Deep Reinforcement Learning-Based Offloading Decision Optimization in Mobile Edge Computing

Hao Zhang*, Wenjun Wu*, Chaoyi Wang*, Meng Li*, and Ruizhe Yang*
*Faculty of Information Technology, Beijing University of Technology, Beijing, P.R. China
Email: edgenovo@emails.bjut.edu.cn, wenjunwu@bjut.edu.cn, wangchaoyi1996@gmail.com,
limeng720@bjut.edu.cn, yangruizhe@bjut.edu.cn

***Abstract*—As a promising technique, mobile edge computing (MEC) has attracted significant attention from both academia and industry. However, the offloading decision for computing tasks in MEC is usually complicated and intractable. In this paper, we propose a novel framework for offloading decision in MEC based on Deep Reinforcement Learning (DRL). We consider a typical network architecture with one MEC server and one mobile user, in which the tasks of the device arrive as a flow in time. We model the offloading decision process of the task flow as a Markov Decision Process (MDP). The optimization object is minimizing the weighted sum of offloading latency and power consumption, which is decomposed into the reward of each time slot. The elements of DRL such as policy, reward and value are defined according to the proposed optimization problem. Simulation results reveal that the proposed method could significantly reduce the energy consumption and latency compared to the existing schemes.**

***Index Terms*—Mobile edge computing, deep reinforcement learning, Markov decision process, wireless networks.**

I. Introduction

With the rapid growth of wireless networks, mobile communications devices, such as smartphones, tablets, wireless surveillance cameras, as well as driverless vehicles, are playing an important role in our daily life [1], [2]. Meanwhile, data computing becomes more and more important in most mobile applications nowadays. Thus, the computing abilities of mobile devices face great challenges [3]. Mobile cloud computing (MCC) has been proposed to enhance computing ability. However, it also has to face various issues such as transmission delay and extra energy consumption [4], [5].

Fortunately, as a promising technology, Mobile Edge Computing (MEC) has been proposed to deploy computing resources closer to mobile devices [6]. Although the performance of CPU in MEC server may be lower than that in the cloud computing server, the total latency can still be significantly reduced since the transmission delay between users and local MEC server is small [7], [8].

In the research of MEC, computation offloading decision is an important branch, which arouses a lot of attention in recent years. The author in [9] provide a detailed survey for computation offloading with mobile edge computing and propose the research on computation offloading to three key areas: decision on computation offloading, allocation of computing resource within the MEC, as well as mobility management. In [10], the authors propose a concept of the radiated energy and consider a general framework that allows joint allocation of radio and computational resources resulting in an optimized trade-off between energy consumption and latency. Also, the authors in [11] propose a scheme that uses a Markov Decision Process (MDP) approach to handling the problem of computation task scheduling policies for MEC systems.

However, the computation complexity of these existing offloading decision schemes is extremely high. Deep Reinforcement Learning (DRL) is a promising method that provides a way to train a decision model automatically. It is a combination of traditional reinforcement learning with the concept of deep neural network. One important branch of DRL is the action-value estimates based methods which learn the values of actions and then select actions based on their estimated values. The most common method in this branch is Q-learning. This method is widely used in many research areas related to wireless communications [12]–[14].

Nevertheless, when the action-value mapping is not a one-to-one mapping, the action selections based on estimated values may not be accurate. Besides, when the decision process is stochastic, action-value methods have no natural way of finding stochastic optimal policies, whereas policy approximating methods can [15], [16]. With the policy approximating method, the policy is represented by a deep neural network. Using the policy network, we can directly map from perceived states of the environment to the probabilities of actions to be taken.

In this paper, with recent advances in DRL, we propose a novel framework for offloading decision optimization in MEC with the policy gradient method. Comparing with the existing schemes, the proposed approach focus on performance optimization of offloading latency and power consumption. The optimization problem is modeled as an MDP firstly, then the states, actions and the total cost function can be defined. After that, the total cost function is decomposed into the reward of each time slot, which is used for policy network training. The episodic simulation for the offloading decision process is constructed, and the policy network is trained and tested via

This work is supported by the National Natural Science Foundation of China under Grants No. U1633115, the Science and Technology Foundation of Beijing Municipal Commission of Education under Grant No. KM201810005027, and the Project funded by China Postdoctoral Science Foundation under Grants No. 2018M640032. *(Corresponding author: Meng Li).*

978-1-5386-7646-2/19/$31.00 ©2019 IEEE
Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on May 04,2021 at 12:21:14 UTC from IEEE Xplore. Restrictions apply.

