IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-0926-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOMWKSHPS54753.2022.9798187

SAC-based Computation Offloading and Resource Allocation in Vehicular Edge Computing

Yanlang Zheng*, Huan Zhou*, Rui Chen*, Kai Jiang†, and Yue Cao†
*College of Computer and Information Technology, China Three Gorges University, Yichang, China
†School of Cyber Science and Engineering, Wuhan University, Wuhan, China
{yanlangzheng, zhouhuan117}@gmail.com, chenrui@ctgu.edu.cn, {kai.jiang, yue.cao}@whu.edu.cn

***Abstract*—The Vehicular Edge Computing (VEC) provides powerful computing resources for intelligent terminals. However, the diversity of computing resources at edge nodes (i.e., edge servers and idle vehicles) and the mobility of vehicles impose great challenges on computation offloading. In this paper, we investigate the joint optimization problem of computation offloading and resource allocation in a cooperative vehicular network by exploiting idle vehicles and Road Side Units (RSUs) equipped with edge servers. In order to minimize the task completion time under latency constraint, a Soft Actor-Critic (SAC)-based algorithm is proposed to solve the problem. The simulation results show that the proposed SAC-based algorithm can effectively reduce the total latency of the system, and its performance is significantly better than other benchmark methods.**

***Index Terms*—collaborative computation offloading, resource allocation, Soft Actor-Critic, vehicle edge computing**

I. Introduction

In recent years, the transportation system has gradually evolved into an era driven by intelligent data along with the rapid development of various novel wireless access techniques. Meanwhile, as an important application scenario in 5G mobile communication, vehicles are able to provide computation-intensive and latency-sensitive applications with the aid of the emerging vehicular network, which effectively improves passenger driving safety and travel comfort [1]-[3]. However, due to limited computing resources, vehicles may not well support various vehicular applications, such as image-aided navigation and augmented vehicle reality. Therefore, it is imperative to offload part of the computation task to other devices with powerful computing resources, such as cloud servers. Nevertheless, cloud servers usually lead to high transmission latency because of long-distance, which are not suitable for processing latency-sensitive applications. Against this background, Mobile Edge Computing (MEC) is emerging as a new means to deal with the above problems and is widely used in various communication networks, such as vehicular network. In this case, MEC can be seen as Vehicle Edge Computing (VEC) [4]-[6]. Vehicles are able to offload tasks to Road Side Units (RSUs) equipped with edge servers by exploiting VEC, which effectively meets the substantial computation requirements for vehicles.

In recent years, a range of works have focused on computation offloading in vehicular networks, which can be divided into two categories according to the offloading approaches [7]. The first type is binary offloading. In this case, the whole task is either offloaded to the local vehicle or the RSU. For binary offloading, game theory has been used to deal with the optimization problem in VEC [8], [9]. In [10] and [11], the authors have modeled the binary offloading problem as a Markov Decision Process (MDP) and utilized Deep Reinforcement Learning (DRL) algorithm to minimize the long-term cost and system utility, respectively. In [12] and [13], the authors have employed blockchain technique to protect the privacy of the offloading tasks in a vehicular network. The second type is partial offloading, where the task is divided into two parts, one is offloaded to the local vehicle and the other to the RSU or the idle vehicle. For partial offloading, a DRL algorithm is proposed to solve the task partial offloading problem in the VEC network to minimize the execution latency [14]. In [15], the authors have proposed a vehicle-to-vehicle partial computation offloading scheme to maximize the mean utility of task vehicles. In [16], the authors have presented a joint allocation of wireless resources and computing resources algorithm to reduce the total latency of the VEC network.

However, the above studies do not take into account the cooperation between edge servers and idle vehicles. Meanwhile, due to the diversity of computing resources, the RSU equipped with a MEC server cannot meet the requirements of all computation offloading tasks. Therefore, it is of great significance to design a suitable computation offloading scheme to ensure the full utilization of edge nodes' computing resources. Motivated by this, this paper considers the cooperation between edge servers and idle vehicles at the same time and investigates the joint optimization problem of computation offloading and resource allocation in multi-vehicle networks. Then, we propose a Soft Actor-Critic (SAC)-based algorithm to solve the optimization problem. Extensive simulation results show that the proposed SAC-based algorithm can minimize the total system latency.

The rest of this paper is organized as follows. Section II introduces the main assumptions, system model as well as problem formulation. Section III proposes the SAC-based algorithm. Section IV presents the simulation results. Finally, conclusions are summarized in Section V.

