

Development of an RL-Based Mechanism to Augment Computation Offloading in Edge Computing

Shintaro Ide(✉) and Bernady O. Apduhan(✉)

Graduate School of Information Science, Kyushu Sangyo University, Fukuoka, Japan
k20gjk01@st.kyusan-u.ac.jp, bob@is.kyusan-u.ac.jp

Abstract. The explosive growth of data generated by the widespread use of IoT devices and the increasing realizations of IoT applications that require real-time responses have made it difficult for traditional cloud computing or edge computing to keep up with the tasks processing demands and/or near real-time response requirements of applications. We employ the strategy of computation offloading to nearby edge nodes to meet these requirements on time. In this research, we developed an efficient offload broker mechanism using deep reinforcement learning to perform optimal task allocation and computation offloading on this platform. Experiments show that the model learns the policies for offloading tasks to the optimal nodes appropriately. These promising results will enlighten more computation offloading issues to improve the efficiency of the model and its deployment in edge computing environment.

Keywords: Edge computing · Computation offloading · Reinforcement learning

1 Introduction

With the increasing realization of IoT (Internet of Things) applications, large amount of data is being generated every day [1]. One such example is the video data taken from the many surveillance cameras installed throughout a city. In the event of an unexpected incident or accident, these camera images are reviewed, verified, and presented as evidence. However, due to the large amount of generated video data, the old data is often discarded without being utilized. To this, by using edge computing and AI technologies, we envisioned to utilize this data and investigate its potential for the prevention and countermeasures against untoward incidents and accidents.

Edge computing [2] is an evolving technology wherein the data is processed at the edge servers which are close to the data source. It has the potential to significantly reduce the processing dependency on cloud data centers to process and support the near real-time responses required by applications. Edge computing complements cloud computing. However, the computing power of edge servers are limited compared to servers at data centers. Due to this limited computing power of edge servers, there is the risk of losing real-time performance when edge servers are processing compute-intensive applications.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12954, pp. 539–548, 2021.
https://doi.org/10.1007/978-3-030-86979-3_38

In this research, we study an efficient computation offloading [3, 4] strategy and develop a computation offloading mechanism (a.k.a, offloading broker) to support the near real-time processing requirements of applications in an edge computing environment. Likewise, this study includes a scheme to orchestrate the computation offloading in part or as a whole to nearby edge servers on demand.

2 Related Work

Zihan, et al. [5] developed a cooperative computing system consisting of MEC (Edge Cloud), MCC (Central Cloud), and local computing (mobile devices) as a test environment to study effective offloading decisions and resource allocation in the limited computing power environment of Mobile Edge Computing (MEC). Based on the proposed system, a Q-Learning based offloading policy was designed to pre-schedule the computational resources for each task and make optimal offloading decisions and resource allocation. The authors built a MEC environment considering mobility as a test environment for optimal task offloading. Whereas, in our study, we constructed and experimented with an environment that considers non-mobile edge devices such as surveillance cameras and sensor devices.

Whereas, Hossain [6] et al., proposed an RL-based scheme for Industrial IoT (IIoT) environment using edge computing. They proposed an RL-based scheme for IIoT, which is a research area with many topics such as computational complexity, latency minimization, task offloading, storage management, and power consumption. The authors found an optimal task offloading policy using Q-Learning, a popular reinforcement learning algorithm, in an IIoT environment where each device has computational power. In our study, we also used a reinforcement learning algorithm for task offloading, but the difference is that each device does not have computational power and the generated tasks must be offloaded.

3 System Organization and Experiment Environment

As shown in Fig. 1, we consider a local edge computing (LEC) system consisting of edge servers $E = \{0, 1, \ldots N\}$, offload broker server, and edge devices $e = \{1, 2, \ldots n\}$. In this system, an edge device, which is the data source, first sends a request to process a task to its edge server. In this case, each edge device is preset to its nearest edge server. (Here, the edge device selects the closest one from the location information of the surrounding edge servers).

