2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom) | 978-1-6654-4377-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/CSCloud-EdgeCom52276.2021.00044

Deep Reinforcement Learning-based Computation Offloading in Vehicular Networks

Liwei Geng, Hongbo Zhao*, Haoqiang Liu, Yujie Wang, Wenquan Feng, Lu Bai
School of Electronic and Information Engineering
Beihang University
Beijing, China
buaaglw@163.com

Abstract—**With the rapid development of 5G communications and the Internet of Things (IoT), vehicular networks have enriched people's lives with abundant applications. Since most of such applications are computation-intensive and delay-sensitive, it is difficult to guarantee the requirements of low latency and low energy consumption by relying on vehicles only. In addition, low latency has posed great challenge to the cloud computing. Therefore, as a promising paradigm, Mobile Edge Computing (MEC) is developed for vehicular networks to relieve the pressure on vehicles, which means to offload tasks to edge servers. However, existing studies mainly consider a constant channel scenario and ignore load balancing of edge servers in the system. In this paper, deep reinforcement learning is adopted to build an intelligent offloading system, which can balance the load balancing in the time-varying channel scenario. First, we introduce a communication model and a calculation model. Then the offloading strategy is formulated as a joint optimization problem. Furthermore, a deep deterministic policy gradient (DDPG) algorithm based on priority experience replay in the distributed scheme, which considers the load balancing, is proposed. Finally, performance evaluations illustrate the effectiveness and superiority of the proposed algorithm.**

Keywords—Vehicular networks, mobile edge computing, reinforcement learning, deep deterministic policy gradient

I. Introduction

With the rapid development of 5th Generation (5G) mobile networks and artificial intelligence, people's requirements for smart vehicles get higher accordingly [1]. As a result, vehicular networks have generated more and more computation-intensive and delay-sensitive tasks, but vehicle terminals do not have sufficient resources to complete them. Although the cloud server has sufficient computing resources [2], it is too far away from the vehicle terminal. However, the completion of these tasks will not only affect the users' experience, but also related to safe driving. Consequently, it is urgent to solve challenges [3]. In such a case, emerging Mobile Edge Computing (MEC) is considered an effective solution [4]. The vehicle can offload the task to the edge server at the edge, which has abundant computing resources. Commonly, we call this method vehicular edge computing (VEC). Via offloading the task to edge server, not only the time delay of the task can be decreased, but also the energy consumption of the vehicle can be greatly reduced.

Computing offloading is a crucial feature of MEC/VEC technologies [5]. A lot of results were achieved in computing offloading. Most early works considered single-user scenarios. Specifically, [6] proposed a dynamic computing offloading algorithm based on Lyapunov optimization in a single user scheme. Offloading in multi-user scenarios also was considered in several works. Reference [7] proposed a computational offloading decision to trade-off energy consumption and delay through game theory in a multi-user environment. Reference [8] studied computing offloading to minimum delay and transmission energy consumption by adopting extreme value theory in multi-user scheme. In addition to binary offloading, partial offloading has also been extensively studied. Reference [9] discussed the problem of partial offloading in a time division multiple access system, and defined an optimal resource allocation strategy based on thresholds. Reference [10] discussed the trade-off between delay and energy consumption in the partial offloading decision problem. Reference [11] studied the computing offloading of vehicles in the 5G scheme, and modeled the energy consumption of vehicles from two aspect. However, the exiting works mostly consider the delay and energy from the user's perspective, which is monotonous.

Deep reinforcement learning (DRL) is a promising solution to solve decision-making problems [12]. Many works about applying reinforcement learning to computational offloading have been carried out. [13] considered the high dynamics of vehicular networks, constructed a synchronized random walk model and used the deep Q-network (DQN) to reduce the system delay. Reference [14] constructed an intelligent offloading system for vehicular edge computing by leveraging the double deep Q-network algorithm. However, when the action apace is continuous, Q-learning and DQN algorithm are not applicable. In order to apply deep reinforcement learning methods, many existing works discretized the continuous action space, which could affect the optimization result.

Most of the above studies are performed under the assumption that the channel is constant. However, in actual situations, user terminals are usually moving, thus the channel is commonly time-varying. Moreover, most of the existing works regard delay and energy consumption as optimization indicators, but ignore the load balancing of edge servers, which indicates the rational use of resources. In addition, the value-base method can not handle continuous action space well.

We consider a vehicular network that multiple vehicles generate separable tasks in the time-varying channel scenario. In addition, a joint strategy optimization problem is formulated to achieve the trade-off among delay, energy consumption, and load balancing. Since it is difficult to solve the problem in such a dynamic scenario, a DDPG algorithm based on priority experience replay, which considers load balancing in the distributed scheme (DLPR-DDPG), is proposed. The main contributions of this article are summarized as follows.