II. System Model

In this section, we introduce the system model and the main assumptions, as well as problem formulation. Let us first

978-1-6654-0926-1/22/$31.00 ©2022 IEEE

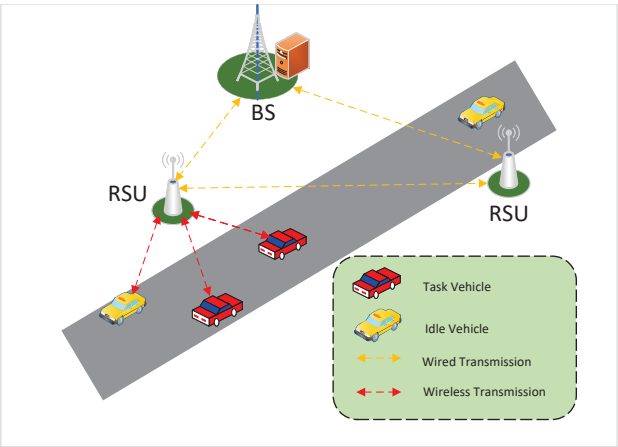

Fig. 1: Vehicle edge computing system architecture.

address the system model and main assumptions.

A. Network Structure and Main Assumptions

As shown in Fig. 1, we consider a vehicular network, which includes a Base Station (BS), K RSUs with MEC servers, N Task Vehicles (TVs) as well as M Idle Vehicles (IVs). In our vehicle network, BS is used to collect and update the global status information continuously, such as Channel Status Information (CSI), edge nodes resource utilization status, service requirements, etc. The RSU and BS as well as other RSUs are wired connect, which can not only receive service requests from the vehicle within its communication range, but also execute the decision of the BS. It is assumed that all vehicles within the coverage of the RSU are randomly distributed on the road and travel at a constant speed.

We assume that the system operates within a fixed slot length $t \in \{1, 2, ..., T\}$, where T represents a finite time range. In each time slot, each TV, say TV_i $(i \in 1, 2, ..., N)$, generates a latency-sensitive task, which is presented as $\psi_{i,t} \triangleq \{C_{i,t}, D_{i,t}, \tau_{i,t}\}$. $C_{i,t}$ denotes the total number of Central Processing Unit (CPU) cycles required to complete the task, which obeys uniformly distributed, $D_{i,t}$ indicates the data size of the task, which obeys uniformly distributed, $\tau_{i,t}$ is the latency constraint of the task.

In our model, TV_i transmits the service request to the BS with the aid of the associated RSU. Considering that the data size of the service request is small, the transmission latency of the request is negligible. After receiving the service request, the BS generates an optimal allocation scheme based on the current global information, and broadcasts to the edge nodes, which includes the RSUs and the IVs. Then, the offloaded proportion of the task is transmitted from the associated RSU to TV_i. After receiving the offloaded task, the associated RSU divides it into several subtasks and transmits them to the cooperative RSUs as well as the IVs within its communication range for collaborative processing. Moreover, when TV_i is away from the communication range of the current RSU, assume that, the BS can predict the next RSU that TV_i arrives with the aid of global information.

Moreover, it is assumed that transmission interference between vehicle and RSU can be ignored by exploiting Orthogonal Frequency Division Multiple Access (OFDMA) technique. Then, in the time slot t, the achievable rate from TV_i and IV_j to the associated RSU are, respectively, given by

$$R_{i,t} = B_{i,t}\log_2(1 + \frac{P_i h_{i,t}}{\sigma_{i,t}^2}), \tag{1}$$

$$R_{j,t} = B_{j,t}\log_2(1 + \frac{P_j h_{j,t}}{\sigma_{j,t}^2}), \tag{2}$$

where $B_{i,t}$ and $B_{j,t}$ are the bandwidth occupied by TV_i and IV_j, P_i and P_j are the transmitting power of TV_i and IV_j, $h_{i,t}$ and $h_{j,t}$ are the channel gain from TV_i and IV_j to the associated RSU, $\sigma_{i,t}^2$ and $\sigma_{j,t}^2$ are the local noise of TV_i and IV_j, respectively.

Since the RSUs are connected through broadband, the transmission rate can be defined as $R_{k,t}$.

B. Latency

1) Offloading Latency: It is assumed that TV_i can offload part of or the whole task $\psi_{i,t}$, $\theta_{i,t}\psi_{i,t}$ and $(1 - \theta_{i,t})\psi_{i,t}$ are the task performed locally by TV_i and offloaded to the associated RSU. In our system, the offloading latency consists of transmission latency and computation latency. Let us first introduce transmission latency.

