information

Article

Decentralized Offloading Strategies Based on Reinforcement Learning for Multi-Access Edge Computing

Chunyang Hu [1,2], Jingchen Li [3], Haobin Shi [3,*], Bin Ning [2] and Qiong Gu [2]

1 Hubei Key Laboratory of Power System Design and Test for Electrical Vehicle, Hubei University of Arts and Science, Xiangyang 441053, China; huchunyang@hbuas.edu.cn
2 School of Computer Engineering, Hubei University of Arts and Science, Xiangyang 441053, China; ningbin2000@hbuas.edu.cn (B.N.); qionggu@hbuas.edu.cn (Q.G.)
3 School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710129, China; staubs1212@mail.nwpu.edu.cn
* Correspondence: shihaobin@nwpu.edu.cn

Abstract: Using reinforcement learning technologies to learn offloading strategies for multi-access edge computing systems has been developed by researchers. However, large-scale systems are unsuitable for reinforcement learning, due to their huge state spaces and offloading behaviors. For this reason, this work introduces the centralized training and decentralized execution mechanism, designing a decentralized reinforcement learning model for multi-access edge computing systems. Considering a cloud server and several edge servers, we separate the training and execution in the reinforcement learning model. The execution happens in edge devices of the system, and edge servers need no communication. Conversely, the training process occurs at the cloud device, which causes a lower transmission latency. The developed method uses a deep deterministic policy gradient algorithm to optimize offloading strategies. The simulated experiment shows that our method can learn the offloading strategy for each edge device efficiently.

Keywords: multi-access edge computing; deep reinforcement learning; task offloading

Citation: Hu, C.; Li, J.; Shi, H.; Ning, B.; Gu, Q. Decentralized Offloading Strategies Based on Reinforcement Learning for Multi-Access Edge Computing. *Information* **2021**, *12*, 343. https://doi.org/10.3390/info12090343

Academic Editor: Diogo Mattos

Received: 23 June 2021
Accepted: 23 August 2021
Published: 25 August 2021

1. Introduction

With the emergence of the fifth-generation mobile communication system (5G), many computing-intensive services have been developed, such as face recognition [1], self-driving cars [2], and real-time translation [3]. However, the existing civil devices cannot provide sufficient computing resources for these computing-intensive tasks. A feasible method is cloud computing [4], which transfers tasks to a high-performance cloud server, and the computing results are returned to user devices. The following problem is that the too-far cloud server results in a higher transmission latency. Therefore, edge computing is required, deploying several services at the edges of the network [5].

Generally, edge services are closer to the user device than the cloud service, but the different distances between these services and the user device make the edge computing network need an efficient offloading strategy to achieve low latency [6]. Two types of latencies are generated in the offloading process: One is the transmission latency, which is resulted from sending tasks to services by uplink and returning the computing results to the user device by downlink [7]. The other one is the computing latency, related to the task and the corresponding service's computation ability. In a multi-access edge computing system (MEC), an excellent offloading strategy can assign edge services, according to the service state and tasks.

In the early design, MEC systems offloaded tasks through some fixed rules or algorithms, such as Round Robin and first-come-first-service. These methods were enough to offload a small number of tasks and avoid the block. However, once more user devices exist, or the MEC system has only finite services, the traditional methods cannot satisfy the user devices' needs. Hence, some heuristic algorithms are developed regarding task offloading as an optimization problem.

As deep reinforcement learning has achieved great success on sequential decision-making scenarios [8], more and more researchers have introduced reinforcement learning into the MEC task offloading strategy. Benefiting from the deep neural network, reinforcement learning is extended to high-dimensional space, and its outstanding generalization ability is what MEC task offloading needs [9]. Modeling task offloading to a Markov Decision Process (MDP), some researchers have attempted to design a reinforcement learning model for MEC task offloading in different ways. For a small-scale MEC system, model-based reinforcement learning methods can be introduced to control the task offloading through modeling the MEC environment [10]. In large-scale MEC systems or mobile edge computing scenarios, model-free reinforcement learning technologies are more feasible [11].

