such as DDPG, the parameters of each function are updated in the direction of optimizing the strategy function and the action value function. The fixed target Q-network can stabilize the learning by updating some parameters instead of the whole parameters. The fixed target Q-network can stabilize the learning by updating some parameters instead of the whole parameters.

The fixed target Q-network can stabilize learning by updating some of the parameters instead of updating the entire parameters. Let θ^{μ} and θ^{Q} be the parameters of the strategy function and action value function before the update, respectively, and $\theta^{\mu'}$ and $\theta^{Q'}$ after

the update (Fig. 3).

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$

$$\tau \ll 1 \tag{9}$$

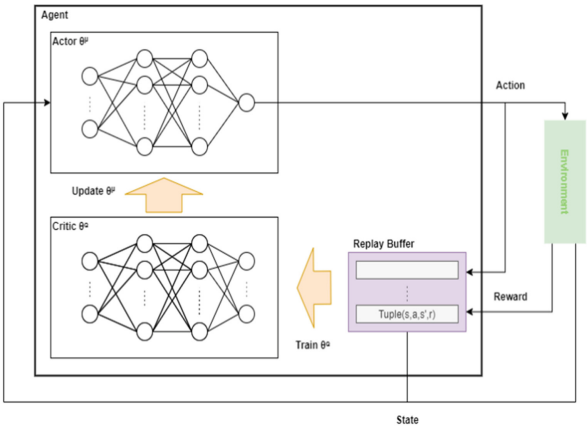

Fig. 3. The DDPG architecture

6 Experiments

In this paper, we conduct experiments to evaluate whether the task offloading mechanism learns according to the DDPG algorithm and allocates tasks to the optimal edge servers.

6.1 Experiment Methods

The simulation experiment was conducted on a virtual edge computing environment. The open-source CRAWDAD mobility dataset was used as the training dataset. This dataset shows mobility devices such as smartphones and wearable devices generating tasks at regular intervals, as shown in Fig. 4. In this experiment, we evaluate the proposed model by observing the offloading of data from this mobility dataset to multiple edge servers and measuring the rewards obtained from the environment. First, the simulations were performed with parameters such as the number of edge servers and the number of user terminals. Next, we simulate the task offloading process using a task offloading mechanism based on the DDPG algorithm. Three patterns were assumed here, i.e., 30, 50, and 80 edge devices, to evaluate the generality of the model. The parameters for each simulation are shown in Table 1. The number of training trials varies dynamically

depending on the training status of the model. The specification is that if the reward obtained from the environment does not increase by the specified number of episodes, learning terminates at that point. In addition, in one episode, the movement of mobility data is divided into 3000 steps and executed. The reward defined in Sect. 5 was used as the evaluation index.

Table 1. Simulation parameters

Parameter	Value
Number of user devices (e)	50
Number of edge servers (E)	10
Maximum number of server processes (f)	5
Generation task size (r)	0–50 MB
Migration bandwidth (b)	0–1 Gbps
Number of training episodes (ep)	130
Number of steps in each episode (st)	3000
Experience replay buffer size (mem_cap)	10000
Learning rate for critic network (lr_c)	1.0×10^{-3}
Learning rate for actor network (lr_a)	2.0×10^{-3}
Bach size (batch)	32
Soft update rate for target network (tau)	1.0×10^{-2}

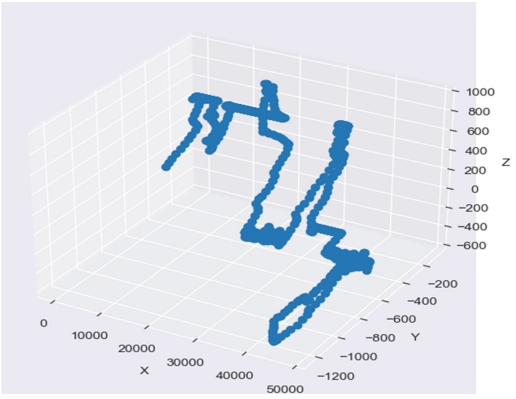

Fig. 4. The CRAWDAD mobility dataset

6.2 Experiment Results

Using the set parameters, we conducted a task offloading experiment and observed the changes in the reward obtained from the environment. The results are shown in Figs. 5 and 6.