The edge server (a.k.a. client edge server) that receives the task processing request determines whether or not the task can be handled by its own task processing capacity. If the task can be handled, it executes the task and returns the result to the edge device. If the task cannot be handled, it makes a query to the offload broker server. The offload broker server periodically obtains and maintain a record of the processing capacity, available network bandwidth, and location information of all edge servers within its domain. Along with these parameters, the offload broker server scans the record and apply an efficient edge server selection algorithm based on the client server requirements. With the selection result, the offload broker server will then inform the client edge server of

the most suitable node which can carry out the efficient processing of the tasks to be offloaded. The task offloading and computation then takes place.

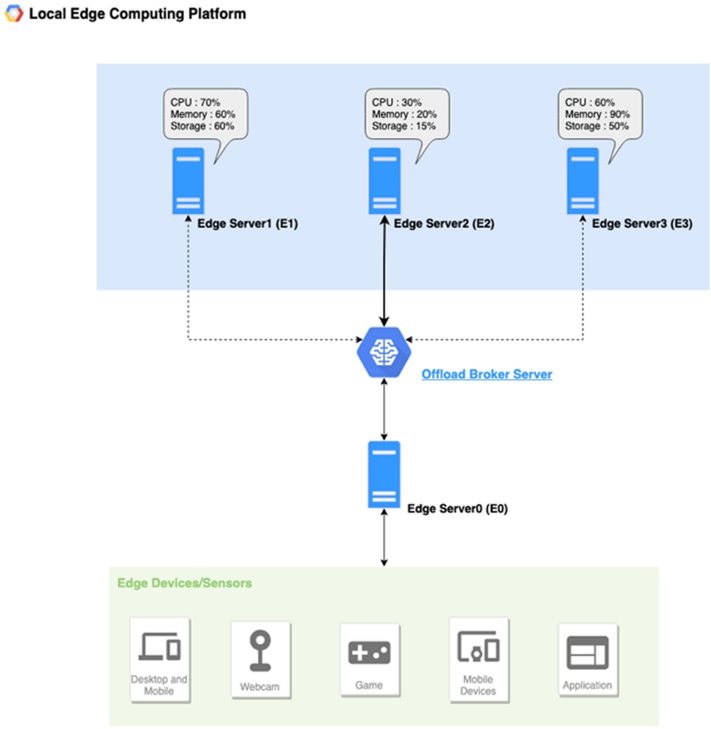

Fig. 1. Local edge computing system

4 Computation Offloading Broker

In this section, we proposed a dynamic computation offloading broker for efficient computation offloading and to minimize the processing delay.

4.1 The Computation Model and Deep Reinforcement Learning

In order to realize the proposed method, we developed a Computation Offloading Broker using DDPG (Deep Deterministic Policy Gradient), which is one of the deep reinforcement learning algorithms.

Deep reinforcement learning is a type of machine learning method that combines reinforcement learning, in which the system itself learns autonomously through trial and error, and neural networks. In reinforcement learning, the agent that actually performs the

action learns to maximize the reward in a given environment. In addition, reinforcement learning is modeled by a Markov Decision Process (MDP), which is defined as follows,

$$\mathrm{P}(s_t|s_{t-1}, s_{t-2}, \ldots, s_0) = P(s_t|s_{t-1}) \tag{1}$$

MDP means that the transition condition to a state s_t depends only on the previous state s_{t-1} and is independent of how the transition has been made before. Let us denote the environment as E, the reward as r, and the action as a. At each step t, the agent observes the current state s_t and selects the next action a according to the strategy π. The strategy is the probability of selecting action a based on the strategy in a certain state. Then, it receives a reward from the environment for the action it has taken, and transitions to the next state $st + 1$. The goal of reinforcement learning is to find a strategy that maximizes the reward the agent receives from the environment by repeating this sequence of events. The future rewards are multiplied by the discount rate γ $(0 \leq \gamma \leq 1)$ to express them in terms of present value. Furthermore, the sum of the rewards, or cumulative reward, is expressed as follows,

$$\mathrm{R} = \sum\nolimits_{t=0}^{\infty} \gamma^t r_{t+1} \tag{2}$$

In reinforcement learning, there are two major methods for finding a strategy that maximizes such a cumulative reward.