The existing reinforcement learning-based offloading strategies devote a centralized strategy for all edge servers, by which an additional communication cost is made. Although these methods can achieve low-latency resource allocation after training, the total cost barely reduces. For this reason, decentralized offloading strategies are desired by MEC systems, especially for large-scale MEC systems. In this work, we regard edge servers as a multi-agent system, building a multi-agent reinforcement learning model for MEC task offloading. Considering a high-powered cloud server and several individual edge servers, we introduce the centralized training and decentralized execution mechanism (CTDE) [12] to learn the offloading strategies. Specifically, every edge server executes its individual offloading strategy, and the corresponding experiences are sent to the cloud server for centralized training. With that, edge servers need no communication channel after training, so no additional cost occurs.

The main contributions of this work are summarized as follows:

- We discuss some shortages in the existing reinforcement learning-based offloading strategies in MEC. For these shortages, we develop a new framework to improve task offloading by introducing the CTDE mechanism.
- We first introduce the centralized training and decentralized execution mechanism into MEC systems, modeling a more feasible reinforcement learning model for MEC task offloading.
- We conduct several experiments on simulation platforms to compare our framework with several existing methods. The results show that our framework outperforms the baseline methods.

The rest of the paper is organized as follows: We first give the background in Section 2. In Section 3, we present the proposed reinforcement learning-based framework. In Section 4, we present and discuss the results of our experiments. Finally, we conclude in Section 5 and give directions for future research.

2. Motivation

The persistent development of the Internet of Things is driven by the support of 5G communication, cloud computing, multimedia, mobile computing, and the use of big data technologies to generate the smart analytics value. To reduce network stress, edge computing shifts resources at the edge of the network, resulting in a lower transmission latency. Multi-access edge computing (MEC) is developed to achieve high quality of service (QoS) with low energy consumption for computation-intensive applications [13]. The multi-access system aims to address transmission latency and computing delay for the video process services, Internet of Things, augmented reality, virtual reality, optimized local data caching, and many other use cases for Smart Cities [14].

Different from traditional cloud computing, in a MEC architecture, several servers can be used for offloading tasks, so the offloading strategy cannot be modeled as a single object optimization issue. One direct method is to regard all servers as an entire server. However, the optimization complexity will not be decreased through such a paradigm. Some researchers modeled task offloading in MEC as an NP-hard Knapsack Problem, calculating the optimum resolution by dynamic planning [15], the genetic algorithm [16], or other heuristic algorithms [17]. The calculations for these methods require a vast iteration or matrix manipulation, and it is hard to satisfy the real-time requirement. For this problem, researchers desired a feasible method to learn a fixed strategy for task offloading. There are three requirements for the learning: the learned strategy is efficient enough for the complicated MEC architecture; the learned strategy can reflect offloading as soon as possible; the learned strategy generates less additional cost in the execution. For these purposes, reinforcement learning has attracted intense scholarly interest. Reinforcement learning can execute real-time decision-making after training, while the learned policy needs no additional calculation, except the policy network.

The following problem is that the state space and action space in a reinforcement learning model will increase with the number of edge servers. In a general or small-scale MEC architecture, reinforcement learning-based methods perform well. However, once these methods are used to learn offloading strategies for a large-scale MEC architecture, the training cost will be unaffordable.

When using multi-agent reinforcement learning methods to learn offloading strategies, the coupling among edge servers should be taken into account. Independent learning ignores the effect of other agents, resulting in misconvergence or suboptimal solutions. Centralized strategies lead to additional communication costs during the execution process. For these reasons, the existing MEC systems need a decentralized offloading strategy with coupling among edge servers.

3. Background

3.1. Task Offloading in MEC

The development of task offloading methods in MEC can be divided into three phases. In earlier research, the resource scheduling in MEC is modeled as a nondeterministic polynomial problem, and heuristic algorithms and subtask optimization play important roles in such research. For example, Xiao et al. [18] divided the task into multiple subtasks, optimizing the placement of the edge computing server; Samanta et al. [19] developed a distributed and latency-optimal microservice scheduling mechanism, enhancing the quality-of-service (QoS) for real-time (mobile) applications; and Xu et al. [20] aimed to the offloading framework for deep learning edge services, leveraging heuristic searching to acquire the appropriate offloading strategy.