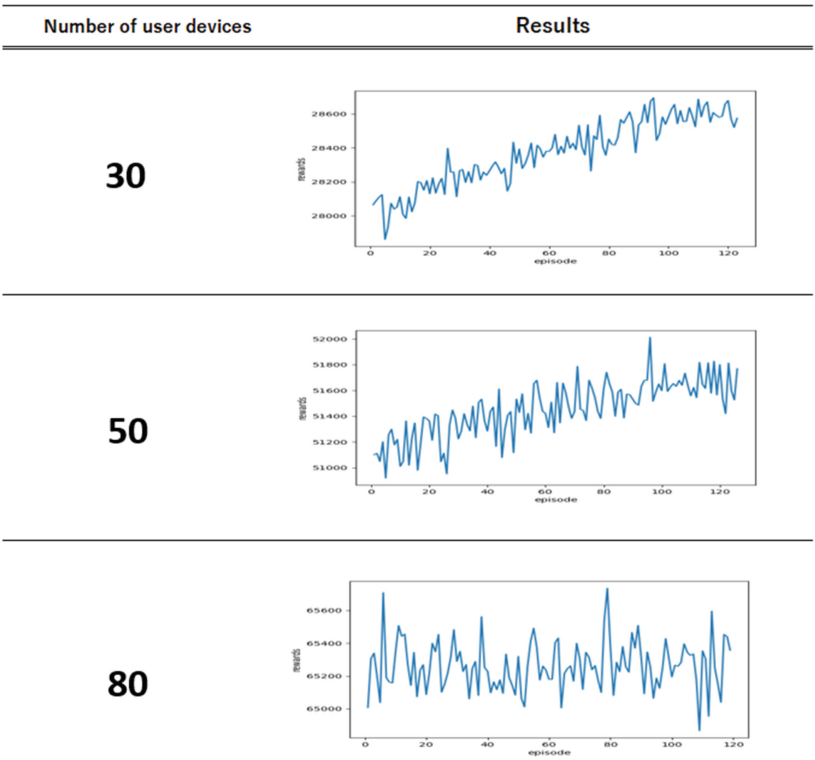

Fig. 5. Rewards from the environment per group of devices.

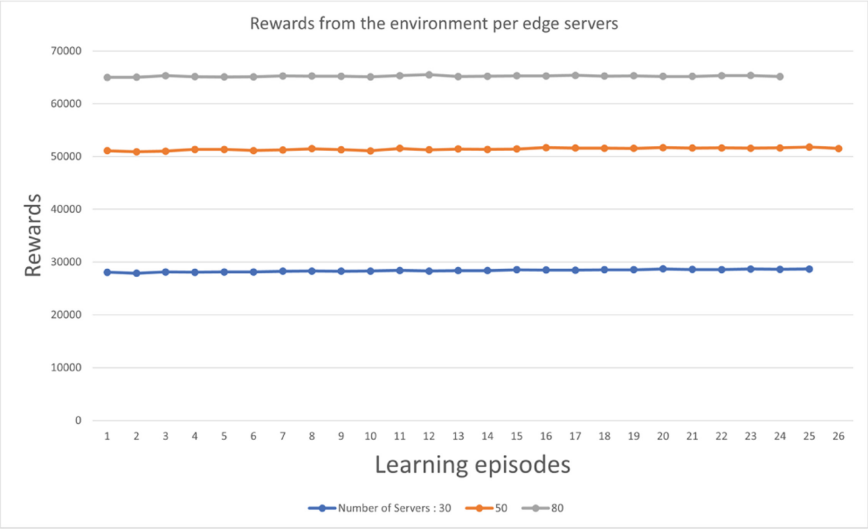

Fig. 6. Rewards from the environment per group of devices (another view)

