Deep Reinforcement Learning based Computation Offloading and Resource Allocation for MEC

Ji Li, Hui Gao, Tiejun Lv, and Yueming Lu
Key Laboratory of Trustworthy Distributed Computing and Service, Ministry of Education
School of Information and Communication Engineering
Beijing University of Posts and Telecommunications, Beijing, China 100876
{lj386588424, huigao, lvtiejun, ymlu}@bupt.edu.cn.

***Abstract*—Mobile edge computing (MEC) has the potential to enable computation-intensive applications in 5G networks. MEC can extend the computational capacity at the edge of wireless networks by migrating the computation-intensive tasks to the MEC server. In this paper, we consider a multi-user MEC system, where multiple user equipments (UEs) can perform computation offloading via wireless channels to an MEC server. We formulate the sum cost of delay and energy consumptions for all UEs as our optimization objective. In order to minimize the sum cost of the considered MEC system, we jointly optimize the offloading decision and computational resource allocation. However, it is challenging to obtain an optimal policy in such a dynamic system. Besides immediate reward, Reinforcement Learning (RL) also takes a long-term goal into consideration, which is very important to a time-variant dynamic systems, such as our considered multi-user wireless MEC system. To this end, we propose RL-based optimization framework to tackle the resource allocation in wireless MEC. Specifically, the Q-learning based and Deep Reinforcement Learning (DRL) based schemes are proposed, respectively. Simulation results show that the proposed scheme achieves significant reduction on the sum cost compared to other baselines.**

I. Introduction

5G networks are now emerging to support massive connections among humans, machines and things with diverse services. The Cloud Radio Access Network (C-RAN) is considered to be a core technology to enable these services in 5G networks with unprecedented spectral efficiency and energy efficiency [1]. Meanwhile, more and more computation-intensive applications in 5G require low delay, such as interactive game, augmented reality (AR), virtual reality (VR), etc, which are also energy-consuming applications. Noting the limited battery power and computing capacity, it is difficult for mobile user equipment (UE) to meet these requirements. Aiming to tackle the contradiction between UEs and applications, mobile edge computing (MEC) has been recently proposed as a viable solution.

MEC is a new paradigm in C-RAN, which can enhance the computation capacity at the edge of mobile networks by deploying high-performance servers [2]. The MEC servers are densely distributed close to mobile users, and the user devices can offload computing tasks to the MEC servers through wireless channels. Through computation offloading, mobile users can observably reduce the experienced delay of applications and improve the Quality of Service (QoS). Therefore, the computation offloading and computational resource allocation themes have attracted great interests as a key point in the MEC system [3].

In recent years, researchers have proposed some themes on MEC computation offloading and resource allocation for different design objectives [4]. Part of the investigations can be dated back to those for conventional mobile cloud computing (MCC) systems. The work in [5] proposed general guidelines to make offloading decision for energy consumption minimization, where the communication links were easily assumed to have a fixed rate. Since the data rates for wireless communications are not constant, the optimal binary computation offloading decision using convex optimization was proposed in [6], where the the famous Shannon-Hartley formula gave the power-rate function. However, the proposed theme in [6] required accurate channel-state information. Considering the random variations of wireless channels in the time domain, these themes are not so suitable for the dynamic systems.

To tackle this problem, Reinforcement Learning (RL) [7] has been used as an effective solution. Taking the future reward feedback from the environment into consideration, the RL agent can adjust its policy to achieve a best long-term goal, which is very important in time-variant systems, such as our considered multi-user wireless MEC system.

As a fundamental theory of RL, Markov Decision Process (MDP) theory was mostly used by researchers. MDP can obtain an optimal policy by dynamic programming methods, which requires a fixed state transition probability matrix P. The work in [8] designed the single-user delay-optimal task scheduling policies based on the theory of MDP, which controlled the states of the local processing and transmission units and the task buffer queue length based on the channel state. Furtherly to optimize the computation delay and energy consumption together, the problem of minimizing the long-term average execution cost of a single user was considered in [9] and a semi-MDP framework was proposed to jointly control the local CPU frequency, modulation scheme as well as data rates. However, it is difficult to obtain the actual probability distribution of matrix P. The MDP based themes proposed in these papers were too dependent on environmental information. What's more, MDP was mostly used in the

978-1-5386-1734-2/18/$31.00 ©2018 IEEE

scenario of one single cell with one user for task scheduling. Considering the practical cost and benefit, it would be better if an MEC server can serve more UEs.