Considering that the task offloading in MEC is continuous and the offloading process has a Markov property, researchers focused on reinforcement learning based task offloading, leveraging the stability and generalization of deep reinforcement learning technologies. Chen et al. [11] utilized a deep Q-network to learn the optimal offloading strategy, minimizing the long-term cost; Wang et al. [21] leveraged the Resource Allocation scheme to improve the reinforcement learning model, learning the offloading strategy in the mutative MEC conditions; Liu et al. [22] considered the channel conditions between the end devices and the gateway, using the reinforcement learning approach to solve the resource allocation problem.

With the MEC systems become more and more complicated, the too-large state space and action space drove researchers to look for a new paradigm. Some researchers have paid attention to multi-agent reinforcement learning, looking for a convenient MDP model. Liu et al. [23] regarded task offloading as a stochastic game, using independent learning agents to learn offloading strategy for each edge server; Wang et al. [24] leveraged multi-agent deep Deterministic Policy Gradient to train UAVs in a MEC architecture; Munir et al. [25] considered a microgrid-enabled MEC network, using asynchronous advantage actor-critic algorithm to train agents.

3.2. Centralized Training and Decentralized Execution

In the training process for a multi-agent reinforcement learning model, agents continuously interact with the environment to generate experiences used to optimize their policy networks. However, mainstream training algorithms use the actor–critic network [26]. That is, the execution is separated from the training. From this point, some researchers developed the CTDE mechanism [12]. Specifically, when agents execute their policy, every agent makes decisions according to only its own observation or perception, while the optimizing process needs to take into account all agents' perceptions and behavior.

Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [12] is the typical multi-agent reinforcement learning model based on CTDE mechanism. A multi-agent reinforcement learning experience with N agents can be termed as $< S, A, S', R >$, where $S = [s_1, s_2, \dots, s_N]$ is the states for all agents, $A = [a_1, a_2, \dots, a_N]$ is the joint action under joint state S, $S' = [s'_1, s'_2, \dots, s'_N]$ denotes the next states for these agents, and $R = [r_1, r_2, \dots, r_N]$ is the rewards returned from the environment. Based on the actor–critic model, MADDPG uses an actor network to fit the policy for each agent, while a critic network is built to evaluate its policy. Let $\pi_i(\dots|\theta^i_a)$ denote the policy network for the i-th agent, and the corresponding critic-network is represented by $Q_i(|\theta^i_c)$. The goal is for agents to maximize the excepted reward:

$$\mathcal{R} = \sum_{t_0}^{T} \sum_{i=1}^{N} \gamma^t R^t_i. \tag{1}$$

In the sampling process, the i-th agent perceives the environment, gets their state s_i, and then selects a behavior according to its policy: $a_i = \pi_i(s_i|\theta^i_a)$. After all agents have selected their actions, they execute their joint action $a = [a_1, a_2, \dots, a_N]$, and rewards $r = [r_1, r_2, \dots, r_3]$ are returned from the environment, while the next joint state for agents is $s' = [s'_1, s'_2, \dots, s'_N]$. Then, an experience $< s, s', a, r >$ is sent to the replay buffer.

Once the replay buffer receives enough experience, the policies can be optimized. For the i-th agent, its critic network estimates the Q-value as follows:

$$Q_i(a, s|\theta^i_c) = Q_i(a_i; s, (a_1, a_2, \dots, a_{i-1}, a_{i+1}, \dots, a_N)|\theta^i_c). \tag{2}$$

The optimization objective for the critic network is fitting the discounted reward $\sum_{t=t_0}^{T} \gamma^{t-t_0} r_{i,t}$, where γ is the discounted factor. The $\gamma r_{i,t+1}, \gamma^2 r_{i,t+2}, \dots, \gamma^{T-t_0} r_{i,T}$ can be replaced with $\gamma Q_i(a', s'|\theta^i_c)|_{a' \sim \pi_i(s'_i|\theta^i_a)}$. Hence, the loss function for the i-th critic-network is as follows:

$$L\theta^i_c = \frac{1}{2}[Q_i(a, s|\theta^i_c) - (r_i + \gamma Q_i(a', s|\theta^i_c))]^2. \tag{3}$$

As for the corresponding actor network, MADDPG uses the Deterministic Policy Gradient to substitute for the Policy Gradient, that is, a deterministic policy is output by the actor network rather than a distribution. With that, the gradient of the objective $J(\theta^i_a) = E_{s_i} R[s_i, a_i]$ can be overwritten as follows:

$$\delta_{\theta^i_a} J(\theta^i_a) = E_s[\delta_{\theta^i_a} \log \pi_i(s_i|\theta^i_a) \delta_a Q_i(a, s|\theta^i_c)]. \tag{4}$$

MADDPG allows every agent to make decisions according to only its own state, but the evaluation for policy needs to consider the joint state and action. With the centralized training and decentralized execution mechanism, MADDPG can achieve decentralized multi-agent reinforcement learning with a centralized communication channel.