As can be seen from the above, when TV_i transmits the offloaded task, the latency is given by

$$T_{i,t}^{\mathrm{Trans}} = \frac{(1 - \theta_{i,t})D_{i,t}}{R_{i,t}}, \tag{3}$$

where $\theta_{i,t} \in [0, 1]$ indicates the local processing proportion of TV_i .

According to the received scheme, the associated RSU divides the offloaded task into several subtasks to form the task allocation vector $\boldsymbol{\mathcal{L}} = \{\psi_{i,1,t}, ..., \psi_{i,M,t}, \psi_{i,(M+1),t}, ..., \psi_{i,(M+K),t}, \psi_{i,r,t}\}$, and distributes them to the cooperative RSUs and IVs for computing. The transmission latency from the associated RSU to IV_j, the cooperative RSU_k are, respectively, given by

$$T_{j,t}^{\mathrm{Trans}} = \frac{\theta_{i,j,t}D_{i,t}}{R_{j,t}}, \tag{4}$$

$$T_{k,t}^{\mathrm{Trans}} = \frac{\theta_{i,k,t}D_{i,t}}{R_{k,t}}, \tag{5}$$

where $\theta_{j,t}D_{i,t}$ and $\theta_{k,t}D_{i,t}$ are the data size of subtask allocated to IV_j and the cooperative RSU_k, respectively.

After receiving the offloaded task, these edge nodes allocate corresponding computing resources. The computation latency of offloaded task to IV_j, the cooperative RSU_k and the associated RSU_r are, respectively, given by

$$T_{j,t}^{\mathrm{Comp}} = \frac{\theta_{i,j,t}C_{i,t}}{F_{i,j,t}}, \tag{6}$$

$$T_{k,t}^{\mathrm{Comp}} = \frac{\theta_{i,k,t}C_{i,t}}{F_{i,k,t}}, \tag{7}$$

$$T_{r,t}^{\mathrm{Comp}} = \frac{\theta_{i,r,t} C_{i,t}}{F_{i,r,t}}, \tag{8}$$

where $\theta_{i,j,t}C_{i,t}$, $\theta_{i,k,t}C_{i,t}$ and $\theta_{i,r,t}C_{i,t}$ denote the required CPU cycles to process subtasks assigned to IV_j, the cooperative RSU_k and the associated RSU_r. $F_{i,j,t}$, $F_{i,k,t}$ and $F_{i,r,t}$ denote the computing resources allocated to the subtasks by IV_j, the cooperative RSU_k and the associated RSU_r, respectively.

From the above, each edge node can perform the subtask in parallel with the aid of the task allocation vector $\boldsymbol{\mathcal{L}}$. Therefore, the offloading latency of TV_i is given by

$$T_{i,t}^{\mathrm{Offload}} = max\{T_{j,t}^{\mathrm{Trans}} + T_{j,t}^{\mathrm{Comp}}, T_{k,t}^{\mathrm{Trans}} + T_{k,t}^{\mathrm{Comp}}, T_{r,t}^{\mathrm{Comp}}\}, \tag{9}$$

2) Local Latency: The latency that TV_i executes the task $\theta_{i,t}\psi_{i,t}$ locally is expressed as

$$T_{i,t}^{\mathrm{Local}} = \frac{\theta_{i,t} C_{i,t}}{F_i}, \tag{10}$$

where F_i represents the computing capability of TV_i itself.

Therefore, the total execution latency of the task $\psi_{i,t}$ depends on the maximum between the local execution latency and the offloaded execution latency, given by

$$T_{i,t}^{\mathrm{Total}} = max\{T_{i,t}^{\mathrm{Local}}, T_{i,t}^{\mathrm{Offload}} + T_{i,t}^{\mathrm{Trans}}\}, \tag{11}$$