6.3 Discussion of Results

From Fig. 5 of the experimental results, the number of system-wide task processes set as rewards varied with the learning episodes. We will look at the patterns in order of the number of user devices, starting with the lowest number. In an environment with 30 user devices, the reward tends to increase continuously, which indicates that the proposed mechanism in the environment can offload tasks to the edge server appropriately for the user devices. The same is true for the case of 50 user devices, but we can observe that the range of rewards is lower than in the case of 30 user devices. This may indicate that some episodes may have dropped out because they could not offload the task appropriately. Finally, in the case of the 80 user devices, some episodes may have high rewards, but the overall rewards are highly variable, and it is difficult to say that the rewards are on an increasing trend. This may be due to too many user devices offloading tasks relative to the number of edge servers in the experimental environment. Since all edge servers in this environment have the same performance and the number of tasks that can be processed is fixed, it is practically impossible to efficiently process too many tasks. This issue needs to be addressed in future experiments and find a solution.

7 Concluding Remarks

In this paper, we described an optimal offloading mechanism for multi-users in edge computing. Specifically, using the DDPG algorithm, which is a method of deep reinforcement learning. We define multiple user devices in the edge computing environment as agents and proposed a mechanism to offload the generated task data to the optimal edge server according to the DDPG algorithm.

In our experiments, we built and simulated a virtual edge computing environment to verify whether the proposed mechanism allocates tasks appropriately. From the experimental results, we confirmed that as learning progressed, the mechanism started to offload tasks appropriately, and the number of tasks processed by the entire system increased.

As future work, it is necessary to conduct comparison experiments with other existing methods to evaluate the usefulness of the offloading mechanism proposed in this paper. In addition, the current environment does not consider the computational power of the user device itself. Therefore, of the three types of task offloading introduced in Sect. 1, only full offloading is being performed. In the future, it will be necessary to construct a new edge computing environment that considers the two offloading methods of local execution and partial offloading to make it closer to the real environment.

References

1. Ashton, K.: That "Internet of Things." Thing Adv. Internet Things **6**(4), 97–101 (2009)
2. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener. Comput. Syst. **25**(6), 599–616 (2009)
3. Gao, Z., Hao, W., Han, Z., Yang, S.: Q-learning-based task offloading and resources optimization for a collaborative computing system. IEEE Access **8**, 149011–149024 (2020)
4. Mochizuki, D., Abiko, Y., Mineno, H., Saito, T., Ikeda, D., Katagiri, M.: Deep reinforcement learning-based method of mobile data offloading. In: 2018 Eleventh International Conference on Mobile Computing and Ubiquitous Network (ICMU), No. 18493208, Auckland, New Zealand, 5–8 October (2018)
5. Shino, Y., Sasai, K., Kitagata, G., Kinoshita, T.: Edge computing using autonomous decentralized task offload. IPSJ Res. Rep. **2017-MBL-85**(26), 1–7 (2017)
6. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, Cambridge (2014, 2015)
7. Makino, T., et al.: Future Strengthening Learning. Morikita Publishing (October 2016)
8. Kimura, G.: Basics of Reinforcement Learning. J-STAGE **52**(1), 72–77 (2013)
9. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: 31st International Conference on Machine Learning, vol. 32, pp. 387–395 (June 2014)
10. Geron, A. (Author), Shimoda, R. (Translation), Nagao, T. (Translation): Practical Machine Learning with Scikit-Learn, Keras, TensorFlow. O'Reilly, Japan (October 2020). (Translated in Japanese)
11. Szepesvari, C. (Author), Oyamada, S. (Editor), Maeda, S., Koyama, M. (Translation): Algorithms for Reinforcement Learning Rapid Learning-Basic Theory and Algorithms, pp. 8–12, Kyoritsu Shuppan, Tokyo (July 2017)
12. Volodymvr, M., et al.: Playing Atari with Deep Reinforcement Learning. arXiv: 1312.5602 (2013)
13. Volodymvr, M., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
14. Lillicrap, T.P., et al.: Continuous control with Deep Reinforcement Learning. arXiv: 1509.02971 (July 2019). (latest version)
15. Long-Ji, L.: Reinforcement Learning for Robots Using Neural Networks. No. CMU-CS-93–103, Carnegie-Mellon University PA School of Computer Science (1993)