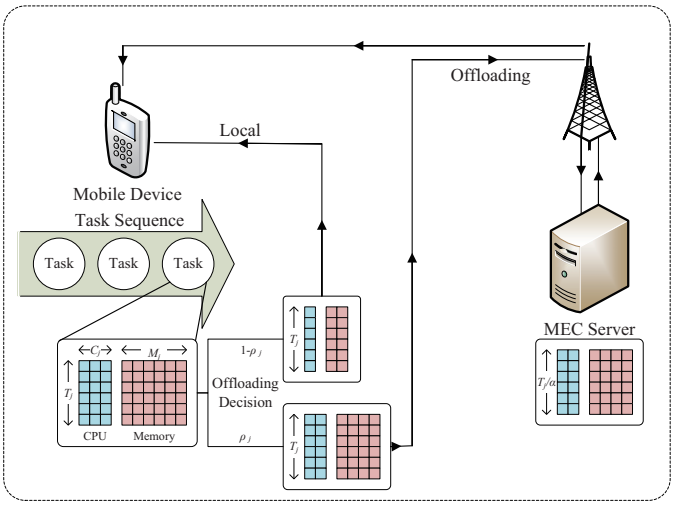


Fig. 1: System model of computing task offloading with mobile edge computing.

extensive episodes. Simulation results confirm that the policy gradient method is working, and the cost of latency and energy is significantly reduced. Moreover, although the training process in DRL takes a long time, the computation complexity in real time offloading decision is low.

The rest of this article is organized as follows. The system model is presented in Section II. In Section III, we formulate an optimization problem for the computation offloading problem and transform it into an MDP. Then the problem solution via deep reinforcement learning is proposed in Section IV. Section V discusses the simulation results. Finally, we conclude this work in Section VI with future works.

II. System Model

In the proposed framework, our model contains one MEC server and one active mobile device, and this device has N different divisible tasks. The system model is described in Fig. 1.

Two kinds of limited resources in the mobile device are considered, that are C CPU resource units and M memory resource units. For the jth task, the amount of data is denoted by M_j and its computation time length on the mobile device is T_j. At each time slot, the jth task requires C_j CPU resource units and M_j memory resource units, where $C_j \leq C$ and $M_j \leq M$.

We assume that the MEC server has superior computing ability comparing with a mobile device, and the CPU and memory resource units are always sufficient for offloaded tasks. Thus, tasks will be processed immediately after arriving at the MEC server. Also, assume the computation speed of the MEC server is α times that of the mobile devices, so the computing latency can be reduced. For the jth task, the processing time on MEC server is $T_j^{\mathrm{s}} = T_j/\alpha$ without waiting.

A. Latency Calculation Model

When a task comes at time t, it will join in the job slots and wait for the offloading decision, the decision waiting time is denoted by τ_j^{wd}, and the offloading ratio is denoted by ρ_j. When the offloading decision is making, the ρ_j parts of the task will be transmitted to the MEC server immediately. The offloaded part of the task is $\rho_j M_j$, which requires $\rho_j C_j$ CPU resource units and $\rho_j M_j$ memory resource units on the MEC server. Its processing time on MEC server is $T_j^{\mathrm{s}} = T_j/\alpha$ according to the system assumptions. The local part of the task is $(1-\rho_j)\, M_j$, which requires $(1-\rho_j)\, C_j$ CPU resource units and $(1-\rho_j)\, M_j$ memory resource units on the mobile device. The processing time on the mobile device is still T_j. However, it may be held up until the CPU and memory of the mobile device has enough idle resources, and the holding up time is denoted by τ_j^{wl}. Thus, the total local serving time is

$$\tau_j^{\mathrm{l}} = T_j + \tau_j^{\mathrm{wl}}. \tag{1}$$

For the offloaded part of the task, the uplink transmission latency τ_j^{t} must be taken into account. We consider an approximate slow fading channel model, whose capacity varies around the average rate H as a Gauss random variable $H_\zeta \sim N\left(H, \sigma^2\right)$ and remains unchanged during the transmission of the jth task. We also introduce a random tail latency τ_j^{tl} which is used to describe the combination time for the preparation and the ending of data transmission [10]. Thus, the offloading transmission latency can be calculated as

$$\tau_j^{\mathrm{t}} = \frac{\rho_j M_j}{H_\zeta} + \tau_j^{\mathrm{tl}}. \tag{2}$$

To simplify the model, the downlink transmission latency between MEC server and the mobile device is assumed to be the same as the offloading transmission latency. This assumption is acceptable in tasks like augmented reality rendering, which requires a similar amount of data transmission in both the uplink and the downlink transmission. The total offloading serving time is

$$\tau_j^{\mathrm{s}} = 2\tau_j^{\mathrm{t}} + T_j^{\mathrm{s}}. \tag{3}$$

The processing of a task will be complicated when both the local and the MEC processing are finished; thus, the final computation serving latency of the jth task is

$$\tau_j = \max(\tau_j^{\mathrm{l}}, \tau_j^{\mathrm{s}}) + \tau_j^{\mathrm{wd}}. \tag{4}$$