C. Problem Formulation

In this paper, we investigate the optimization problem of joint of computation offloading strategy $\boldsymbol{\mathcal{X}_t}$ and resource allocation strategy $\boldsymbol{\mathcal{Z}_t}$ to minimize the task completion time under latency constraint, which can be written as

$$\min_{\boldsymbol{\mathcal{X}_t}, \boldsymbol{\mathcal{Z}_t}} \ \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{N} T_{i,t}^{\mathrm{Total}} \tag{12}$$

$$s.t. \qquad (1 - \theta_{i,t}) = \sum_{j=1}^{M} \theta_{i,j,t} + \sum_{k=M+1}^{M+K} \theta_{i,k,t} + \theta_{i,r,t}, \tag{12a}$$

$$\sum_{i \in \mathcal{N}} F_{i,k,t} \leq F_k, \forall k \in \mathcal{K}, \tag{12b}$$

$$\sum_{i \in \mathcal{N}} F_{i,j,t} \leq F_j, \forall j \in \mathcal{M}, \tag{12c}$$

$$\sum_{i \in \mathcal{N}} F_{i,r,t} \leq F_r, \tag{12d}$$

$$T_{i,t}^{\mathrm{Total}} \leq \tau_{i,t}. \tag{12e}$$

where (12a) is the offloaded proportion of subtasks to IVs and the cooperative RSUs, as well as the associated RSU_r, which add up to the offloaded proportion of TV_i; (12b), (12c) and (12d) indicate the computing resources allocated to these subtasks cannot exceed the total computing resources of edge nodes. F_k, F_j, and F_r represent the total computing resources of the cooperative RSU_k, IV_j, and the associated RSU_r, respectively; (12e) indicates the maximum task tolerance latency.

III. Deepreinforcement Learning Approach

In this section, in order to minimize the task completion time under latency constraint, we propose the SAC-based algorithm to solve the optimization problem. Firstly, let us introduce the concepts about state function, action function as well as reward function.

A. Problem Formulation based on SAC

As can be seen from the above, the interaction process of TV_i and RSUs can be seen as a MDP, which can be denoted by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$, $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, and $\mathcal{P}$ represents the state transition probability, which denotes the probability density of the next state $s_{t+1} \in \mathcal{S}$ through the given current state $s_t \in \mathcal{S}$ and action $a_t \in \mathcal{A}$.

The motivation of the system is to find strategies that can maximize reward and select the best behavior in a dynamic state. Therefore, the key elements about MDP in our model can be defined as

- **State space**: The state space consists of two parts: the vehicle location set $\mathcal{VL}_t = \{VL_{i,t}, VL_{j,t}\}$ persents the location of TV_i and IV_j at time slot t, and the edge resources set $\mathcal{ER}_t = \{ER_{j,t}, ER_{k,t}, ER_{r,t}\}$ presents the remaining resources of IV_j, cooperative RSU_k and the associated RSU_r. Therefore, the sate space can be expressed as $s_t = \{\mathcal{VL}_t, \mathcal{ER}_t\}$.
- **Action space**: The action space consists of two parts: the computation offloading strategy $\mathcal{X}_t = \{x_{1,t}, ..., x_{i,t}, ..., x_{N,t}\}$ and the resource allocation strategy $\mathcal{Z}_t = \{z_{1,t}, ..., z_{i,t}, ..., z_{N,t}\}$. Therefore, the action space can be expressed as $a_t = \{\mathcal{X}_t, \mathcal{Z}_t\}$.
- **Reward**: The reward function can be represented by $r_t = -\sum_{t=1}^{T} \sum_{i=1}^{N} T_{i,t}^{\mathrm{Total}}$.

B. Proposed SAC-based Algorithm

SAC is an off-policy and stochastic policy algorithm that is centrally characterized by entropy regularization. Due to this feature, it can accelerate learning while preventing policy prematurely convergence to the local optimum. In our model, we convert the original standard RL maximum cumulative reward $\mathbb{E}\{\sum_{t=1}^{T} \gamma^t r_t | \pi\}$ into a more general maximum entropy objective after adding an expected entropy $\mathcal{H}(\pi(a|s)) = -log\pi(a|s)$, thus the objective of the expected entropy is given by

$$J(\pi) = \sum_{t=1}^{T} [\gamma^t [r_t - \alpha \mathcal{H}(\pi(a_t|s_t))] | \pi], \tag{13}$$

where $\gamma \in [0, 1]$ is the discount factor and distinguishes the importance between immediate and future rewards and α is the temperature parameter that controls the stochasticity of the optimal policy.

According to the architecture of actor-critic, we can divide the SAC algorithm into two parts: one actor network and four critic networks. Specifically, the V critic and target V critic network are used to output state value functions. Meanwhile, the two Q critic networks are utilized to output Q-value

Algorithm 1 SAC-based Algorithm for Computation Offloading and Resource Allocation.

Require: Initialize the parameters about the tested environment :φ, $\widehat{\varphi}$, θ_1, θ_2, ω, replay memory $\mathcal{D}$, the initial state s for available resources and maxium iteration T;

Ensure: Optimal sequences of actions, which include the computation offloading and resource allocation strategies to minimize the task completion time.