4. The Proposed Method

In this work, we consider a multi-access edge computing architecture with a cloud server and several edge servers, developing a Decentralized Multi-Agent Reinforcement Learning-based offloading strategy (DMARL) for MEC. In this section, we first describe the proposed DMARL framework. Then, the specific learning process with the CTDE mechanism is given.

4.1. Learning Model for MEC

As shown in Figure 1, there are three layers in our DMARL model: a user layer with user devices, an edge layer with several edge servers, and a cloud layer with a cloud server. Using N to represent the number of edge servers, we make $s_i^E = < w_i, f_i >$ denote the state for the i-th server, in which w_i is the work state for the server and f_i represents its computation ability. In the user layer, the tasks published by the user devices constitute a task set $[t_1, t_2, ..., t_M]$, where M is the maximal number of tasks. The j-th task in the task set can be termed as a tuple $< d_j, l_j, u_j, k_j >$, where d_j is the importance for the j-th task, l_j is the transmission delay caused by downlink and uplink, u_j denotes the workload for executing the j-th task, and k_j is the waiting time. We develop a decentralized reinforcement learning model for task offloading, using actor–critic networks for learning the offloading strategies for edge servers. We assign each edge server with an actor network $\pi_i(|\theta_a^i)$ in the edge layer, and its critic network is deployed in the cloud server for centralized optimization.

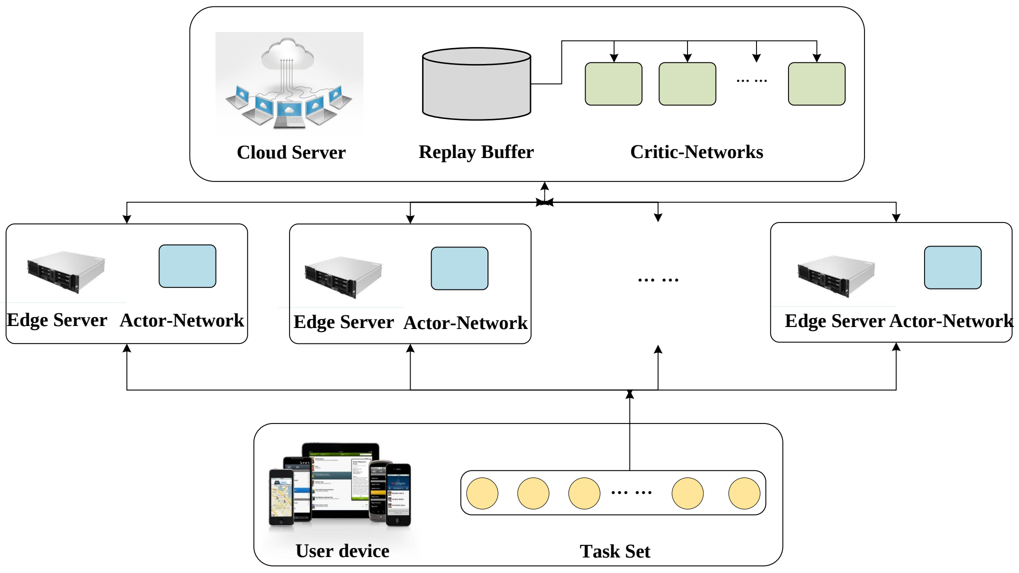

Figure 1. The proposed architecture, in which the centralized critic networks are deployed at the cloud layer, and the actor networks are deployed at the edge layer.