B. Energy Consumption Model

We mainly focus on the energy consumption of the mobile device, which is consist of local computing energy consumption $\varepsilon_j^{\mathrm{c}}$ and local transmission energy consumption $\varepsilon_j^{\mathrm{t}}$. Therefore, it can be written as

$$\varepsilon_j = \varepsilon_j^{\mathrm{c}} + \varepsilon_j^{\mathrm{t}}. \tag{5}$$

For simplicity, the energy consumption of both CPU and memory for a certain task are all modeled in the energy consumption of its occupied CPU units. Assume the energy consumption of each running CPU resource unit at each time

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on May 04,2021 at 12:21:14 UTC from IEEE Xplore. Restrictions apply.

slot is P_c. The local computing energy consumption of the jth task is

$$\varepsilon_j^{\mathrm{c}} = (1-\rho_j) C_j T_j P_c. \tag{6}$$

Generally, the mobile device needs transmission processors to serve the wireless communication. For the convenience of the resource state representation, the transmission processor is modeled together with CPU resources. Assume one additional CPU resource unit is occupied as the transmission processor while a mobile transmission is open. The corresponding local transmission energy consumption at each transmitting time slot is approximately defined as P_{tr}, which includes the energy consumption of the RF, the circuit, the transmission serving CPU, etc. For the mobile device receiving slot, the transmission serving CPU unit is also occupied. Although the RF energy consumption of the receiving slot is usually smaller than that of the transmission slot, the receive algorithms are more complex in most cases, which consumes more energy. Thus, the transmission energy consumption at each receiving time slot is also approximately defined as P_{tr} for simplicity. The local transmission energy consumption of the jth task can be represented as

$$\varepsilon_j^{\mathrm{t}} = 2 P_{tr} \tau_j^{\mathrm{t}} \tag{7}$$

III. Problem Formulation

A. Optimization Problem Formulation

In our study, the optimization variable is the offloading ratio ρ_j for each task j, and constraints are the limitations of CPU and memory resource units of the mobile device. The optimization goal is to minimize the function related to the average task latency and energy consumption. The overall cost function is given by

$$F = \sum_{j=1}^{N} F_j = \sum_{j=1}^{N} (L_j + \lambda E_j), \tag{8}$$

where, N is the number of tasks, F_j is the cost function of the jth task, L_j is the latency cost, E_j is the energy cost and λ is the weight ratio between latency and energy cost.

We use a ratio between the actual latency and the baseline latency to denote the latency cost, which is named as "slow-down". The baseline latency is the computation time length of the job T_j at the mobile device. According to Eq. 4, L_j can be calculated as

$$L_j = \frac{\tau_j}{T_j}. \tag{9}$$

The definition of the energy cost is similar. The baseline of the energy cost is the energy consumption when the jth task is completely executed at the mobile device. Therefore, it can be written as

$$\varepsilon_j^{\mathrm{b}} = P_c C_j T_j. \tag{10}$$

According to Eq. 5, 6 and 7, the energy cost can be calculated as

$$E_j = \frac{\varepsilon_j}{\varepsilon_j^{\mathrm{b}}}. \tag{11}$$

B. Markov Decision Process Formulation

As the arriving flow of tasks is a sequence in time, we must make offloading decisions for these tasks at each time slot. The offloading decision problem can be modeled as a Markov Decision Process (MDP). In the rest of this section, we will describe the definition of states, actions, rewards and values.

1) States: The state of the MDP at each time slot t is defined as the resource units occupation in the mobile device and the resource requirements of tasks in the job slot waiting list.

$S^{\mathrm{rc}}(t)$ and $S^{\mathrm{rm}}(t)$ are $T_h \times C$ and $T_h \times M$ matrices denoting the observed CPU and memory resource units occupation in the next T_h time slots in the mobile device, respectively. We predefined J_m values between 0 and 1. When a task is scheduled, a value randomly selected from the J_m values will be assigned to it, and the resource units occupied by this task will be filled by the randomly selected value.