- **Value function based**

It is a method of indirectly obtaining optimal measures by estimating a value function. There are two types of value functions: the state value function and the action value function. The state value function is a function that expresses the value in a certain state when a certain measure is taken, and it can be expressed as follows.

$$V^{\pi}(s) = E\left[\sum\nolimits_{i=1} \gamma^{i-1} r_{t+i}|s\right] \tag{3}$$

The action value function is a function that expresses the value of a certain action in a certain state when a certain measure is being taken, and the same equation can be obtained as follows. The action value function is also called the Q-value.

$$Q^{\pi}(s, a) = E\left[\sum\nolimits_{i=1} \gamma^{i-1} r_{t+i}|s, a\right] \tag{4}$$

The above two value functions are interconnected, and the action a based on the strategy π that maximizes the action value function $Q^{\pi}(s, a)$ is the factor that ultimately maximizes the state value function $V^{\pi}(s)$.

$$\pi^*(s) = \arg max_{a \in A(s)} Q^*(s, a) \tag{5}$$

- **Policy function based**

It refers to the method of directly obtaining measures and improving the current measures without going through a value function. A typical approach is the measure

gradient method. The policy gradient method is an approach in which the measures are functions represented by a certain parameter θ, and the measures are learned directly by learning that parameter. Assuming that the measure model is $\pi_\theta(s, a)$, the learning of the parameter θ can be expressed as follows

$$\theta \leftarrow \theta + \delta \nabla_\theta J(\theta) \tag{6}$$

Here, δ means the learning rate. The effect of a change in the parameter θ on the objective function can be expressed by the measure gradient theorem as follows

$$\nabla_\theta J(\theta) = E_s\left[\sum_a Q^\pi(s, a)\nabla_\theta(a|s; \theta)\right] = E_{(s,a)}\left[Q^{\pi_\theta}(s, a)\nabla_\theta log\pi_\theta(a|s)\right] \tag{7}$$

4.2 Actor-Critic

Actor-Critic is a policy-based reinforcement learning algorithm [7]. It is an approach that directly improves the actor that decides the behavior, while simultaneously learning the criterion that evaluates the strategy. In general, the use of Actor-Critic is less susceptible to blurring of rewards and enables fast and stable learning. Figure 2 below shows the architecture of Actor-Critic.

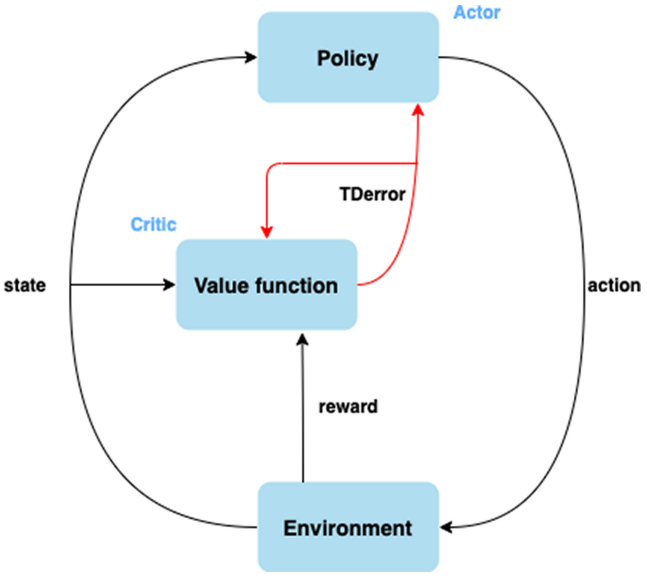

Fig. 2. The actor-critic architecture

4.3 Deep Deterministic Policy Gradient (DDPG)

DDPG is an algorithm that combines the decisive measure gradient method and DQN in the framework of Actor-Critic. In DQN, *Q(s,a)* of each state is calculated by measures,