When a new task is published or an edge server completes its current task, the task set is encoded as s^T and transmitted to the edge layer. Every edge server receives the encoded task set, combining s^T and its server state s_i^E as its individual state $s_i = [s^T, s_i^E]$. Then an action is selected by the actor network: $a_i = \pi_i(s_i|\theta_a^i)$, which is a deterministic policy. Each server has $M + 1$ selectable actions: offloading one of the M tasks and no offloading. The edge server executes its action while sending s_i and a_i to the cloud server. Then a reward is generated by calculating the mean waiting time for the task as follows:

$$r_i = \sum_{j=1}^{m} k_j, m \leq M. \quad (5)$$

These processes for different edge servers are decentralized. When all edge servers conduct this operation and the next step is done, the cloud server combines the received data as an experience $< S, S', A, A', R >$, where $S = (s_1, s_2, \ldots, s_N)$, S' and A' is the joint state and joint action at next step, $A = (a_1, a_2, \ldots, a_N)$, and $R = (r_1, r_2, \ldots, r_N)$. The experience is sent to the replay buffer, which is also in the cloud layer. The replay buffer samples experiences train the actor networks and critic networks in the case of enough experiences. In the optimization process, the critic networks are updated in a centralized mechanism, while the experiences are transmitted to the edge layer for the update of the actor networks in a decentralized way. The specific update is described in the next subsection.

4.2. Update and Training

Similar to MADDPG, the update for edge servers' offloading strategies use the CTED mechanism. After the replay buffer receives enough experiences, the cloud server optimizes the critic networks, while the corresponding experiences and evaluations are sent to edge servers to optimize the actor networks. The update for critic networks is centralized, and that for actor networks is decentralized.

A experience batch sampled from the replay buffer is termed as D. The critic networks are used to estimate the Q-values, so that the purpose for its optimization is to approximate the discounted reward $r + \gamma Q(s', a')$. Let $Q_i(|\theta_c^i)$ denote the critic network for the i-th edge server, and $\pi_i(|\theta_a^i)$ is the corresponding actor network. For the critic network of the i-th edge server, its loss function is calculated as follows:

$$L(\theta_c^i) = \mathbb{E}_{S,S',A,A',r_i\ D}[Q_i(S, A|\theta_c^i) - (r_i + \gamma Q_i(S', A'|\theta_c^i))]. \tag{6}$$

Then, the calculated $Q_i(S, A|\theta_c^i)$ is packaged with (S, A), and they are sent to the edge layer to update the actor networks. A packaged experience for the actor network is $< S, A, Q >\in G$, where $Q = (Q_1(S, A|\theta_c^1, Q_2(S, A|\theta_c^2)), \dots, Q_N(S, A|\theta_c^N))$. In the update for actor networks, $Q_i(S, A|\theta_c^i$ represents the reward for executing the $a_i \in A$ at state $s_i \in S$. The policy gradient for the i-th actor-network is as follows:

$$\delta_{\theta_i^a} J(\pi_i) = \mathbb{E}_{S,A,Q\ G}[\delta_{\theta_i^a} P(a_i|\pi(s_i|\theta_a^i))\delta_{a_i} Q_i(S, A)], \tag{7}$$

where $P(a_i|\pi(s_i|\theta_a^i))$ is the probability of selection a_i at state s_i.

With the proposed DMARL, edge servers have decentralized offloading strategies, which is more suitable for distributed systems. Moreover, we deploy the critic networks at the cloud layer while the actor networks are stored in corresponding edge servers, resulting in a lower training cost and additional transmission.

Then, we analyze the training cost of the proposed DMARL and other multi-agent reinforcement learning-based methods. We make D_s denote the dimensionality of s_i and D_a represent the dimensionality of a_i. In centralized learning-based methods, the policy network can be regarded as a mapping from $(N * D_s)$ to (D_a), and the evaluation network is also fed by a $(N * D_s)$-dimensional data. In CTDE-based methods, the policy network is decentralized so that the corresponding input is D_s-dimensional, but the evaluation network needs a $(N * D_s)$-dimensional input. The proposed DMARL models MEC systems as decentralized multi-agent reinforcement learning models so that the inputs in both the policy network and evaluation network are D_s-dimensional, which results in a lower space cost. As for the computational cost and training time, decentralized networks require fewer floating point operations than centralized networks. However, the total floating point operations depend on the learning efficiency, which is difficult to quantify theoretically. We present the performance of DMARL and baseline methods under the same number of training episodes to investigate the learning efficiency in the next section.

5. Experiments

In this section, several simulated experiments are conducted to validate our method. We first introduce the baseline methods used in these experiments; then, the experimental setups are given. Finally, we give and analyze the experiment results.