To reduce the dimensions of the state, the tasks in the waiting list are represented in two levels that are job slots and backlog. $S^{\mathrm{jc}}(t)$ is a $T_h \times C J_s$ matrix consists of J_s blocks, which is given as

$$S^{\mathrm{jc}}(t) = \left[S_1^{\mathrm{jc}}(t), S_2^{\mathrm{jc}}(t), \ldots, S_j^{\mathrm{jc}}(t), \ldots, S_{J_s}^{\mathrm{jc}}(t) \right]. \tag{12}$$

The block $S_j^{\mathrm{jc}}(t)$ denotes the CPU resource requirements of the jth task in the job slots. For the task which requires C_j CPU resource units in T_j time slots, the first C_j columns of the first T_j rows of $S_j^{\mathrm{jc}}(t)$ is set as 1 and the rest of the block is 0. $S^{\mathrm{jm}}(t)$ denotes the memory resource requirements of the tasks in the job slots, and its definition of is similar to that of $S_j^{\mathrm{jc}}(t)$. When the number of waiting tasks are larger than J_s, the rest of the tasks will be marked in the backlog $S^{\mathrm{bl}}(t)$ without details, which is a $T_h \times B_l$ matrix. According to the balance between complexity and accuracy, the number of tasks in the job slots J_s can be configured.

Thus, the system state can be written in the following form

$$S(t) = \left[S^{\mathrm{rc}}(t), S^{\mathrm{jc}}(t), S^{\mathrm{rm}}(t), S^{\mathrm{jm}}, (t), S^{\mathrm{bl}}(t) \right]. \tag{13}$$

2) Actions: For each offloading decision at time slot t, the action is given by

$$a(t) \in \{0, 1, 2, \cdots, C\}, \tag{14}$$

where $a(t)$ denotes the offloading number of CPU process units of the task scheduled at time slot t. Obviously, for the task which requires C_j CPU resource units, we must ensure that $a(t) \le C_j$ and $\rho_j = a(t)/C_j \le 1$.

3) Reward: Assume the sequence of the N tasks arrives during a limited observing time T_M, the task arriving probability is N/T_M. The overall cost function F given by Eq. 8 can be modeled as opposite of the value V of the initial state $S(1)$. Therefor, the relationship between the overall cost function and the reward function can be expressed as

$$F = -V(S(1)) = -\sum_{t \in [1, T_M]} \gamma^{t-1} R(t), \tag{15}$$

where $R(t)$ is the reward of time slot t, γ is the reward discount ratio describing how the rewards of the future time slots affects

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on May 04,2021 at 12:21:14 UTC from IEEE Xplore. Restrictions apply.

the overall cost function. In this paper, the discount ratio is defined as $\gamma = 1$, and the overall cost function can be rewritten as the sum of reward at each time slot, which is

$$\begin{aligned} F &= -\sum_{t\in[1,T_M]} R\left(t\right) \\ &= -\sum_{t\in[1,T_M]} \left[\sum_{j\in\mathbf{J}(t)} R_j^{\mathrm{l}}\left(t\right) + \lambda \sum_{i\in\mathbf{I}} R_i^{\mathrm{e}}\left(t\right)\right], \end{aligned} \tag{16}$$

where, $R_j^{\mathrm{l}}\left(t\right)$ is the latency reward of the jth task at time slot t, $\mathbf{J}(t)$ is the set of tasks that are currently active in the system, $R_i^{\mathrm{e}}\left(t\right)$ is the energy reward of the ith CPU unit at time slot t, $\mathbf{I}$ is the set of CPU resource units of the mobile device.

According to the MDP model, the latency reward transformation is simple. Initially, the latency is used to describe the time that the task is keeping in the job slot waiting list and the CPU workload. During the transmission process, a transmission serving CPU unit is occupied, which can be included in the case that the task in the CPU workload. So we can decide that if a task exists either in the CPU resource units or in the job slots, the latency reward will decrease by $1/T_j$. Hence, it can be written as

$$R_j^{\mathrm{l}}\left(t\right) = \frac{-1}{T_j}. \tag{17}$$

The energy reward is decomposed from another dimension. More specifically, it is mentioned in Eq. 16 that the energy reward of each time step is the sum of the energy reward of every CPU resource units. Then, For the ith CPU resource unite occupied by the jth task, $R_i^{\mathrm{e}}\left(t\right)$ can be calculated as

$$R_i^{\mathrm{e}}\left(t\right) = \frac{-P_i\left(t\right)}{\varepsilon_j^{\mathrm{b}}}, \tag{18}$$

where $P_i\left(t\right)$ denotes the energy consumption of the ith CPU unit at time slot t.

According to our system model, there exist two kinds of CPU units occupation, that are computing occupation and transmitting occupation. To identify the different occupation, we introduce an indicator variable g_i for each CPU resource unit. If $g_i = 0$, this CPU resource unit is idle. If $g_i = 1$, this CPU resource unit is used for computation. Otherwise, if $g_i = 2$, this CPU resource unit is used as the transmission processor. The power consumption in different conditions can be calculated as

$$P_i\left(t\right) = \begin{cases} 0 & \text{if } g_i = 0 \\ P_c & \text{if } g_i = 1 \\ P_{tr} & \text{if } g_i = 2 \end{cases} \tag{19}$$

Notice that all the energy consumed in the transmission is included in P_{tr} when the CPU status indicator $g_i = 2$.