1). We model the offloading scheduling process by a carefully designed Markov decision process (MDP), in which time-varying channels, divisible tasks, delay, energy consumption, and server load balancing are all considered. In addition, DRL is utilized to find the optimal policy.

2). Deep deterministic policy gradient (DDPG) is proposed to optimize the offloading policy. In addition, to train the network more efficiently, we adopt a distributed structure and use the priority experience replay.

3). We implement extensive simulations to compare with baseline schemes in the proposed MEC system to verify the effectiveness of the DLPR-DDPG. The numerical simulation results show that the convergence and robustness of DLPR-DDPG are better than the baseline methods.

The rest of the article is organized as follows. Section 2 presents the system model. Section 3 presents problem formulation. Section 4 introduces the offloading strategy. Section 5 presents the evaluation results. Section 6 concludes our work.

II. System Mode

As shown in Fig. 1, a city-wide vehicular network can be divided into several zones. Each zone consists of a central base station (BS), several edge servers, and many vehicles. The BS has abundant computation resources and connects to multiple heterogeneous edge servers placed by network operators through optical fibers. The edge servers provide computing

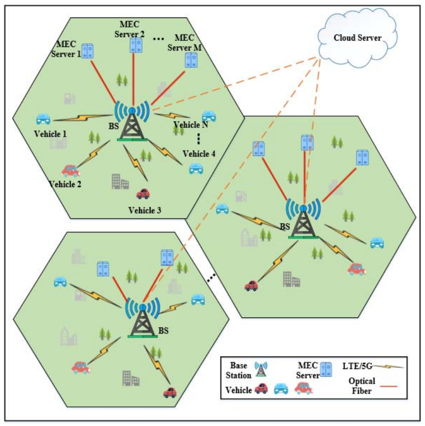
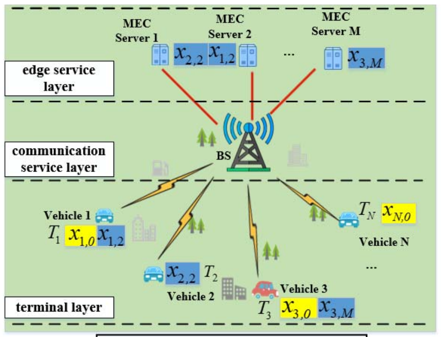


Fig. 1. The architecture of MEC-based vehicular networks



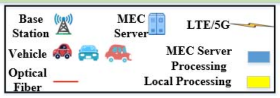


Fig. 2. The computing offloading model in vehicular network

computing tasks. In addition, the BS also connects with the cloud server. In this paper, we assume that the edge server can accomplish the vehicle's tasks, thus we do not consider offloading to remote cloud. As for vehicles, they can communicate with the BS through Long Term Evolution (LTE) or 5G. We consider that cellular networks can completely cover urban areas [15,16]. In order to facilitate the analysis, we consider a zone, and the model can be extended to other zones.

As shown in Fig. 2, we consider an area based on a vehicular network, and divide the vehicular network into three layers. There are N vehicles with different computing capabilities distributed at the terminal layer. We divide the time into equally spaced time slots, and the vehicle terminal will generate a task in each slot,. The communication service layer has a BS that can communicate with the vehicles and evenly allocate bandwidth to them. The edge service layer is distributed with M heterogeneous edge servers. At the beginning of each slot, vehicles allocate their speeds independently and randomly, and there is no correlation between speeds. Let $\mathrm{M} = \{1,2,\ldots \mathrm{M}\}$ and $\mathrm{N} = \{1,2,..\mathrm{N}\}$ be the sets of edge servers and vehicles, respectively. We define the task generated by vehicle-n as $T_{\mathrm{n}} = \{D_n^{in}, C_n, T_n^{max}\}$, where D_n^{in} (in bits) is the size of the task input data, C_n (in CPU cycles) is the required total CPU cycles to accomplish the task, and T_n^{max} denotes maximum permissible processing delay of the task. A vehicle can only be served by one edge server in a slot when the offloading is determined. The task offloading decisions of vehicle-n can be denoted as:

$$\mathbf{a}_{\mathrm{n}} = \{x_{n,0}, x_{n,1}, x_{n,2} \ldots x_{n,m} \ldots x_{n,M}\}, \forall m \in M \tag{1}$$

where $x_{n,0}$, $x_{n,m} \in [0, D_i^{in}]$. $x_{n,0}$ represents the size of task T_{n} executed locally, $x_{n,m}$ represents the size of task T_{n} offloaded to edge sever m. It is assumed that one vehicle can only be served by one edge server in a slot, thus the following constraint should be satisfied:

$$\left\{ \begin{array}{l} x_{n,0} + x_{n,m} = D_n^{in} \\ \sum_{i \in M, i \neq m} x_{n,i} = 0 \end{array} \right. \tag{2}$$

We denote the computational capacity of vehicles are $F_{vehicle} = \{f_1^V, f_2^V, f_3^V ... f_N^V\}$ and the computational capacity of the edge servers are $F_{MEC} = \{f_1^{MEC}, f_2^{MEC}, f_3^{MEC} ... f_M^{MEC}\}$.