and the action that maximizes the Q value is selected and acted upon. However, this method has the disadvantage that it can only handle discrete actions. On the other hand, DDPG does not seek the action that maximizes the Q-value, but rather parameterizes the measures and directly outputs the action, which corresponds to a continuous action space [8, 9].

Definitions of Parameters Using RL

To model this problem using RL, we define the Agent, State, Action, and Reward as follows.

- **Agent:** Edge devices that perform real-time critical tasks n.
- **State:** The agent state S_n, consists of the computational resources available at each edge server and the location information of each edge device.
- **Action:** $A_t \in [0, 1]$ where $A_t = 0$, edge device n is not offloaded and its task is processed, and when $A_t = 1$, the edge device offloads its task to the optimal edge server according to the current policy.
- **Reward:** Total number of tasks processed at each time step t.

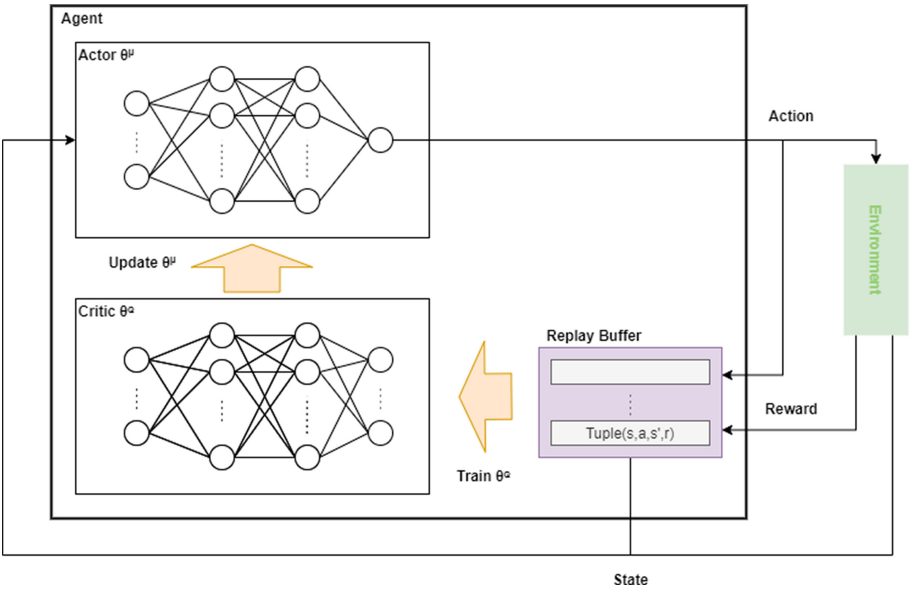

Fig. 3. DDPG architecture

In this study, the behavioral space of the offloading decision and computational offloading process in the edge computing environment is modeled using DDPG as a continuous behavioral space. The architecture of DDPG is shown in Fig. 3. First, we initialize the memory for the current state, transition state, reward function, and replay buffer. The replay buffer is a method that stores some of the experience in the environment

in a buffer rather than directly. This makes it possible to utilize not only the most recent experience but also the past experience during learning. Next, a soft update is performed to prevent overestimation of the state. After that, the Actor network makes the offloading decision according to the current strategy, and the Critic network evaluates it. If the specified episode has not been reached, the same process is repeated again.

The process is summarized in the flow chart below Fig. 4.