C. Complexity Analysis of the MDP

According to the definition of states in Eq. 13, the number of states n_s of the proposed MDP can be calculated as

$$n_s = J_m^{T_h(C+M)} 2^{T_h J_s (C+M) + T_h B_l}. \tag{20}$$

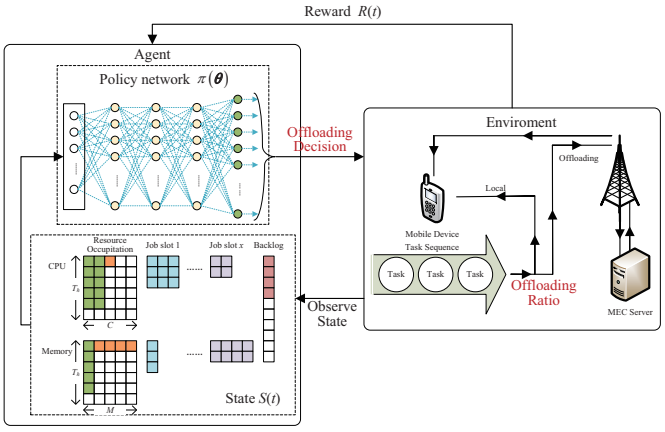


Fig. 2: Framework of deep reinforcement learning based task offloading.

The global optimal solution can be obtained by Dynamic Programming (DP) such as value iteration and policy iteration. These approaches have the polynomial complexity of $O\left(n_s^2\right)$ [14] [17]. Unfortunately, the state of the MDP contains too much information and the state space is extremely large. As a result, it is difficult to get the complete probability distributions of all possible state transitions. Thus, using DP to solve the proposed MDP is inconvenient.

In this paper, we introduce in the DRL method to solve the MDP. The learning method requires only experience, which are sample sequences of states, actions and rewards from actual or simulated interaction with an environment. Although a model is required, the model need only generate sample transitions, not the complete probability distributions of all possible state transitions that are required for DP [15].

IV. Deep Reinforcement Learning based Problem Solution

In this section, we describe the proposed Deep Reinforcement Learning based Tasks Offloading (DRL-TO) method in detail, and the framework is given in Fig. 2.

The state of time step t defined in Eq. 13 is represented as a picture of the global state view. The environment state transition is obtained by episodic simulations. The policy, which is a mapping from perceived states of the environment to the probabilities of actions to be taken, is represented by a deep neural network. $\pi(\theta)$ denotes the policy network with parameter θ. The reward function $R\left(t\right)$ is described in the previous section and the value of time slot t can be calculated by $V\left(t\right) = \sum_{k\in[t,T_M]} R\left(k\right)$. We train the policy network to achieve the optimization goal via the reward signal of each time slot generated from the episodic simulation.

A simplified process of one time slot simulation of one episodic simulation is given in Alg. 1. To improve the efficiency of the proposed scheme, we allow the program to make multiple decisions in a single time slot to schedule more than one task until there are not enough resources. This is implemented as Step 4.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on May 04,2021 at 12:21:14 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 One time slot simulation process

1: Get the picture of the global state view at the current time.
2: Decide whether to schedule a task
a) If there is no task in the job slots, go to Step 5.
b) Otherwise, make a decision use the policy network. If there are enough resources for the task, go to Step 3. Otherwise, go to Step 5.
3: Allocation
a) Calculate the actual processing time of the scheduled task as $\max(\tau_j^{\mathrm{l}}, \tau_j^{\mathrm{s}})$ and calculate the end time of it by adding the actual processing time to current time.
b) Update the resource units occupation state and the global state view.
4: Decide whether to move on
a) If there is no task in the job slots, go to Step 5.
b) Otherwise, go back to Step 2.
5: Reward computation
a) Migrate new coming tasks into the job slots and running task list and update the global state view.
b) Initialize the reward value $R(t) = 0$.
b) For each task in the running task list: if current time exceed the task's end time, remove this task from the list; add all remaining tasks' latency reward $\sum_{j\in J} R_j^{\mathrm{l}}(t)$ to $R(t)$.
c) For all the CPU resource units, add the energy reward $\lambda \sum_{i\in I} R_i^{\mathrm{e}}(t)$ to $R(t)$.
6: Move on
a) Move forward the time by adding the current time by 1.
b) Output the new global state view and the reward value.
7: Begin the next time slot, until all tasks are accomplished.