Initialize parameters: φ, $\widehat{\varphi}$, θ_1, θ_2, ω; empty replay memory set $\mathcal{D}$;
for episode in episodes **do**
  Initialize simulation environment setup;
  Randomly generate and receive an initial state s;
  for t from 1 to T **do**
    In actor network, select action a_t according to $\pi_\omega(\cdot|s_t)$ and execute it in the environment;
    Get the new state s_{t+1}, reward r_t;
    Store $\{s_t, a_t, r_t, s_{t+1}\}$ in the replay memory $\mathcal{D}$;
    $s_t = s_{t+1}$;
    Randomly smaple a batch of tuples, $\mathcal{B} = \{(s_t, a_t, r_t, s_{t+1})\}$ from the replay memory $\mathcal{D}$;
    Update Q critic network parameters θ_1, θ_2 by the gradient of loss function equation (17);
    $\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} L_Q(\theta_i)$ for $i \in \{1, 2\}$, where λ_Q is the learning rate of Q critic network;
    Update V critic network and target V critic network parameters φ, $\widehat{\varphi}$ by the gradient of loss function equation (19):
    $\varphi \leftarrow \varphi - \lambda_V \nabla_\varphi L_V(\varphi)$, where λ_V is the learning rate of V critic network;
    $\widehat{\varphi} \leftarrow \tau\varphi + (1-\tau)\varphi$;
    Update the actor network parameter ω by the gradient of loss function equation (23):
    $\omega(t+1) \leftarrow \omega(t) - \lambda_\pi \nabla_\omega L_\pi(\omega)$, where λ_π is the learning rate of actor network;
    If s_{t+1} is the end state, finish this current iteration. If not, go back to Step. (6);
  end for
end for

functions. Next, we define the parameters of networks: φ denotes the parameter of V critic network; $\widehat{\varphi}$ denotes the parameter of target critic network; θ_1 and θ_2 represent the two Q critic network parameters, respectively.

Then, the SAC-based algorithm in VEC can be expressed as follows. In the beginning, the centralized agent collects information from the environment to formulate the state space. After obtaining the initial state s_k, we input the current state into the actor network, which utilizes a parameterized Deep Neural Networks (DNN) to approximate policy $\pi(\cdot|s_k)$. Meanwhile, the environment state is transferred from s_k to s_{k+1} when the action a_k is implemented. Once the next state s_{k+1} and reward r_k are obtained, the tuple (s_k, a_k, r_k, s_{k+1}) is stored in the replay memory $\mathcal{D}$ to update network parameters by randomly sampling a tuple to break the correlations between samples.

From the above, the Q value function is expressed as

$$Q^\pi(s_k, a_k) = \sum_{t=0}^{T} [\gamma^t [r_t - \alpha \mathcal{H}(\pi(a_t|s_t))] | s_0 = s_k, a_0 = a_k, \pi], \tag{14}$$

At the same time, the relationship between Q-value function and value function can be expressed as

$$Q^\pi(s_k, a_k) = r_k + \gamma V^\pi(s_{k+1}), \tag{15}$$

Then, we randomly sample a batch of tuples $\mathcal{B}$ from the replay memory $\mathcal{D}$ to update the network. In our method, the Mean Squared Error (MSE) is used as the loss function to train the Q critic network with the aid of the optimal Bellman equation, given by

$$L_Q(\theta_i) = \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} [Q_{\theta_i}(s_k, a_k) - \widehat{Q}_{\widehat{\varphi}}(s_k, a_k)]^2, \quad i \in \{1, 2\} \tag{16}$$

where $\widehat{Q}_{\widehat{\varphi}}(s_k, a_k)$ satisfies $\widehat{Q}_{\widehat{\varphi}}(s_k, a_k) = r_k + \gamma \widehat{V}_{\widehat{\varphi}}(s_{k+1})$, $\widehat{V}_{\widehat{\varphi}}(s_{k+1})$ is the target state value. Hence, the gradient of $L_Q(\theta)$ can be expressed as

$$\begin{aligned} \nabla_{\theta_i} L_Q(\theta_i) = \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \nabla_{\theta_i} Q_{\theta_i}(s_k, a_k) (Q_{\theta_i}(s_k, a_k) \\ - r_k - \gamma \widehat{V}_{\widehat{\varphi}}(s_{k+1})), \quad i \in \{1, 2\} \end{aligned} \tag{17}$$