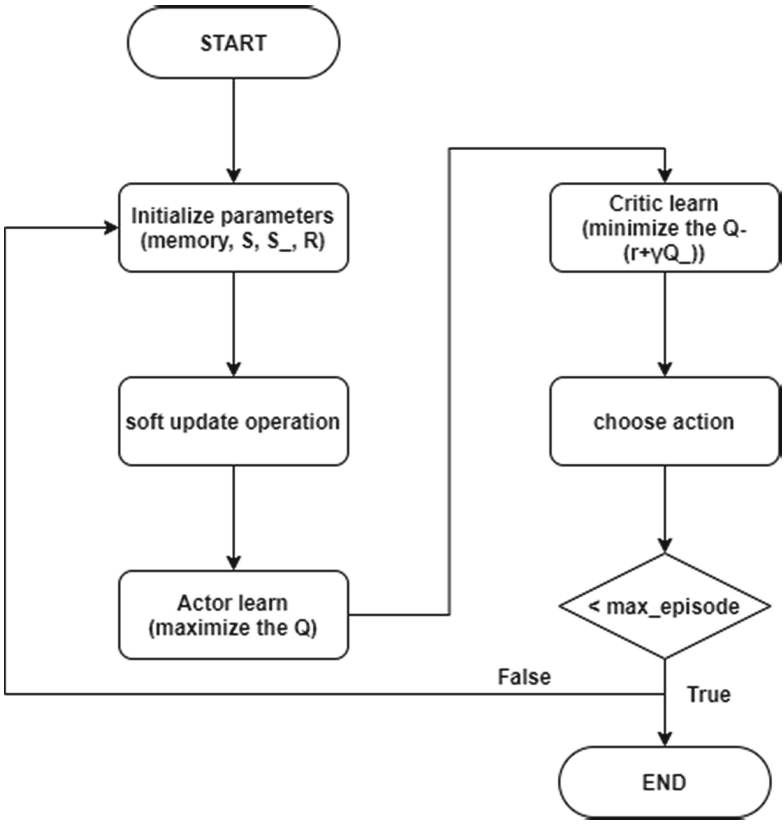

Fig. 4. Flow diagram of DDPG network training process

5 Simulation Results and Observations

In this section, we perform simulations to evaluate the proposed method using deep reinforcement learning. DDPG is implemented using Tensorflow [10].

5.1 Simulation Setting

In order to implement the Local Edge Computing (LEC) System in a virtual simulation environment, we set some parameters. The parameters in the simulation environment are summarized in Table 1, and the parameters for the DDPG algorithm are summarized in Table 2.

Table 1. Parameters in the simulation environment

Number of Edge devices	50
Number of Edge servers	10
Bandwidth between Edge servers	100 Mbps

Table 2. Parameters in the DDPG algorithm

Learning rate for Actor	1.0×10^{-4}
Learning rate for Critic	2.0×10^{-4}
Discount rate for rewards	0.99
Learning memory capacity	1.0×10^{4}
Actor's Layer	4
Critic's Layer	4
Hidden Layer	2
Activation function (Hidden Layer)	Relu (Rectified linear unit)
Activation function (Output Layer)	Sigmoid
Optimization function	Adam(Adaptive Moment Estimation)
Mini Batch Size	32
Soft Update rate	0.01

5.2 Simulation Results

The results of the simulation with the above parameters are shown in Fig. 5 below. Here, the x-axis is the number of episodes and the y-axis is the reward obtained in each episode.