To train the policy network, N_s task sequences are generated randomly. For each training iteration, we run N_e episodic simulations for each task sequence to get enough trajectories. For the kth episodic simulation of the ith task sequence, the trajectory is defined as $[S_{ik}(t), a_{ik}(t), R_{ik}(t)]_{t\in[1,T_M]}$. We can obtain

$$\Delta\theta_{ikt} = \nabla_\theta \log p\left[\pi(\theta), S_{ik}(t), a_{ik}(t)\right]\left[V_{ik}(t) - b_i(t)\right], \quad (21)$$

where $p\left[\pi(\theta), S_{ik}(t), a_{ik}(t)\right]$ is the probability of action $a_{ik}(t)$ at state $S_{ik}(t)$ when the policy network is $\pi(\theta)$, $b_i(t) = (1/N_e)\sum_{k\in[1,N_e]} V_{ik}(t)$ is a baseline value of the ith task sequence. Therefore, in each training iteration, the policy network can be updated by

$$\theta \leftarrow \theta + r_l \sum_{i=1}^{N_s}\sum_{k=1}^{N_e}\sum_{t=1}^{T_M} \Delta\theta_{ikt}, \quad (22)$$

where r_l is the learning rate. More detailed information of the DRL training process and the policy gradient method we used can be found in [16].

V. Numerical Results and Discussions

In this section, we built a policy network that includes a simple neural network with two fully connected hidden layers, each of them contains 20 neurons. To train the policy network, we generated 50 task sequences. Each of them lasts for 250 time slots with task arriving probability 0.6 at each time slot. T_j of 80% of the tasks are uniformly chosen between 1 and 3 time slots and the remaining are chosen uniformly from 10 to 15 slots. To define the resource demands C_j and M_j, assume each task has a dominant resource requirement which is picked independently at random. The demand for the dominant resource is chosen uniformly between 5 and 10 and the demand of the other resource is chosen uniformly between 1 and 2. The resource limitation of the mobile device is $C = 10$ and $M = 10$. The computation efficiency ratio between MEC server and the mobile device is $\alpha = 2$. Without loss of generality, the weight parameter in cost function is chosen as $\lambda = 1$. We ran 1000 training iterations, and in each iteration, 10 episodes were simulated for each task sequence. The $rmsprop$ algorithm with a learning rate of 0.001 was used to update the policy network parameters [16]. To test the performance of the trained policy network, task sequence groups with task arriving probability from 0.3 to 0.9 are generated independently. Each group contains 500 task sequences with the same task arriving probability.

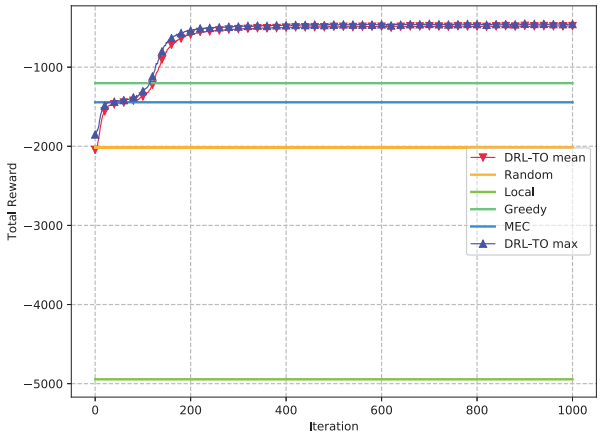
The proposed DRL-TO method is compared with four different schemes: the non-offloading scheme that process all tasks in the mobile device denoted by "Local", a complete-offloading scheme that process all tasks in MEC server denoted by "MEC", a random scheme that choose offloading ratio randomly denoted by "Random", and a greedy method that only offload when the task load exceed the mobile device's computation capacity.

In Fig. 3, we focus on the average values of three aspects of the performance: the total reward $\sum_{t\in[1,T_M]} R(t)$, the latency reward $\sum_{t\in[1,T_M]}\sum_{j\in\mathbf{J}(t)} R_j^{\mathrm{l}}(t)$ and the energy reward $\sum_{t\in[1,T_M]}\sum_{i\in\mathbf{I}} R_i^{\mathrm{e}}(t)$. We can observe that in all the comparing algorithms, "Local" performs the worst. While the "Greedy" has the best results in these comparing schemes since it can adjust the amount of offloading according to the remaining resource status.