Then, the critic network can be updated by exploiting a batch of tuples $\mathcal{B}$, and the state value function can be evaluated by the entropy. Similarly, the MSE is utilized to measure the performance of the V critic network, given by

$$L_V(\varphi) = \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} (V_\varphi(s_k) - [Q_\theta(s_k, a_k) - \alpha log \pi_\varphi(a_k|s_k)])^2, \tag{18}$$

Correspondingly, the gradient of $L_V(\varphi)$ is expressed as

$$\begin{aligned} \nabla_\varphi L_V(\varphi) = \frac{1}{|\mathcal{B}|} \sum_{k=1}^{|\mathcal{B}|} \nabla_\varphi V_\varphi(s_k) [V_\varphi(s_k) - Q_\theta(s_k, a_k) \\ + \alpha log \pi_\varphi(a_k|s_k)], \end{aligned} \tag{19}$$

Finally, the actor network is responsible for mapping the states to the actions. Hence, its main task is to obtain the optimal policy according to equation (14), given by

$$\pi_{opt}(\cdot|s) = exp(\frac{1}{\alpha}(Q^\pi(s, a) - V^\pi(s)), \tag{20}$$

Since the state space is continuous, we need to utilize a function approximator to solve a policy for each state. Hence, the optimal policy can be obtained by minimizing the expectation of KL divergence, given by

$$L_\pi(\omega) = D_{KL}(\pi_\omega(\cdot|s) || \pi_{opt}(\cdot|s)), \tag{21}$$

where ω represents the parameter of the actor network. $D_{KL}(p||q)$ is the KL divergence, which is used to measure the difference between distribution p and distribution q.

Therefore, considering both formula (20) and formula (21), the optimal policy can be rewritten as

$$L_\pi(\omega) = \alpha log\pi_\omega(g_\omega(\tau; s_k)|s_k) + V^\pi(s_k) - Q^\pi(s_k, g_\omega(\tau; s_k)), \tag{22}$$

where action a_k is replaced by $(g_\omega(\tau; s_k)$, and τ is an action noise vector that sampled from the standard normal distribution.

Correspondingly, the gradient of $L_\pi(\omega)$ is expressed as

$$\nabla_\omega L_\pi(\omega) = \nabla_\omega \alpha log\pi_\omega(a_k|s_k) + \nabla_\omega g_\omega(\tau; s_k) (\nabla_a \alpha log\pi_\omega(a_k|s_k) - \nabla_a Q_\theta(s_k, a_k)). \tag{23}$$

The details of SAC-based algorithm can be summarized by Algorithm 1.

IV. Numerical Results

In this section,we provide a comprehensive performance evaluation on the proposed algorithm. Without loss of generality, the main assumptions utilized in our simulations are summarized as follows: B = 20 MHz , $\sigma_i^2 = \sigma_j^2 = 10^{(-10)}$ mW, $P_i = P_j$ = 600 mW, and $h_i = h_j$ = 128.1 + 37.6log(d). The CPU frequency of RSUs and vehicles is assumed to be uniformly distributed, i.e. $Z^{\text{RSU}} \sim$ Unif([6, 7] GHz) and $Z^{\text{Vehicle}} \sim$ Unif([1, 2] GHz). Then, the basic parameter settings of our system and algorithm are shown in Table I.

To evaluate the proposed SAC-based algorithm, it is compared with DDPG algorithm as well as other three benchmark schemes as follows: Offloading Decision (OD) scheme only considers the computation offloading optimization without considering the optimization of resource allocation; All Local (AL) scheme considers the computation tasks of all TVs to be performed locally; All RSU (AR) scheme considers that all TVs offload their computation tasks to associated RSU for processing remotely.

In Fig. 2, we compare the convergence of the proposed SAC-based algorithm and DDPG algorithm, which is characterized by the cumulative average reward. It can be seen that the proposed algorithm has a faster convergence speed and a smaller oscillation amplitude than the DDPG algorithm when the number of training rounds increases. It is due to the fact that the SAC-based algorithm utilizes the concept of entropy, resulting in better exploration capabilities, and is not sensitive to super parameters.