5.1. Baseline Methods

In the following paragraph, three task offloading methods are introduced, and they are chosen as the baselines to evaluate the proposed DMARL.

HEFT (Heterogeneous Earliest Finish Time) [27] is a heuristic algorithm that calculates the earliest finish time of executing each task in each server. **Round-Robin** [28] assigns tasks to servers in turn, in which the load and current state of each task is ignored. Round-Robin is also a heuristic method. **PPO** [29] is a reinforcement learning-based method that uses the Proximal Policy Optimization (PPO) algorithm to train the joint offloading strategy for edge servers. Different from our method, PPO regards just the task set as the state for servers.

5.2. Experiment Setup

In the simulated experiment, we set eight edge servers at the edge layer ($N = 8$), with a cloud server in the cloud layer. A user device publishes tasks continuously, and the maximal number of tasks is 10 ($M = 10$). The computing capabilities of the eight servers are 5, 10, 10, 15, 15, 20, 20, and 25 GHz, and the transmission latency is sampled from a uniform distribution [1, 20] ms. The required number of CPU cycles by a task ranges from 1 to 10 megacycles.

As for the reinforcement learning model, we set the learning rate to 0.001, and the discounted factor γ is 0.9. The encoder for the task set is a multi-layer perceptron with a hidden layer (64 nodes). The actor network and critic network consist of fully-connected layers with 128 neurons. The capacity of the replay buffer is set to 10,000, while the batch size is 128.

5.3. Result and Analysis

The experiment results are given in Table 1, which records the average of total waiting times $r_1, r_2, \ldots .r_N$ after training. From Table 1, we can see that the reinforcement learning-based methods' performance is related to the training episodes. At the beginning of the training (0–10,000 episodes), both the proposed DMARL- and PPO-based methods fall behind, and HEFT obtains the best result. As the episodes increase, our method and PPO-based method can rival the heuristic algorithm. At the end of the training (100,000 episodes), the proposed DMARL achieves the best result: the total waiting time is just 1586 ms. Compared with heuristic methods, our method needs less time to conduct the learned policy, and the learned offloading strategies can adapt in most cases, due to the generalization of neural networks. Moreover, the proposed DMARL utilizes the CTDE mechanism to train offloading strategies, capturing the coupling among edge servers by centralized critic networks so that it outperforms the PPO-based method. As an independent learning method, the PPO-based method has the advantage of quick learning speed, benefiting from decoupling the relationship among different agents. However, this decoupling makes it difficult for the edge server to obtain an accurate estimation of the Q-value, so that some correct behavior may be punished due to other edge servers' incorrect policies. The policies learned by the PPO-based method always converge to a local optimum or misconverge. On the contrary, our DMARL uses the CTDE mechanism to ensure the coupling of different agents, resulting in an accurate estimation for the Q-value. That is why our DMARL methods outperforms the PPO-based method at the end of the training.

Table 1. The results for the experiments.

	HEFT	**Round-Robin**	**PPO-Based**			**DMARL**		
Episodes			**10 k**	**50 k**	**100 k**	**10 k**	**50 k**	**100 k**
Waiting Time	2394	5170	6132	4028	3231	6547	3005	**1586**

To show the superiority of our DMARL more convincingly, we calculate the average collision (two or more edge servers offloading the same task) rates for the proposal and PPO-based method. To compare algorithms, we therefore report the performance averaged across 10 runs. We do this to ensure that we are not simply reporting a result due to stochasticity. The results are given in Figure 2. For all edge servers in the MEC system, using our DMARL can achieve a lower collision rate than using the PPO-based method. The reason is that the PPO-based method is an extension of the independent learning mechanism, neglecting the relationship among servers. Our DMARL connects these servers' behavior in the process of evaluation, which ensures the relationship for them to some extent. The relationship among servers ensures their coordination. Once a server ignores others' policies, it cannot accurately judge others' behaviors when making a decision. Likewise, other servers also cannot avoid collision by predicting the server's behavior. However, the CTDE mechanism estimates the Q-value in a centralized way, and the coordination of servers occurs in the unbiased training, so the collision hardly occurs.