To this end, we dedicate to design a RL-based theme for a multi-user MEC computation offloading system. We propose a Q-learning based theme to replace MDP at first. But the number of possible actions may increase rapidly with the increasing number of user for Q-learning [10], then it would be complicated to compute and store the action-value Q. To work out this problem, we utilize Deep Reinforcement Learning (DRL) [7] as an enhanced version of RL. Based on DRL, we propose a Deep Q Network (DQN) which can use a Deep Neural Network (DNN) to estimate the value function of Q-learning algorithm.

In summary, the main contributions of this paper are:

- We propose a Q-learning based theme and a DRL based theme to solve the computation offloading problem. Unlike MDP methods applied with single-user in other papers, we present a novel DRL-based theme in a multi-user system.
- We define the state, reward and specially a binary action for the DRL agent in detail, while the action usually contains one parameter in other papers. Moreover, we add a pre-classification step before learning to limit the action space of the agent.
- We evaluate the performance of the proposed Q-learning and DQN themes compared to other two baselines. Simulation results show the effectiveness of the proposed themes in delay and energy consumption reduction with different system parameters.

The rest of this papers is organized as follows. In section II, we present the system model, including network model, task model and computing model. In section III, we describe the formulation of our optimization problem. In section IV, we propose a Q-learning based theme and a DRL based theme in detail. In sectionV, we show the simulation results. Finally, we conclude this study in sectionVI.

II. System Model

A. Network Model

Fig. 1 shows the network model, we consider a scenario of one small cell. There is one eNodeB (eNB) and N UEs in the cell. An MEC server is deployed with the eNB. The set of UE is denoted as $\mathcal{N} = \{1, 2, \ldots, N\}$. We assume that each UE has a computation-intensive task to be completed. Each UE could offload the task to the MEC server through wireless or execute it locally. The capacity of the MEC server is limited and may be not sufficent for all UEs to offload tasks .

We define W as the bandwidth of wireless channel. There is only one eNB in one small cell, so the interval interference is neglected. It's assumed that if multiple UEs choose to offload the task simultaneously, the wireless bandwidth would be equally allocated to the offloading UEs for uploading data. According to [11], the achievable upload data rate for UE n is

$$r_n = \frac{W}{K} \log \left(1 + \frac{P_n h_n}{\frac{W}{K} N_0} \right) \tag{1}$$

where K is the number of offloading UEs, P_n is the transmission power of UE n for uploading data, h_n is the channel gain of UE n in the wireless channel, N_0 is the variance of complex white Gaussian channel noise.

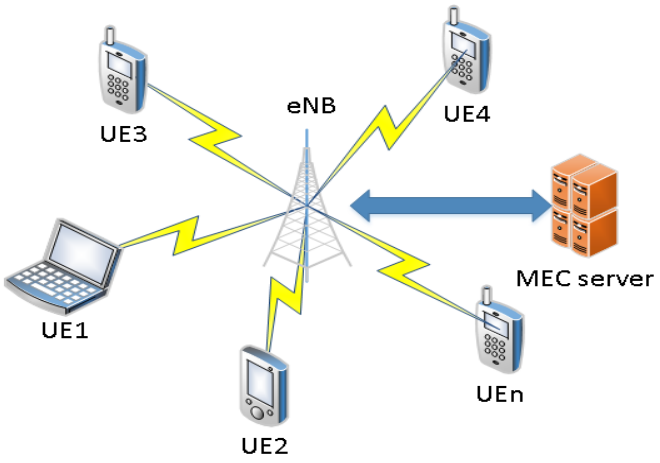

Figure 1. Network model.