TABLE I: Parameters

Parameters	Value
Number of BS	1
Number of RSUs	2
Number of IVs	2
Number of TVs	2∼6
RSU coverage radius	150 m
Data size of computation tasks	[2, 8] Mbits
CPU cycles of computation tasks	[1, 5] GHz
Maximum completion deadline	1.5 s
Capacity of replay memory	10000
Mini-batch sample	128
Learning rate	0.001
Discount factor for reward	0.99

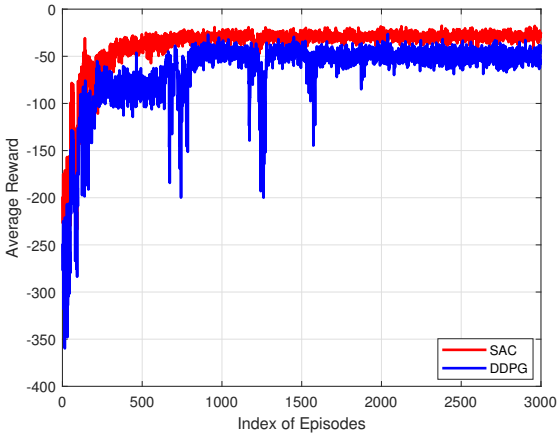

Fig. 2: The convergence of the proposed SAC-based algorithm and DDPG algorithm.

Fig. 3 shows the total latency for different numbers of TVs. By varying the number of TV from 2 to 6, it shows that the proposed algorithm has the lowest total latency. Moreover, the slope of the curve of AR scheme becomes steeper as the number of TV increases. That is because the resource budget of the associated RSU assigned to each time slot becomes relatively tight when more tasks need to perform, and therefore the time to complete a single task increases.

Fig. 4 shows the total latency for different computing resources of the associated RSU. As shown in the figure, by relaxing F_r from 5 GHz to 9 GHz, it shows a significant decrease in the total latency. That is because the more powerful the computing resources of the associated RSU, the less time it takes to complete the task. Meanwhile, it can be seen that the total latency of the SAC-based algorithm is the lowest. Moreover, it shows that the proposed SAC-based algorithm can reduce 23% total latency compared to the DDPG algorithm when the computing resources of the associated RSU is small, such as 5GHz.

To further evaluate the effectiveness of the proposed algorithm, Fig. 5 shows the total latency versus the computing resources required by the tasks of TVs under varying the CPU cycles from 2000 to 4000. It can be seen that the total latency of all methods gradually increases with the increase of the required computing resources. It is due to the fact that the computing resources allocated to the task by the associated RSU become less when the number of TVs increases, and coupled with the transmission latency of the task, the effect is not as good as being performed locally. Moreover, the proposed algorithm has the lowest latency due to the fact that it takes full advantag of edge nodes' resources and thus reduces the total latency of the system.

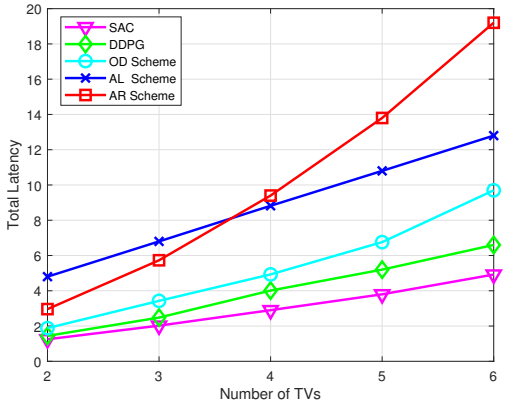

Fig. 3: Total latency with different numbers of TVs.

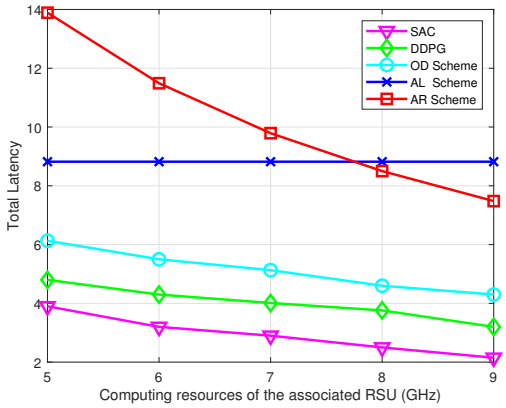

Fig. 4: Total latency with different computing resources of the associated RSU.

V. Conclusion

In this paper, we have considered a VEC system architecture composed of mobile vehicles and road infrastructures. Then, we have formulated the joint optimization problem of computation offloading and resource allocation. Moreover, a SAC-based algorithm has been proposed to solve the problem. The simulation results show that the proposed algorithm can effectively reduce the total latency of the system, and its performance is significantly better than other methods.