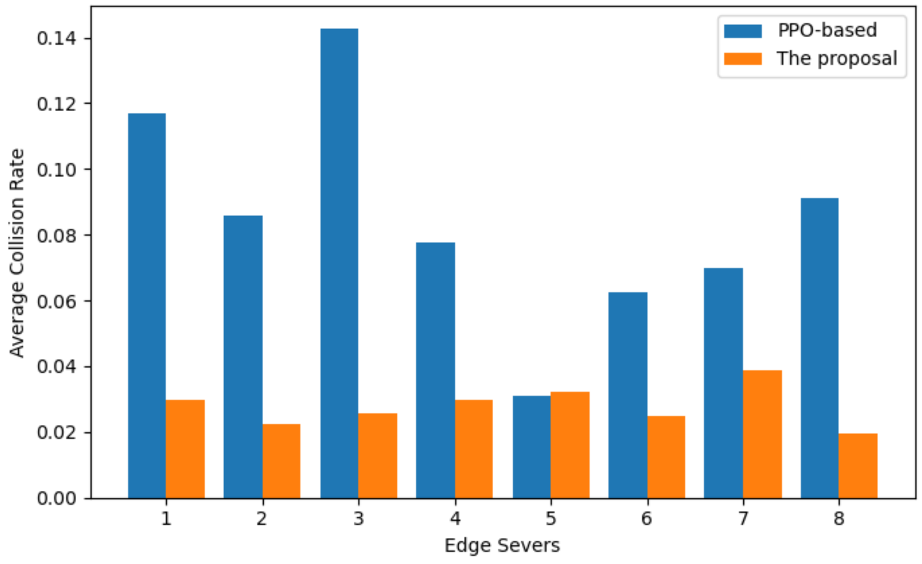

Figure 2. The average collision rates for edge servers.

6. Conclusions

Aiming at a decentralized offloading strategy for MEC systems, this work leverages the CTDE mechanism for training the offloading strategy for each edge server. Modeling task offloading as a reinforcement model, we separate the sampling process from the update process. The proposed DMARL method is based on the actor critic algorithm. However, we deploy the critic networks at the cloud layer, while the actor networks are in the edge layer. With the centralized learning and decentralized execution mechanism, edge servers can hold their coupling through centralized critic networks. The simulated experiment shows that the proposed DMARL outperforms heuristic methods and other reinforcement learning based methods.

Author Contributions: Conceptualization, C.H. and J.L.; methodology, C.H.; software, H.S.; validation, B.N. and Q.G.; formal analysis, J.L.; investigation, C.H.; resources, B.N.; data curation, H.S.; writing—original draft preparation, J.L.; writing—review and editing, J.L.; visualization, C.H.; supervision, C.H.; project administration, C.H.; funding acquisition, C.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a grant from the Major Research Development Program of Hubei Province (No. 2020BBB092) and Hubei Superior and Distinctive Group of "Mechatronic and Auto mobiles" (XKQ2021020).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Siregar, S.; Syahputra, M.; Rahmat, R. Human face recognition using eigenface in cloud computing environment. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2018; Volume 308, p. 012013.
2. Badue, C.; Guidolini, R.; Carneiro, R.V.; Azevedo, P.; Cardoso, V.B.; Forechi, A.; Jesus, L.; Berriel, R.; Paixao, T.M.; Mutz, F.; et al. Self-driving cars: A survey. *Expert Syst. Appl.* **2020**, *165*, 113816. [CrossRef]
3. Nasereddin, H.H.; Omari, A.A.R. Classification techniques for automatic speech recognition (ASR) algorithms used with real time speech translation. In Proceedings of the 2017 Computing Conference, London, UK, 18–20 July 2017; pp. 200–207.