B. Task Model

We assume that each UE n has a computation-intensive task $R_n \triangleq (B_n, D_n, \tau_n)$, which can be executed on the local CPU of UE or on the MEC server via computation offloading [12]. Here B_n denotes the size of computation input data needed for computing R_n, including program codes and input parameters. D_n denotes the total number of CPU cycles required to accomplish the computation task R_n and D_n is positively related to the size of B_n. D_n reflects the amount of computing resource required to finish the task R_n. We assume that whether executed by UE n locally or on the MEC server, the size of D_n remains the same. τ_n denotes the maximum tolerable delay of task R_n, that means the experience delay of each user should not exceed τ_n, this will be an important constraint of our optimization problem. All three parameters are related to the feature of the applications and can be estimated through task profiles, so they may be quite different between different kinds of applications.

We assume the task can not be divide into partitions to be processed on different devices, which means that each UE should execute its task by *local computing* or *offloading computing*. We denote $\alpha_n \in \{0, 1\}$ as the computation offloading decision of UE n and define the offloading decision vector as $\mathcal{A} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$. We have $\alpha_n = 0$ if UE n executes its task by *local computing*, otherwise, $\alpha_n = 1$.

C. Computation Model

1) Local Computing Model : If UE n chooses to execute its task R_n locally, we define T_n^l as the local execution delay of

UE n, which only includes the processing delay of local CPUs. Then we denote f_n^l as the computation capacity (i.e., CPU cycles per second) of UE n, it is possible that computational capabilities may be different between different UEs. The local execution delay T_n^l of task R_n is

$$T_n^l = \frac{D_n}{f_n^l} \tag{2}$$

and we have E_n^l as the corresponding energy consumption of task R_n, which is expressed as

$$E_n^l = z_n D_n \tag{3}$$

where z_n represents the energy consumption per CPU cycle to complete task R_n. We set $z_n = 10^{-27}\left(f_n^l\right)^2$ according to the practical measurement in [13].

Combining the time (2) and energy (3) cost, the total cost of *local computing* can be given as

$$C_n^l = I_n^t T_n^l + I_n^e E_n^l$$

where $I_n^t$and I_n^e represent the weights of time and energy cost of task R_n. The weights satisfy $0 \le I_n^t \le 1$, $0 \le I_n^e \le 1$ and $I_n^t + I_n^e = 1$. Like the three notations mentioned in the task model, the weights may be different for each type of task. For sake of simplicity, we assume that the weights of task R_n remain the same during a whole computation offloading process.

2) Offloading Computing Model : If UE n chooses to execute task R_n by offloading computing, the whole offloading approach will be divided into three steps. Firstly, UE n needs to upload sufficient input data (i.e., program codes and parameters) to eNB through wireless access network, and eNB forwards data to the MEC server. Then the MEC server allocates part of computational resource to execute the computing task instead of UE n, finally the MEC server returns the execution results to UE n.

According to the steps above, the required time for the first step of $offloading\ computing$ is the transmission delay,

$$T_{n,t}^o = \frac{B_n}{r_n} \tag{4}$$

where r_n stands for the uplink rate of UE n in the wireless channel mentioned in the network model. And the corresponding energy consumption of the first step is

$$E_{n,t}^o = P_n T_{n,t}^o = \frac{P_n B_n}{r_n} \tag{5}$$

For the second step of $offloading\ computing$, the required time is the processing delay of the MEC server, which can be given as

$$T_{n,p}^o = \frac{D_n}{f_n} \tag{6}$$

where f_n is defined as the allocated computational resource (i.e., CPU cycles per second) by the MEC server to complete task R_n, and F is defined as the entire resource of the MEC server. Then it follows $\sum_{n=1}^{N} \alpha_n f_n \le F$, for the total amount of assigned resource can not exceed the entire computational resource of the MEC server. In this step, we assume that UE n stays idle and define the power consumption of the idle state as P_n^i, the corresponding energy consumption is

$$E_{n,p}^o = P_n^i T_{n,p}^o = \frac{P_n^i D_n}{f_n} \tag{7}$$

For the last step of offloading computing, the required time is the downloading delay of processed result, which is expressed as

$$T_{n,b}^o = \frac{B_b}{r_b} \tag{8}$$

where B_b is the size of processed result, r_b is the download data rate of UE n. But according to [14], [15], the download data rate is very high in general, and the data size of the result is much smaller than that of input data, so the delay and energy consumptions of this step are neglected in the rest of this paper.