A. Communication Model

In the MEC system of a vehicular network, we consider a 5G macro-cell BS equipped with K antennas. There are N single-antenna vehicles within the covered region of the macro-cell BS. The BS manages the vehicles through uplink transmissions and employs linear detection algorithm zero-forcing (ZF) [17]. Here, we consider the number of antennas at the BS is larger than the number of vehicles, i.e., $K > N$. For each time slot $t \in \mathcal{T}$, $h_n(t) \in \mathbb{C}^{K\times 1}$ represents the channel vector of each vehicle $n \in \{1,2,...N\}$, and the received signal of the BS can be expressed as

$$\mathbf{y}(t) = \sum_{n=1}^{N} \mathbf{h}_n(t)\sqrt{p_{o,n}(t)}s_n(t) + \mathbf{n}(t) \tag{3}$$

where $p_{o,n}(t)$ is the transmission power of vehicle *n*, $s_n(t)$ is the data symbol with unit variance, and $\mathbf{n}(t) \sim \mathcal{CN}(0, \sigma_R^2 \mathbf{I}_K)$ is a vector of additive white Gaussian noise with variance σ_R^2. We use the time correlation autoregressive (AR) model [18] to represent the transformation of the channel state between time slot t and $t-1$

$$\mathbf{h}_n(t) = \rho_n \mathbf{h}_n(t-1) + \sqrt{1-\rho_n^2}\,\mathbf{e}(t) \tag{4}$$

where ρ_n is the normalized channel correlation coefficient between slot t and $t-1$, and $\mathbf{e}(t)$ is the error vector, which is complex Gaussian distribution and is uncorrelated with $\mathbf{h}_n(t)$.

Let $\mathbf{H}(t) = [\mathbf{h}_1(t),...,\mathbf{h}_N(t)]$ be the $K \times N$ channel matrix between the BS and N vehicles. The ZF linear decoder can be given as

$$\mathbf{H}^{\dagger}(t) = \left(\mathbf{H}^H(t)\mathbf{H}(t)\right)^{-1}\mathbf{H}^H(t) \tag{5}$$

Let $\mathbf{g}_n^H(t)$ represents the *n*-th row of $\mathbf{H}^{\dagger}(t)$, and $\mathbf{g}_i^H(t)\mathbf{h}_j(t) = \delta_{ij}$ for ZF detection [16]. Here, $\delta_{ij}=1$ when $i = j$ and 0 otherwise. Therefore, the received signal for vehicle *n* can be calculated by

$$\mathbf{g}_n^H(t)\mathbf{y}(t) = \sqrt{p_{o,n}(t)}s_n(t) + \mathbf{g}_n^H(t)\mathbf{n}(t) \tag{6}$$

The signal-to-interference-plus-noise ratio (SINR) of vehicle n at time slot *t* can be given by

$$\gamma_n(t) = \frac{p_{o,n}(t)}{\sigma_R^2\|\mathbf{g}_n\|^2} = \frac{p_{o,n}(t)}{\sigma_R^2\left[\left(\mathbf{H}^H(t)\mathbf{H}(t)\right)^{-1}\right]_{nn}} \tag{7}$$

where $[\mathbf{A}]_{nk}$ is the (n,k)-th element of matrix $\mathbf{A}$.

The maximum achievable data rate between vehicle *n* and the BS can be calculated as

$$r_n(t) = B\log_2\left(1+\gamma_n(t)\right) \tag{8}$$

B. Computation Model

1) Local computing

The time latency of the task when it is processed locally can be calculated by

$$t_n^L = \frac{x_{n,0}\cdot C_n}{D_n^{in}\cdot f_n^V} \tag{9}$$

Vehicle *n*'s energy consumption is given by

$$E_n^L = \kappa(f_n^V)^2 \cdot \frac{x_{n,0}}{D_n^{in}} \cdot C_n \tag{10}$$

Here, $x_{n,0}$ is the data size of task processed locally, and κ is the effective switching capacitance in the chip [19].

2)Edge computing

The processing time of the MEC server computing model includes the transmission time and response time. We neglect the computing delay while the MEC server transmits the computing results to the vehicle. The transmission time refers to the time of uploading task data from the vehicle to the BS. According to (8), transmission time is presented as

$$t_{n2B}^{up} = \frac{x_{n,m}}{r_n} \tag{11}$$

Here, $x_{n,m}$ is the data size of the task transmitted to the BS.