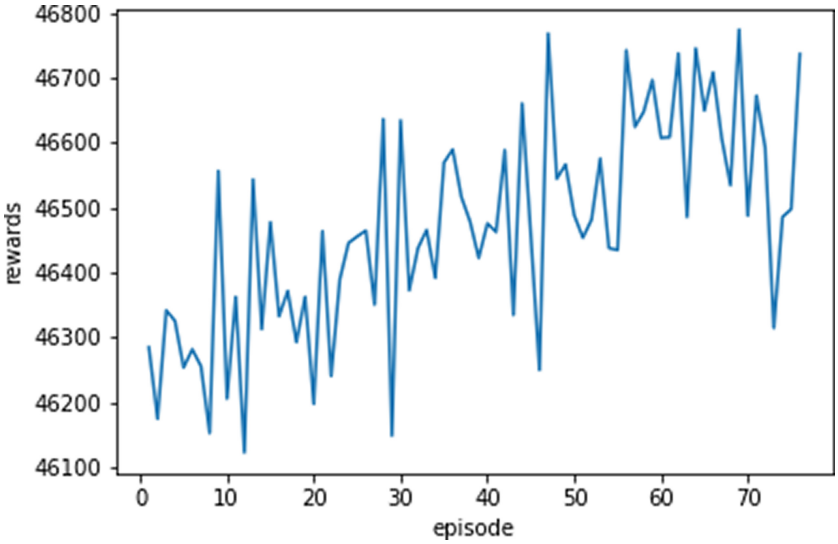

Fig. 5. Simulation results

5.3 Discussion of Results

From Fig. 5, we can see that the value of the reward earned by the agent increases as the episode progresses. Here, the reward value means the total value of tasks processed within one episode. This can be considered that the learning model is able to offload the requested tasks to the appropriate edge server properly. The variation in the value of the reward can be attributed to the fact that the epsilon-greedy method (which takes random actions with a certain probability and discovers unknown gradients) is employed in the learning process of the model.

6 Conclusion and Future Work

In this paper, we considered strategies to deal with the rapidly increasing task processing demands from edge devices and proposed a dynamic computational offloading method using deep reinforcement learning. We employed the DDPG algorithm to minimize the task processing speed per unit time, and confirmed that the agent learns a policy to offload the task to the most appropriate node.

In the future, we will compare our proposed method with existing methods in order to evaluate its efficiency and provide us insights on related vital issues. We also aim to improve the simulation system to be closer to the real-world edge computing environment.

References

1. IDC Corporate, IoT Growth Demands Rethink of Long-Term Storage Strategies, 27 July 2020. https://www.idc.com/getdoc.jsp?containerId=prAP46737220

2. Cao, K., Liu, Y., Meng, G., Sun, Q.: An overview on edge computing research. IEEE Access **8**, 85714–85728 (2020). https://doi.org/10.1109/ACCESS.2020.2991734
3. Lin, L., Liao, X., Jin, H., Li, P.: Computation offloading toward edge computing. Proc. IEEE **107**, 1–24 (2019). https://doi.org/10.1109/JPROC.2019.2922285
4. Jiang, C., Cheng, X., Gao, H., Zhou, X., Wan, J.: Towards computation offloading in edge computing: survey. IEEE Access **7**, 131543–131558 (2019). https://doi.org/10.1109/ACCESS.2019.2938660
5. Gao, Z., Hao, W., Han, Z., Yang, S.: Q-learning-based task offloading and resources optimization for a collaborative computing system. IEEE Access (2020). https://doi.org/10.1109/ICCC49849.2020.9238925
6. Sajjad Hossain, Md., Nwakanma, C.I., Lee, J.M., Kim, D.-S.: Edge computational task offloading scheme using reinforcement learning for IIoT scenario. ICT Express **6**, 291–299 (2020). https://doi.org/10.1016/j.icte.2020.06.002
7. Awate, Y.P.: Policy-gradient based actor-critic algorithms, pp. 505–509. IEEE Xplore, May 2009. https://doi.org/10.1109/GCIS.2009.372
8. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: ICML 2014, vol. 32, pp. 387–395, June 2014
9. Lillicrap, T.P., et al.: Continuous control with deep reinforcement learning. In: ICLR 2016, arXiv:1509.02971, Jul 2019 (latest version)
10. Tensorflow: The Python Deep Learning library https://www.tensorflow.org/