According to (4), (5), (6) and (7), the execution delay and energy consumption of UE n through $offloading\ computing$ approach is

$$T_n^o = \frac{B_n}{R_n} + \frac{D_n}{f_n} \tag{9}$$

and

$$E_n^o = \frac{P_n B_n}{R_n} + \frac{P_n^i D_n}{f_n} \tag{10}$$

Combining the time (9) and energy (10) cost, the total cost of $offloading\ computing$ can be given as

$$C_n^o = I_n^t T_n^o + I_n^e E_n^o \tag{11}$$

and the sum cost of all users in the MEC offloading system is expressed as

$$C_{all} = \sum_{n=1}^{N} (1-\alpha_n) C_n^l + \alpha_n C_n^o \tag{12}$$

where $\alpha_n \in \{0,1\}$denotes the offloading decision of UE n, if UE n executes its task by $localcomputing$, $\alpha_n = 0$, otherwise, $\alpha_n = 1$.

III. Problem Formulation

In this section, we formulate the offloading and resource allocation for MEC system as an optimization problem, the objective of this paper is to minimize the sum cost combining execution delay and energy consumption of all users in MEC system. Under the constrain of maximum tolerable delay and computation capacity, the problem is formulated as follows:

$$\begin{aligned} \min_{\mathcal{A},\boldsymbol{f}} \ & \sum_{n=1}^{N} \alpha_n C_n^o + (1-\alpha_n) C_n^l \\ \text{s.t.} \quad & C1: \ \alpha_n \in \{0,1\}, \forall n \in N \\ & C2: \ (1-\alpha_n) T_n^l + \alpha_n T_n^o \le \tau_n, \forall n \in N \\ & C3: \ 0 \le f_n \le \alpha_n F, \forall n \in N \\ & C4: \ \sum_{n=1}^{N} \alpha_n f_n \le F, \forall n \in N \end{aligned} \tag{13}$$

In (13), $\mathcal{A} = [\alpha_1, \alpha_2, \ldots, \alpha_n]$ is the offloading decision vector, $\boldsymbol{f} = [f_1, f_2, \ldots, f_N]$ is the computational resource allocation. The objective of the optimization problem is to minimize the sum cost of the entire system.
$C1$ represents that each UE chooses to execute its computational task by $local\ computing$ or $offloading\ computing$.
$C2$ indicates that either executed by $local\ computing$ or $offloading\ computing$, the time cost should not exceed the maximum tolerable delay.
$C3$ makes sure that computational resource to be allocated for UE n can not exceed F of the MEC server.
$C4$ guarantees that the sum of computational resource allocated to the offloading UEs can not exceed F of the MEC server.

Problem (13) can be solved by finding optimal values of offloading decision vector $\mathcal{A}$ and computational resource allocation $\boldsymbol{f}$. However, for the fact that $\mathcal{A}$ is binary variable, the feasible set and objective function of problem (13) is not convex. Moreover, the size of problem (13) could increase very rapidly if the number of user is increasing, so it is NP hard solve this non-convex problem extended from the Knapsack problem [16]. Instead of solving the NP hard problem (13) by conventional optimization methods, we propose reinforcement learning methods to find the optimal $\mathcal{A}$ and $\boldsymbol{f}$.

IV. Problem Solution

In this section, we firstly define the specific state, action and reward of the DRL agent in detail. Then we introduce the classical Q-learning based theme. Furtherly to avoid the curse of dimensionality, we propose a DQN method which use a DNN to estimate the action-value function of Q-learning.

A. three key elements for RL

There are three key elements in the reinforcement learning method, namely state, action, reward, specifically to the system model in this article:

- **state**: the system state consists of two components $s = (tc, ac)$. We define tc as the sum cost of the entire system, $tc = C_{all}$; ac is the available computational capacity of the MEC server, and can be computed as $ac = F - \sum_{n=0}^{N} f_n$.
- **action**: in our system, the action consists of two parts, respectively the offloading decision of n UEs $\mathcal{A} = [\alpha_1, \alpha_2, \ldots, \alpha_n]$ and resource allocation $\boldsymbol{f} = [f_1, f_2, \ldots, f_N]$. So the action vector can be given as $[\alpha_1, \alpha_2, \ldots, \alpha_n, f_1, f_2, \ldots, f_N]$ with the combination of some possible values of $\mathcal{A}$ and $\boldsymbol{f}$.
- **reward**: for each step, the agent will get a reward $R(s, a)$ in a certain state s after executing each possible action a. In general, the reward function should be related to the objective function. Consequently, the objective of our optimization problem is to get the minimal sum cost and the goal of RL is get the maximum reward, so the value of reward should be negatively correlated to the size of the sum cost, we define the immediate reward as normalized $\frac{tc_{local} - tc(s,a)}{tc_{local}}$, where tc_{local} is the sum cost of all tasks executed by $local\ computing$ and $tc(s, a)$ gives the actual sum cost of current state.

However, the action space would increase rapidly if there are more and more UEs. To limit the size of action space, we propose a pre-classification step before learning process. For UE n, if constraint $T_n^l \leq \tau_n$ can not be satisfied by $local\ computing$, UE n will be divided into offloading UEs, α_n will be fixed to 1 in this decision period and the allocated computational resource f_n should satisfy $f_n \geq \frac{D_n}{\tau_n - \frac{B_n}{R_n}}$. In this way, we can reduce the possible value of $\mathcal{A}$ and $\boldsymbol{f}$ to limit the action space of the RL agent.

B. Q-learning method

Q-learning is a classical RL algorithm, which is a learning method of recording the Q value. Each state-action pair will have a value $Q(s, a)$. For each step, the agent compute and store $Q(s, a)$ in a Q-table, this value can be regarded as a long-term reward, then $Q(s, a)$ can be given as:

$$Q(s, a) = R(s, a) + \gamma * \max Q(s', a') \tag{14}$$

where s, a is the current state and action, s', a' is the next state and action. And we define γ as the learning parameter, γ is a constant that satisfies $0 \leq \gamma \leq 1$. It is worth noting that if γ tends to 0 that means the agent mainly considers the immediate reward, if γ tends to 1 that means the agent is also very focused on the future reward. For each step, the value of $Q(s, a)$ is iterated. In this way, we can get the optimal $\mathcal{A}$ and $\boldsymbol{f}$ Algorithm 1 shows the process of Q-learning algorithm.

Algorithm 1 Q-learning method

Initialize $Q(s, a)$
for each episode:
 Choose a random state s.
 for each step, **do**
  Choose an action a from all possible actions of state s
  Execute chosen a, observe reward and s'
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
  Let $s \leftarrow s'$
 until reach expected state $s_{terminal}$
end for

C. Deep Q Network method

Although Q-learning can solve problem (11) by obtaining the biggest reward, it is not so ideal. If we use Q-learning, that means for each state-action group, we need to compute and store its corresponding Q value in a table. But in practical problems, the possible state may be more than ten thousands. If we store all the Q values in the Q-table, the matrix $Q(s, a)$ would be very large. Then it can be difficult to get enough samples to traverse each state, which would lead to failures of the algorithm. Therefore we use a Deep Neural Network to estimate $Q(s, a)$ instead of computing Q value for each state-action pair, this is the basic idea of Deep Q Network (DQN). Algorithm 2 shows the detail of DQN algorithm.

Algorithm 2 Deep Q-learning with Experience Delay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = 1, M **do**
Initialize sequence $s_1 = x_1$ and preprecessed sequenced $\phi_1 = \phi(s_1)$
for $i = 1$, T **do**
With probability ε select random action a_t
otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
Execute action a_t in emulator and observe reward r_t and image x_{t+1}
Set $s_{t+1} = s_t, a_t, x_{t+1}$ and process $\phi_{t+1} = \phi(s_{t+1})$
Store transition $(\phi_t, a_t r_t, \phi_{t+1})$ in D
Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{otherwise} \end{cases}$
Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
end for
end for

V. Simulation Result

In this section, we present the simulation results to estimate the performance of the proposed theme. In simulation, we assume a scenario as follows. We consider an single small cell using a bandwidth W =10MHz., and an eNB deployed with an MEC server is located at the center. The UEs are randomly scattered within 200m distance away from the eNB. The computation capacity of the MEC server is $F = 5$GHz/sec and the CPU frequency of each UE is $f_n^l = 1$GHz/sec. The UE's transmission power and idle power are set to be $P_n = 500$mW and $P_n^i = 100$mW [17]. And we assume that the data size of the computation offloading B_n(in kbits) obeys uniform distribution between (300, 500), the number of CPU cycles D_n(in Megacycles) obeys uniform distribution between (900,1100). Also for simplicity, the decision weight of each UE is set to be $I_n^t = I_n^e = 0.5$.