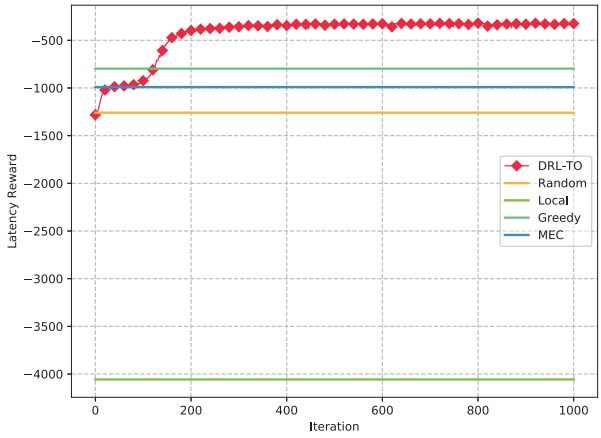
From all these figures, we can observe that the improvements over iterations are significant and DRL-TO out-performs other schemes with less than 200 iterations of training. All the results show that the deployment of MEC offloading can deliver a considerable improvement over the "Local" scheme. Fig. 3 (a) shows that the average total reward can improve about 4200 and 600 compared with "Local" and "Greedy", respectively. It means the cost function defined in Eq. 8 is reduced by about 46% at least and 85% at most. Fig. 3 (b) indicates that the latency reward can be improved up to 50% when compared with "Greedy" scheme, and Fig. 3 (c) shows that the energy reward can be reduced by nearly 60% compared with "Greedy" scheme.

Fig. 4 demonstrates the performance test of DRL-TO scheme using task sequence groups with task arriving probability from 0.3 to 0.9. Results show that DRL-TO scheme outperforms other schemes. The reduction in average cost value is between

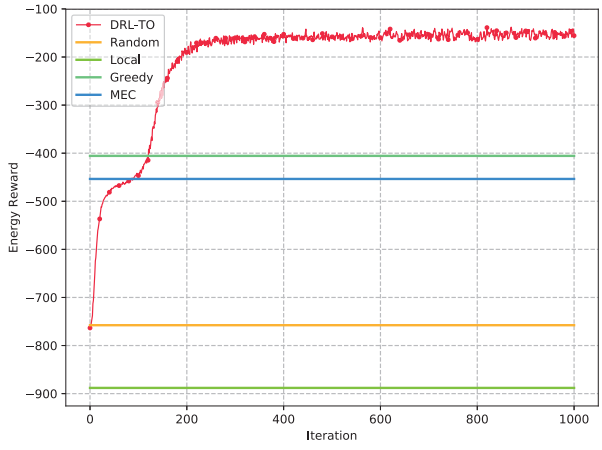
Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on May 04,2021 at 12:21:14 UTC from IEEE Xplore. Restrictions apply.



(a)



(b)



(c)

Fig. 3: Training performance of the total reward, the latency reward, and the energy reward.

58% to 68% in comparison with "Greedy", which is the second best scheme. Since the computing capability of the MEC server is strong enough, the average cost value of the "MEC" scheme changes slightly with the growing of task arriving probability. It is also worth noting that although the policy network is

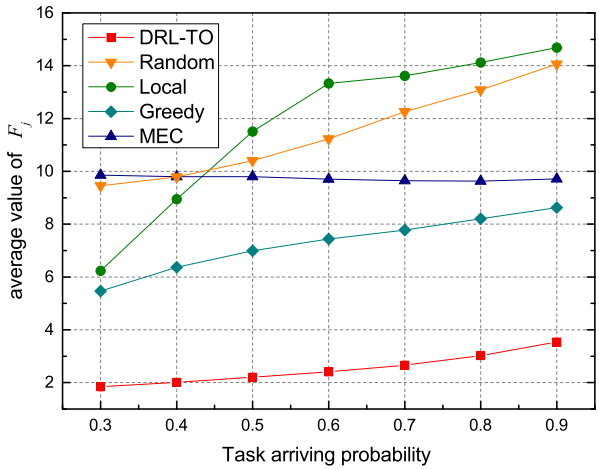


Fig. 4: Average value of F_j vs. task arriving probability.

trained by 50 task sequences with task arriving probability of 0.6, it performs well through different scenarios with different task arriving probabilities. This is because the policy network makes decisions according to the state and we have simulated $N_s N_e T_M = 125000$ time slots to observe enough state in each training iteration.