As for the response time, it contains the time in the queue and performing calculation on MEC. For each edge server, assuming that it uses the first-come-first-served strategy to server the arriving computing tasks, the response time can be expressed as

$$\begin{aligned} t_{n,m}^{resp} &= t_{n,m}^{queue} + t_{n,m}^{exe} \\ t_{n,m}^{queue} &= \frac{\sum_{j\in k, j\neq n} x_{j,m}\cdot C_j}{D_j^{in}\cdot f_m^{MEC}} \\ t_{n,m}^{exe} &= \frac{x_{n,m}\cdot C_n}{D_n^{in}\cdot f_m^{MEC}} \end{aligned} \tag{12}$$

where $x_{j,m}$ is the size of the task arrives at MEC *m* before vehicle *n*'s task.

The energy consumption of the task offloaded to the server for processing is equivalent to the upload energy consumption. Therefore, the energy consumption of task generated by vehicle *n* and offloaded to the server *m* is defined as

$$E_n^{MEC} = p_{n2B}\cdot t_{n2B}^{up} \tag{13}$$

where p_{n2B} is the transmit power of the vehicle *n*.

III. Problem formulation

In this section, we first describe the indicators for evaluating the algorithm performance, then formulate the optimization problem of this paper.

A. Optimization Indicators

Considering the characteristic of selfishness, users tend to choose edge servers with richer resources. In order to make full use of the resources of each server and achieve load balancing, we define the load factor as

$$LF_m(\mathrm{t}) = \frac{\sum_{n\in N} x_{n,m} \cdot C_n}{D_n^{in} \cdot f_m^{MEC}} \tag{14}$$

According to (14), we propose the parameter of load balancing factor LBF, which can reflects the load balancing situation of the edge system, and it can be formulated by

$$LBF = \frac{\sum_{j\in M} (LF_j - (\frac{\sum_{m\in M} LF_m}{M}))^2}{M} \tag{15}$$

In order to reflect the performance improvement of the optimization objective relative to local processing, we use the system utility as the evaluation metric. The system utility can be defined as follow

$$U = \sum_{n\in N} (\lambda_t \frac{T_n^{Local} - T_n}{T_n^{Local}} + \lambda_e \frac{E_n^{Local} - E_n}{E_n^{Local}}) - \lambda_l LBF \tag{16}$$

where λ_t, $\lambda_e \in [0,1]$ and $\lambda_t + \lambda_e = 1$. λ_l can be adjusted. T_n^{Local} and E_n^{Local} are the delay and energy consumption of task generated by vehicle-*n* executed locally, respectively.

B. Optimization Problem

The optimization problem can be formulated as follows

$$\begin{aligned} &P1: \max \mathrm{U} \\ &\text{s.t.} \quad C1: \{ \begin{matrix} x_{n,0} + x_{n,m} = D_n^{in} \\ \sum_{i\in M,\ i\neq m} x_{n,i} = 0 \end{matrix} \\ &\qquad C2: \mathrm{T}_n \le T_n^{\max} \\ &\qquad C3: \mathrm{E}_n \le E_n^{\max} \end{aligned} \tag{17}$$

In the set of constraints, constraint (C1) guarantees that the task can be executed locally and on MEC at the same time. In addition, it makes sure each vehicle can only be served by an edge server at a slot. Constraint (C2) guarantees that the execution time of the task will not exceed the maximum latency. Constraint (C3) ensures that the energy consumption of the task will not exceed the total resources of the vehicle-*n*.

The optimization problem (P1) is a mixed-integer programming problem, which is difficult to solve in general. In the next section, we propose an approximate algorithm based on deep learning to efficiently and effectively solve (P1).

IV. REINFORCEMENT LEARNING-BASED SOLUTION

In order to enhance the robustness of the system and protect the privacy of users, we adopt a distributed offloading structure. In this section, we first define the state space, action space, and reward function in reinforcement learning. Then the DDPG algorithm is introduced. Finally, considering improving the learning efficiency of the DDPG algorithm, we propose the DLPR-DDPG algorithm.

A. State, Action, and Reward

1)State Space:

We define $\mathbf{S}$ as the state space, and the state $s(t)$ can be expressed as

$$\begin{aligned} s(t) = [&d_1^{MEC}(t-1), d_2^{MEC}(t-1) \ldots d_M^{MEC}(t-1), \\ &x_1(t-1), x_2(t-1) \ldots x_n(t-1) \ldots x_N(t-1), \\ &\mathbf{h}_1^T(t), \quad \mathbf{h}_2^T(t) \ldots \quad \mathbf{h}_n^T(t) \ldots \quad \mathbf{h}_N^T(t)] \end{aligned} \tag{18}$$

where

$d_m^{MEC}(t-1)$: The data size of the task processed by the *m*-th edge server in the last slot

$x_n(t-1)$: The data size of the task offloaded by the *n*-th vehicle in the last time slot

$\mathbf{h}_n^T(t)$: The transposed matrix of channel vector $\mathbf{h}_n(t)$.