We compare the proposed algorithms with other two methods with respect to some parameters as follows. "Full Local" stands for that all UEs execute their tasks by *local computing*. "Full offloading" stands for that all UEs offload their tasks to the MEC server and the whole computational resource F is allocated equally to each UE.

We first present the sum cost of the MEC system with an increasing total number of UEs in Fig. 2, where computation capacity of the MEC server is $F = 5$GHz/sec. Taken as a whole, when the number of UE keeps increasing, the sum cost of four methods increase naturally. In Fig. 2, the proposed DQN method can achieve the best result, then the Q-learning follows with a small gap, the performance of these two methods are relatively stable. And the Full Offloading curve is a little higher than DQN and Q-learning at 3 UEs point but increases much more rapidly when there are more UEs. This is because when the number of UEs becomes large, the

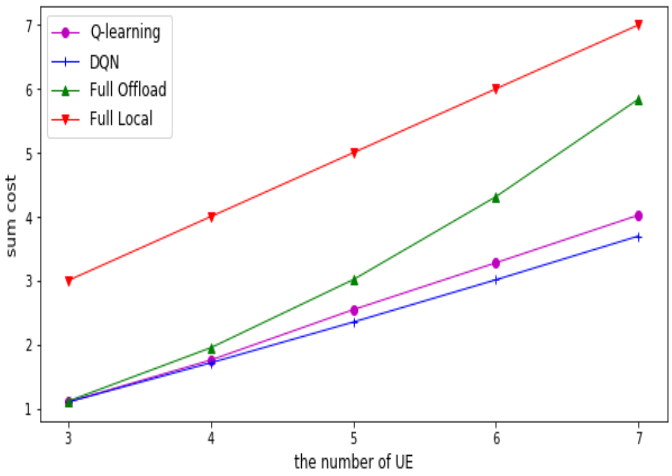

Figure 2. Sum cost verus the number of UE

capacity of the MEC server is not quite sufficient to offer all UEs by *offloading computing*. One MEC server with limited capacity should not serve too many UEs, so how to choose the offloaded UEs becomes quite important under this circumstance.

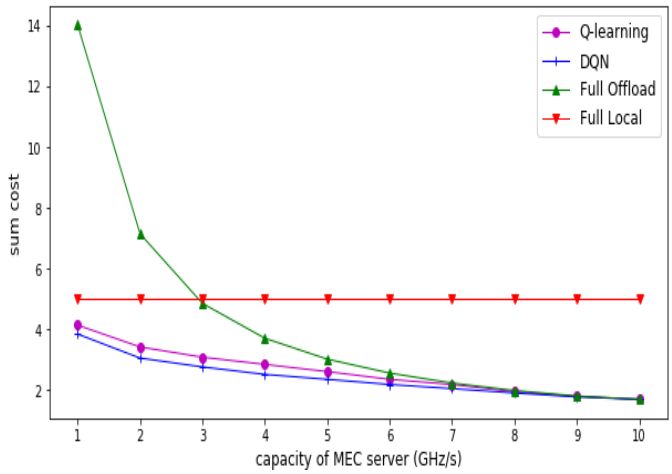

Figure 3. Sum cost verus the capacity of the MEC server

Fig. 3 shows the sum cost of the MEC system with respect to the increasing computational capacity of the MEC server, where the number of UEs is 5. From Fig. 3, the proposed DQN method can achieve best result and Q-learning has a small gap with DQN. In Fig. 3, the Full Local curve does not change with the increasing capacity of the MEC server, this is because the *local computing* does not employ the MEC servers computational resource. The other curves decrease with the the increasing computational capacity of the MEC server, because the execution time gets shorter if each UE is allocated more computational resource. What's more, when $F > 8$GHz/s, the sum costs of Full-Offloading, Q-learning and DQN decrease slowly and the performance of these offloading methods is almost the same. The result implies the sum cost of the MEC system is mainly constrained by other factors

such as radio resource when there is much more computational resource on the MEC server than *local computing*.