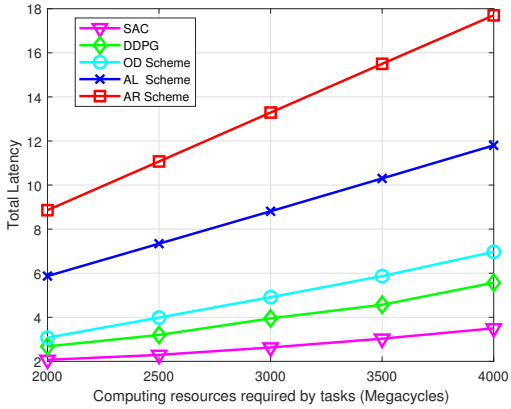

Fig. 5: Total latency with different CPU cycles.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants NO. 62172255 and NO. 61872221, and the Suzhou Municipal Key Industrial Technology Innovation Program (SYG202123). The corresponding author is R. Chen.

References

[1] B. Brik, P. A. Frangoudis, and A. Ksentini, "Service-Oriented MEC Applications Placement in a Federated Edge Cloud Architecture," *in Proc. IEEE ICC*, 2020, pp. 1-6.

[2] H. Zhou, T. Wu, H. Zhang, and J. Wu, "Incentive-driven Deep Reinforcement Learning for Content Caching and D2D Offloading," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2445-2460, 2021.

[3] K. Jiang, H. Zhou, D. Zeng, and J. Wu, "Multi-Agent Reinforcement Learning for Cooperative Edge Caching in Internet of Vehicles," *in Proc. IEEE MASS*, 2020, pp. 1-9.

[4] J. Shi, J. Du, J. Wang, and J. Yuan, "Distributed V2V Computation Offloading Based on Dynamic Pricing Using Deep Reinforcement Learning," *in Proc. IEEE WCNC*, 2020, pp. 1-6.

[5] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep Reinforcement Learning for Energy Efficient Computation Offloading in Mobile Edge Computing," *IEEE Internet Things J.*, vol. PP, no. 99, pp. 1-1, 2021.

[6] H. Ke, J. Wang, L. Deng, Y. Ge, and H. Wang, "Deep Reinforcement Learning-Based Adaptive Computation Offloading for MEC in Heterogeneous Vehicular Networks," *IEEE Trans. Veh. Technol.*, vol. 69, no. 7, pp. 7916-7929, 2020.

[7] J. Wang, D. Feng, S. Zhang, J. Tang, and T. Q. S. Quek, "Computation Offloading for Mobile Edge Computing Enabled Vehicular Networks," *IEEE Access*, vol. 7, pp. 62624-62632, 2019.

[8] Y. Wang, P. Lang, D. Tian, J. Zhou, X. Duan, Y. Cao, and D. Zhao, "A game-based computation offloading method in vehicular multiaccess edge computing networks," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4987–4996, Jun. 2020.

[9] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation Offloading and Resource Allocation For Cloud Assisted Mobile Edge Computing in Vehicular Networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944-7956, 2019.

[10] W. Zhan, C. Luo, J. Wang, C. Wang, G. Min, H. Duan, and Q. Zhu, "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5449–5465, 2020.

[11] K. Zhang, Y . Zhu, S. Leng, Y . He, S. Maharjan, and Y . Zhang, "Deep learning empowered task offloading for mobile edge computing in urban informatics," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7635–7647, 2019.

[12] H. Liao, Y. Mu, Z. Zhou, M. Sun, Zhao Wang, and Chao Pan, "Blockchain and Learning-Based Secure and Intelligent Task Offloading for Vehicular Fog Computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7,pp. 4051-4063, 2021.

[13] A. Lakhan, M. Ahmad, M. Bilal, A. Jolfaei, and R. M. Mehmood, "Mobility Aware Blockchain Enabled Offloading and Scheduling in Vehicular Fog Cloud Computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no.7, pp. 4212-4223, 2021.

[14] J. Wang, T. Lv, P. Huang, and P. Mathiopoulos, "Mobility-aware partial computation offloading in vehicular networks: A deep reinforcement learning based scheme," *China Communications*, vol. 17, no. 10, pp. 31-49, 2020.

[15] J. Shi, J. Du, J. Wang, and J. Yuan, "Deep Reinforcement Learning-Based V2V Partial Computation Offloading in Vehicular Fog Computing," *in Proc. IEEE WCNC*, 2021, pp. 1-6.

[16] Y. Hou, C. Wang, M. Zhu, X. Xu, X. Tao, and X. Wu, "Joint allocation of wireless resource and computing capability in MEC-enabled vehicular network," *China Communications*, vol. 18, no. 6, pp. 64-76, 2021.