2)Action Space:

We define the action space is $\mathbf{A} = \{a_n\}_{n\in N}$, and the action of an agent can be described by

$$\mathbf{a}_n = [x_{n,0}, x_{n,1}, x_{n,2} \ldots x_{n,M}] \tag{19}$$

3)Reward Function

In this paper, we have $\mathbf{R} = \{r_n\}_{n\in N}$, and the r_n can be expressed as

$$r_n = \lambda_t \frac{T_n^{Local} - T_n}{T_i^{Local}} + \lambda_e \frac{E_n^{Local} - E_n}{E_i^{Local}} - \lambda_l LBF \tag{20}$$

B. Deep Reinforcement Learning Based DDPG

In this article, we use the DDPG algorithm to solve the optimal problem P1. There are four deep neural networks in DDPG, which are online policy network, target policy network, online Q network, and target Q network. Fig. 3 shows the architecture of the DDPG algorithm.

In the actor network, we define the parameters of the online policy network as $\theta^{\mu} = \{\theta_1, \theta_2, \theta_3, \ldots \theta_n\}$ and the parameters of the target policy network as $\theta^{\mu'} = \{\theta_1', \theta_2', \theta_3', \ldots \theta_n',\}$. In the process of generating action, in order to balance the relationship between exploration and exploitation, Ornstein-Uhlenbeck stochastic noise ξ_t is added [20]. The output action of the

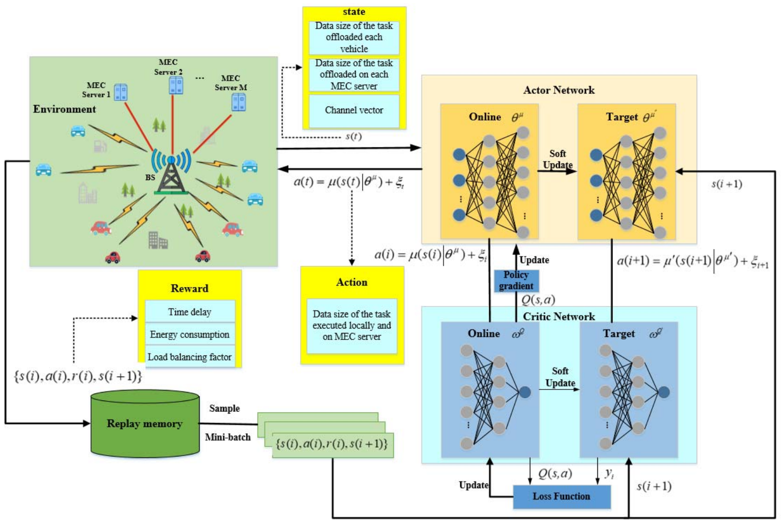


Fig. 3. Structure of the DDPG algorithm

online policy network can be calculated by

$$a(\mathrm{t}) = \mu(\mathrm{s}(\mathrm{t})|\theta^{\mu}) + \xi_t \tag{21}$$

After the online policy network taking action, the agent interacts with the environment to obtain the instant rewards and state in the next time slot. Also, the transition (s_t, a_t, r_t, s_{t+1}) will be stored in the replay memory. The objective function can be written as

$$J(\mu_\theta) = \int_S d\mu(s) Q(s, \mu_\theta(s)) ds \tag{22}$$

where $Q(s, \mu_\theta(s)$ can be obtained through the critic network.

The actor network μ updates its weights in the direction of getting larger cumulative discounted reward, that is

$$\begin{aligned}\nabla_\theta J(\mu_\theta) &= \int_S d^{\mu}(s) \nabla_\theta \mu_\theta(\mathrm{s}) \nabla_\theta Q(s,a)\big|_{a=\mu_\theta(\mathrm{s})} ds \\ &= E_\mu[\nabla_\theta \mu_\theta(\mathrm{s}) \nabla_\theta Q(s,a)\big|_{a=\mu_\theta(\mathrm{s})}]\end{aligned} \tag{23}$$

Then, we define the parameters of the online Q network as $\omega^{Q} = \{\omega_1, \ \omega_2, \ \omega_3, ...\omega_n\}$, and define the parameters of the target Q network as $\omega^{Q} = \{\omega'_1, \ \omega'_2, \ \omega'_3, ...\omega'_n\}$. We sample mini-batch transitions from the replay memory to train the critic network, and assume that the transition i is (s_i, a_i, r_i, s_{i+1}). Taking (s_i, a_i) as the input of the online Q network, the output of the network is $Q_\omega(s_i, a_i)$. In addition, the target Q value can be expressed by

$$y_i = r(i) + \gamma \cdot \mathrm{Q}'(s(i+1), \mu'(s(i+1)|\theta^{\mu'})|\omega^{Q'}) \tag{24}$$

Therefore, the mean square error L can be calculated by

$$L = \frac{1}{G}\sum_i (y(i) - \mathrm{Q}(s(i), a(i)))^2 \tag{25}$$

Here, G is the size of the mini-batch. The parameters of the online Q network can be updated by minimizing (25). Then, the target Q network can be updated by

$$\begin{aligned}\omega^{Q'} &\leftarrow \tau\omega^{Q} + (1-\tau)\omega^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}\end{aligned} \tag{26}$$

where τ is 0.001 in general.