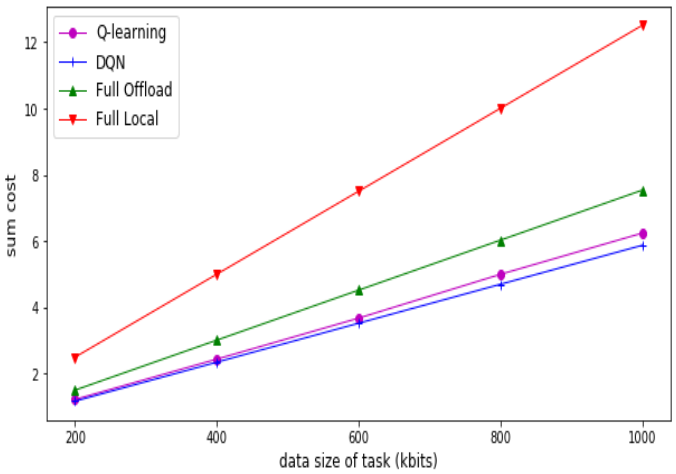

Figure 4. Sum cost verus the size of offloading data

Fig. 4 shows the sum cost of the MEC system with respect to the data size B_n of offloading task, where the number of UEs is 5. As is shown in Fig. 4, the sum costs of all methods increase with the increasing data size of offloading task, because bigger data size leads to more time and energy consumptions for offloading. to increase the sum cost of MEC system. The proposed DQN method can achieve best result for its increasing trend is slower than other methods. And the Full Local curve increases much more rapidly than other three methods with the increasing data size, which shows that the bigger data size of offloading task, the more profit on delay and energy consumptions we get from $offloading\ computing$. Also, we can see that the increase of data size can lead to a significant increase on the sum cost of MEC system, this is because D_n and B_n are positively related to achieve a double growth of our objective function.

VI. Conclusion

In this paper, we presented an integrated framework for multi-user computation offloading and resource allocation with mobile edge computing. We formulated the computation offloading decision and MEC computation resource allocation problems in this framework. Then, we derived the RL-based solutions to these two problems. The performance evaluation of the proposed schemes are presented in comparison with some baseline solutions. Simulation results demonstrate that the proposed schemes can achieve better performance than other baseline solutions under various system parameters. Future work is in progress to consider a more complex framework which takes interference and radio resource into consideration.

References

[1] M. Peng, Y. Sun, X. Li, Z. Mao, and C. Wang, "Recent advances in cloud radio access networks: System architectures, key techniques, and open issues," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2282–2308, thirdquarter 2016.

[2] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, Oct 2016.

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

[4] S. Li, N. Zhang, S. Lin, L. Kong, A. Katangur, M. K. Khan, M. Ni, and G. Zhu, "Joint admission control and resource allocation in edge computing for internet of things," *IEEE Network*, vol. 32, no. 1, pp. 72–79, Jan 2018.

[5] K. Kumar and Y. H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[6] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks," *IEEE Signal Processing Magazine*, vol. 31, no. 6, pp. 45–55, Nov 2014.

[7] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[8] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 1451–1455.

[9] S. T. Hong and H. Kim, "Qoe-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, June 2016, pp. 1–9.

[10] S. Xiao and X. lin Wang, "The method based on q-learning path planning in migrating workflow," in *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, Dec 2013, pp. 2204–2208.

[11] P. Zhao, H. Tian, C. Qin, and G. Nie, "Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing," *IEEE Access*, vol. 5, pp. 11 255–11 268, 2017.

[12] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 8, pp. 7432–7445, Aug 2017.

[13] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 2716–2720.

[14] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[15] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[16] A. Jaszkiewicz, "On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - a comparative experiment," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 402–412, Aug 2002.

[17] Y. Cao, T. Jiang, and C. Wang, "Optimal radio resource allocation for mobile task offloading in cellular networks," *IEEE Network*, vol. 28, no. 5, pp. 68–73, September 2014.