In addition to high performance, the computational complexity is also critical. In DRL-To scheme, the neural network utilizes matrix multiplication for training and get action probability. Therefore, the computational complexity of a neural network is related to its structural parameters. Since the number of neurons in the hidden layer and the output layer is small in the proposed DRL-TO framework, the computational complexity mainly depends on the number of neurons in the input layer n_i. In this paper, n_i is equal to the number of elements in each state, which can be calculated as $n_i = T_h\,(C+M)\,(1+J_s) + T_h B_l$. Thus, the computational complexity of each decision using DRL-To scheme is similar to $O(n_i)$. For the "Greedy" scheme, which is the second best scheme, it needs to compare the resource requirement of the task with the resource state and then make a decision. The computational complexity of each decision using "Greedy" scheme is $O(n_g)$, where $n_g = T_h\,(C+M)$. Since the value of J_s and B_l is less than 10 in this paper, it is clear that the DRL-To scheme is comparable in computational complexity with "Greedy" scheme. In a word, the high performance and the comparable computational complexity confirm the effectiveness of the proposed DRL-To scheme.

VI. Conclusions and Future Work

In this paper, we proposed a novel framework of task offloading decision for MEC using DRL. A typical network architecture with one MEC server and one mobile user was taken into account in the proposed framework. The optimization object was designed as minimizing the weighted sum of offloading latency cost and power consumption cost. The policy gradient method is adopted to solve the problem. Simulation results

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on May 04,2021 at 12:21:14 UTC from IEEE Xplore. Restrictions apply.

show that the proposed DRL-TO scheme can significantly reduce the cost we defined. The test simulation also confirms that the DRL-TO scheme can adjust to different scenarios with different task arriving probability and performs well. To focus on the validation of the effectiveness of the policy gradient method in making MEC traffic offloading decision, the system model and problem formulation of this paper are simplified. In the future, we will consider multiple mobile devices in the system and model the wireless transmission resources in the global state view more accurately.

REFERENCES

[1] D. T. Wiriaatmadja and K. W. Choi, "Hybrid random access and data transmission protocol for machine-to-machine communications in cellular networks," *IEEE Trans. Wireless Commun.*, vol. 14, no. 1, pp. 33–46, Jan. 2015.

[2] M. Li, F. R. Yu, P. Si, E. Sun, Y. Zhang, and H. Yao, "Random access and virtual resource allocation in software-defined cellular networks with machine-to-machine (M2M) communications," *IEEE Trans. Veh. Tech.*, vol. 66, no. 7, pp. 6399–6414, Dec. 2016.

[3] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, Jun. 2016.

[4] C. Wang, Y. He, F. R. Yu, Q. Chen, and L. Tang, "Integration of networking, caching and computing in wireless systems: A survey, some research issues and challenges," *IEEE Commun. Surveys Tutorials*, vol. 20, no. 1, pp. 7–38, First quarter 2018.

[5] M. Li, F. R. Yu, P. Si, H. Yao, E. Sun, and Y. Zhang, "Energy-efficient M2M communications with mobile edge computing in virtualized cellular networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*. Paris, France, May 2017, pp. 1–6.

[6] ETSI, "Mobile-edge computing: Introductory technical white paper," *ETSI White Paper*, Sep. 2014.

[7] Y. He, F. R. Yu, N. Zhao, V.C.M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Comm. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[8] M. Li, F. R. Yu, P. Si, and Y. Zhang, "Green machine-to-machine communication with mobile edge computing and wireless network virtualization," *IEEE Comm. Mag.*, vol. 56, no. 5, pp. 148–154, May 2018.

[9] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading view document," *IEEE Commun. Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, Third quarter 2017.

[10] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Joint allocation of radio and computational resources in wireless application offloading," in *2013 Future Network and Mobile Summit*. Lisboa, Portugal, Jul. 2013, pp. 1–10.

[11] J. Liu, Y. Mao, J. Zhang, and K.B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE International Symposium on Information Theory (ISIT)*. Barcelona, Spain, Jul. 2016, pp. 1–6.

[12] Y. He, Z. Zhang, F. R. Yu, N. Zhao, H. Yin, V.C.M. Leung, and Y. Zhang, "Deep-reinforcement-learning-based optimization for cache-enabled opportunistic interference alignment wireless networks," *IEEE Trans. Veh. Tech.*, vol. 66, no. 11, pp. 10 433–10 445, Nov. 2017.

[13] J. Zhu, Y. Song, D. Jiang, and H. Song, "A new deep-q-learning-based transmission scheduling mechanism for the cognitive Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2375–2385, Aug. 2018.

[14] H. Liu, S. Liu, and K. Zheng, "A reinforcement learning-based resource allocation scheme for cloud robotics," *IEEE Access*, vol. 6, pp. 17 215–17 222, 2018.

[15] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2017.

[16] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM Workshop on Hot Topics in Networks*. Atlanta, GA, Nov. 2016, pp. 50–56.

[17] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2005.

Authorized licensed use limited to: HEFEI UNIVERSITY OF TECHNOLOGY. Downloaded on May 04,2021 at 12:21:14 UTC from IEEE Xplore. Restrictions apply.