4. Varghese, B.; Buyya, R. Next generation cloud computing: New trends and research directions. *Future Gener. Comput. Syst.* **2018**, *79*, 849–861. [CrossRef]
5. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A survey on the edge computing for the Internet of Things. *IEEE Access* **2017**, *6*, 6900–6919. [CrossRef]
6. Sheng, J.; Hu, J.; Teng, X.; Wang, B.; Pan, X. Computation offloading strategy in mobile edge computing. *Information* **2019**, *10*, 191. [CrossRef]
7. Wen, W.; Fu, Y.; Quek, T.Q.; Zheng, F.C.; Jin, S. Joint uplink/downlink sub-channel, bit and time allocation for multi-access edge computing. *IEEE Commun. Lett.* **2019**, *23*, 1811–1815. [CrossRef]
8. Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; Meger, D. Deep reinforcement learning that matters. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
9. Han, X.; Gao, G.; Ning, L.; Wang, Y.; Zhang, Y. Approximation Algorithm for the Offloading Problem in Edge Computing. In *International Conference on Wireless Algorithms, Systems, and Applications, Proceedings of the 15th International Conference, WASA 2020, Qingdao, China, 13–15 September 2020*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 134–144.
10. Munir, M.S.; Abedin, S.F.; Tran, N.H.; Hong, C.S. When edge computing meets microgrid: A deep reinforcement learning approach. *IEEE Internet Things J.* **2019**, *6*, 7360–7374. [CrossRef]
11. Chen, X.; Zhang, H.; Wu, C.; Mao, S.; Ji, Y.; Bennis, M. Performance optimization in mobile-edge computing via deep reinforcement learning. In Proceedings of the 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall), Chicago, IL, USA, 27–30 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
12. Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 6382–6393.
13. Alameddine, H.A.; Sharafeddine, S.; Sebbah, S.; Ayoubi, S.; Assi, C. Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 668–682. [CrossRef]
14. Porambage, P.; Okwuibe, J.; Liyanage, M.; Ylianttila, M.; Taleb, T. Survey on multi-access edge computing for internet of things realization. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 2961–2991. [CrossRef]
15. Gao, H.; Huang, W.; Zou, Q.; Yang, X. A dynamic planning framework for qos-based mobile service composition under cloud-edge hybrid environments. In Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing, London, UK, 19–22 August 2019; Springer: Berlin/Heidelberg, Germany, 2019; pp. 58–70.
16. Li, Z.; Zhu, Q. Genetic algorithm-based optimization of offloading and resource allocation in mobile-edge computing. *Information* **2020**, *11*, 83. [CrossRef]
17. Tran, T.X.; Pompili, D. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Trans. Veh. Technol.* **2018**, *68*, 856–868. [CrossRef]
18. Xiao, K.; Gao, Z.; Wang, Q.; Yang, Y. A heuristic algorithm based on resource requirements forecasting for server placement in edge computing. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC), Bellevue, WA, USA, 25–27 October 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 354–355.
19. Samanta, A.; Li, Y.; Esposito, F. Battle of microservices: Towards latency-optimal heuristic scheduling for edge computing. In Proceedings of the 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 24–28 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 223–227.
20. Xu, X.; Li, D.; Dai, Z.; Li, S.; Chen, X. A heuristic offloading method for deep learning edge services in 5G networks. *IEEE Access* **2019**, *7*, 67734–67744. [CrossRef]
21. Wang, J.; Zhao, L.; Liu, J.; Kato, N. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Trans. Emerg. Top. Comput.* **2019**. [CrossRef]
22. Liu, X.; Qin, Z.; Gao, Y. Resource allocation for edge computing in IoT networks via reinforcement learning. In Proceedings of the ICC 2019—2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–6.
23. Liu, X.; Yu, J.; Feng, Z.; Gao, Y. Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing. *China Commun.* **2020**, *17*, 220–236. [CrossRef]
24. Wang, L.; Wang, K.; Pan, C.; Xu, W.; Aslam, N.; Hanzo, L. Multi-Agent Deep Reinforcement Learning Based Trajectory Planning for Multi-UAV Assisted Mobile Edge Computing. *IEEE Trans. Cogn. Commun. Netw.* **2020**, *7*, 73–84 [CrossRef]
25. Munir, M.S.; Abedin, S.F.; Tran, N.H.; Han, Z.; Hong, C.S. A Multi-Agent System toward the Green Edge Computing with Microgrid. In Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019; pp. 1–7.
26. Konda, V.R.; Tsitsiklis, J.N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, US , 2000; pp. 1008–1014.
27. Bittencourt, L.F.; Sakellariou, R.; Madeira, E.R. Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm. In Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Pisa, Italy, 17–19 February 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 27–34.
28. Rasmussen, R.V.; Trick, M.A. Round robin scheduling–a survey. *Eur. J. Oper. Res.* **2008**, *188*, 617–636. [CrossRef]
29. Li, H.; Che, X. DRL-Based Edge Computing Model to Offload the FIFA World Cup Traffic. *Mob. Inf. Syst.* **2020**, *2020*, 8825643.