C. Prioritized Experience Replay based DDPG Algorithm

DDPG adopts the method of experience replay. Usually, when we sample in the replay buffer, we perform uniform sampling. But it must be aware that there are differences in the importance of samples. Based on this, we propose a DDPG algorithm considering prioritized experience replay, which sorts the transitions in the replay memory according to importance. In this article, we use absolute TD-error to indicate the importance of the sample. Absolute TD-error of experience j can be calculated as

$$\sigma_j = |r(t) + \gamma \cdot \mathrm{Q}'(s(t+1), \mu'(s(t+1))) - \mathrm{Q}(s(t), \mu(s(t)))| \tag{27}$$

Then sort the experiences in the replay memory according to absolute TD-error, and replay the relatively large transitions of absolute TD-error more frequently. Let rank(j) is the ranking result of experience j according to absolute TD-error. The probability of the sampled experience j can be defined as

$$P(j) = \frac{(z_j)^{\alpha}}{\sum_g (z_g)^{\alpha}} \tag{28}$$

where $z_j = 1/rank(j)$, and α is the prioritization parameter.

Finally, in order to correct the bias introduced by changing the state visitation frequency, we use importance sampling weights.

$$\phi_j = \frac{1}{W^{\varphi} \cdot P(\mathrm{j})^{\varphi}} \tag{29}$$

Here, W is the size of the replay buffer, $P(j)$ is the probability of the sampled experience j, and the parameter φ controls to what extent the correction is used.

The DLPR-DDPG algorithm is shown in Algorithm 1.

V. PERFORMANCE EVALUATION

In this section, we will evaluate the performance of the DLPR-DDPG Algorithm through computer simulations.

```
Algorithm 1 DLPR-based DDPG Algorithm
1:Initialize online actor network with $\theta^{\mu}$ and online critic network with $\omega^{Q}$
2:Initialize target actor network with $\theta^{\mu'} \leftarrow \theta^{\mu}$
3:Initialize target critic network with $\omega^{Q'} \leftarrow \omega^{Q}$
4:Initialize the size of replay memory buffer $W$ and sample frequency $f_{sample}$
5: Initialize updating rate $\xi$ of the target network, and mini-batch $G$
6:for episode=1, EP do
7:  Initialize a random process for action exploration
8:  Initialize observation state $s_1$ of the environment
9:  for each step $t$=1, T do
10:     Select action with (21) according to current policy
11:     Execute action $a_t$ and interact with the environment
12:     Obtain reward $r_t$ and the next state $s_{t+1}$
13:     Store transition $(s_t, a_t, r_t\ s_{t+1})$ in replay memory buffer
14:     if $t > W$ and $t\ \%\ f_{sample} = 0$ then
15:        for $j$=1, $G$ do
16:          Calculate priority of transition $j$ according to absolute TD-error
17:          Sample transition j with probability $P(j)$
18:          Compute corresponding importance-sampling weight
19:        end for
20:        Calculate the $y_i$ in the target critic network with (24)
21:        Update online critic network according to (25)
22:        Update online actor network according to (23)
23:        Update the target policy network and target Q networks according
           to (26) with an updating rate $\xi$
24:        Update $s_t$, with $s_{t+1}$,
25:        BS broadcasts the load factors of all MEC servers
26:     end if
27:  end for
28:end for
```

A. Simulation Setup

In the scheme, the input data size of the task is chosen from [100KB, 500KB], and the range of CPU cycles required to complete a task is [40Mc, 200Mc]. In addition, the maximum time delay allowed by the task is chosen from [1s, 2s]. At the beginning of each episode, the channel vector of vehicle n is initialized as $\mathbf{h}_n(0) \sim \mathcal{CN}(0, h_0(d_0/d_n)^{\varpi}\mathbf{I}_K)$, where the path-loss constant $h_0 = -30$dB, the reference distance $d_0 = 1$m, the path-loss exponent $\varpi = 3$, and d_n is the distance between the vehicle n and the BS. In the meanwhile, the bandwidth allocated to each vehicle is set to 20MHz. The transmission power and the CPU cycle frequency of each vehicle is chosen from [0.25W, 1W] and [0.5GHz, 2GHz], respectively. Moreover, the CPU cycle frequency of MEC servers is chosen from 4GHz to 8GHz.

As for the design of the four networks of the DDPG, we conduct an input layer, two hidden fully-connected layers and an output layer. The two hidden layer contains 200 nodes and 100 nodes, respectively. The learning rate for the actor network and target network are 0.0001 and 0.001, respectively. The size of the experience replay buffer is set to 100, and the mini-batch is set to 32.

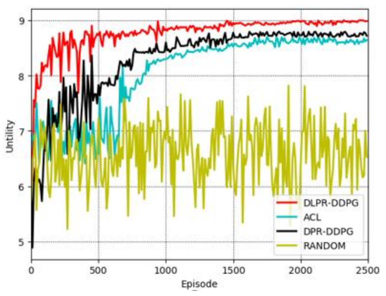


Fig. 4. The untility of the system

B. Performance Comparison

In this part, we set up baseline algorithms, which are the priority experience replay based DDPG algorithm that does not consider load balancing in distributed system (DPR-DDPG), the actor-critic algorithm with load balance coefficient (ACL), full offloading algorithm and random algorithm.

1) Convergence

In Fig. 4, the proposed DLPR-DDPG algorithm is compared with the baseline algorithms in terms of convergence. We assume that there are 10 vehicles in a zone, and the BS connects to 6 MEC servers. Fig. 4 shows the utility in different episodes. It is observed that the DLPR-DDPG algorithm has converged in the 1000-th episode. Although the ACL algorithm and DPR-DDPG can also converge, the speeds are much slower than the DLPR-DDPG algorithm. In addition, the DLPR-DDPG algorithm can achieve the highest utility. As it is expected, the random algorithm is the worst among the four algorithms. Therefore, compared with the other three algorithms, the fastest convergence speed and the highest utility are achieved by our proposed algorithm.

2) The number of vehicles

In Fig. 5, the average time of processing tasks versus the numbers of vehicles is investigated. It can be seen that the average time of vehicles increases nonlinearly with the increasing number of vehicles. To further analysis the reason for the results, when the number of vehicles is relatively small, MEC servers have sufficient resources and the average time for vehicles to complete tasks increases slowly. However, when the number of vehicles is large, it is difficult for MEC to meet the needs of vehicles due to the large size of the tasks. At this time, the average time for vehicles to complete tasks will increase significantly. Nevertheless, we can see that the average time of the DLPR-DDPG algorithm is much lower than that of the ACL algorithm and full offload strategy. In addition, the average time growth rate of the DLPR-DDPG algorithm is also the lowest, which indicates the DLPR-DDPG algorithm outperforms other baseline algorithms in terms of robustness.

Fig. 6 investigates the energy consumption improvement relative to local processing with the increasing of the vehicles' number. The DLPR-DDPG algorithm improves the most relative to local processing strategy. Comparing with the DLPR-DDPG, the energy consumption improvement of the ACL is slightly worse. With no doubt, the full offload strategy is the worst. Moreover, as the number of vehicles increases, the proposed algorithm can achieve the lowest decline rate of the performance.

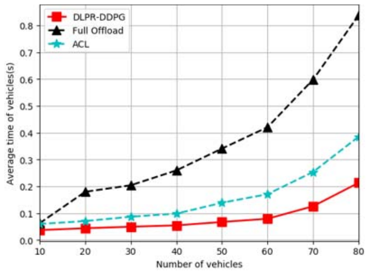


Fig. 5. Comparison of average time under different numbers of vehicles

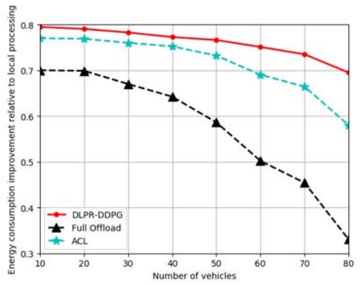


Fig. 6. Energy consumption improvement relative to local processing under different numbers of vehicles

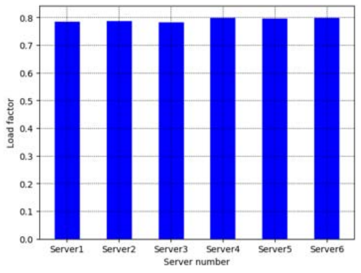


Fig. 7. Load factors of the MEC servers

3) Load balance

Fig. 7 shows the load balancing results achieved by the DLPR-DDPG algorithm. According to our assumptions, the computing power of each MEC server is different, but as shown in Fig. 7, the load of each server is balanced. Since we regard the load balancing as one of the factors that affect the system utility. In summary, the DLPR-DDPG algorithm is effective in balancing the load of the servers.

VI. Conclusion

In this paper, we consider a vehicular network with multiple users and multiple MEC servers. Considering a more realistic scenario, the time-varying channel is assumed. Moreover, a joint optimization problem is formulated to minimize the delay and energy consumption as well as to achieve a better load balancing. To solve the problem, reinforcement learning is utilized to make offloading decision. Numerical results are compared with four baseline algorithms and shows the superiority of our proposed algorithm in reducing delay and energy consumption and balancing server load. For future work, it should be meaningful to study the computing offloading problems through cooperating with remote cloud.

References

[1] D. Zhang, T. He, and F. Zhang, “Real-time human mobility modeling with multi-view learning,” ACM Trans. Intell. Syst. Technol 9, 3 (2018), 22.

[2] Z. Tong, H. Chen, X. Deng, K. Li, “A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization,” Soft Comput. 23 (21) (2019) 11035–11054.

[3] K. Gai, M. Qiui, “Optimal resource allocation using reinforcement learning for IoT content-centric services,” Applied Soft Computing 70, 12-21, 2018

[4] L.Lin, X.Liao, H. Jin, and P.Li, “Computation offloading toward edge computing,” Proc. IEEE, vol. 107, no. 8, pp. 1584–1607, Aug. 2019.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “Mobile edge computing: Survey and research outlook,” [Online]. Available: https://arxiv.org/abs/1701.01090

[6] Y. Mao, J. Zhang, KB. Letaief, “Dynamic computation offloading for mobile-edge computing with energy harvesting devices,” IEEE Journal on Selected Areas in Communications, 2016,34(12):3590−3605.

[7] X. Chen, L. Jiao, W. Li and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” IEEE/ACM Trans. On Networking, 2016,24(5):2795−2808.

[8] Y. Duan, C. She, G. Zhao, and TQS. Quek, “Delay Analysis and Computing Offloading of URLLC in Mobile Edge Computing System,” Hanzhou, China: 2018 10th International Conference on Wireless Communications and Signal Processing(WCSP), 2018.

[9] C. You, K. Huang, “Multiuser resource allocation for mobile-edge computation offloading,” In: Proc. of the 2016 IEEE Global Communications Conf. (GLOBECOM). IEEE, 2016. 1−6.

[10] O. Munoz, AP. Iserte, J. Vidal, and M. Molina, “Energy-latency trade-off for multiuser wireless computation offloading,” In: Proc. of the Wireless Communications and Networking Conf. Workshops. IEEE, 2014. 29−33.

[11] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, “Mobile Edge Computing Empowered Energy Efficient Task Offloading in 5G,” IEEE Transactions on Vehicular Technology, 2018:1-1.

[12] N. Luong, Z. Xiong, P. Wang, and D. Niyato, “Optimal auction for edge computing resource management in mobile blockchain networks: A deep learning approach,” In Proceedings of the IEEE International Conference on Communications (ICC’18). IEEE, 1–6.

[13] J. Zhang, H. Guo, and J. Liu, “Adaptive Task Offloading in Vehicular Edge Computing Networks: a Reinforcement Learning Based Scheme,” Mobile Netw Appl 25, 1736–1745 (2020).

[14] H. Ke, J. Wang, L. Deng, Y. Ge and H. Wang, “Deep Reinforcement Learning-Based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks,” in IEEE Transactions on Vehicular Technology, vol. 69, no. 7, pp. 7916-7929, July 2020, doi: 10.1109/TVT.2020.2993849.

[15] Z. Ning, “A cooperative quality-aware service access system for social Internet of vehicles,” IEEE Int. Things J. 5, 4 (2018), 2506–2517.

[16] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, “Fog following me: Latency and quality balanced task allocation in vehicular fog computing,” In Proceedings of the 15th IEEE International Conference on Sensing, Communication, and Networking (SECON’18). IEEE, 1-9.

[17] H. Ngo, E. Larsson, and T. Marzetta, “Energy and spectral efficiency of very large multiuser mimo systems,” IEEE Trans. Commun., vol. 61, no. 4, pp. 1436–1449, 2013.

[18] H. Suraweera, T. Tsiftsis, G. Karagiannidis, and A. Nallanathan, “Effect of feedback delay on amplifyand-forward relay networks with beamforming,” IEEE Transactions on Vehicular Technology, vol. 60, no. 3, pp. 1265–1271, 2011.

[19] Y. Mao, J. Zhang, S. Song, K.B. Letaief, “Power-delay tradeoff in multi-user mobile-edge computing systems,” in: 2016 IEEE Global Communications Conference (GLOBECOM), IEEE, 2016, pp. 1–6.

[20] M. Li, J. Gao, L. Zhao and X. Shen, “Deep Reinforcement Learning for Collaborative Edge Computing in Vehicular Networks,” in IEEE Transactions on Cognitive Communications and Networking, vol. 6, no. 4, pp. 1122-1135, Dec. 2020, doi: 10.1109/TCCN.2020.3003036.