Multi-user Multi-channel Computation Offloading and Resource Allocation for Mobile Edge Computing

Samrat Nath*, Yaze Li*, Jingxian Wu*, and Pingzhi Fan†
*Department of Electrical Engineering, University of Arkansas, Fayetteville, AR 72701, USA
†Institute of Mobile Communications, Southwest Jiaotong University, Chengdu 611756, P. R. China
Email: *{snath, yazeli, wuj}@uark.edu, †pzfan@home.swjtu.edu.cn

***Abstract*—We study the problem of computation offloading and resource allocation in multi-user multi-channel mobile edge computing (MEC) systems. Each user equipment (UE) in the system has a computation-intensive and time-sensitive task that needs to be executed either locally or remotely in an MEC server. All UE tasks have individual deadline constraints that are treated as soft constraints. The MEC server has limited computational resources, which impose hard constraints on the overall offloading computation capacity. The objective is to minimize a cost function that is expressed as a weighted sum of energy consumption, delay, and deadline penalty of all UEs. The optimum design is performed with respect to three decision parameters: whether to offload a given task, which wireless channel to use during offloading, and how much MEC resources should be allocated for an offloaded task. We apply a Deep Reinforcement Learning approach known as Deep Deterministic Policy Gradient to solve the problem. Simulation results demonstrate that the proposed algorithm outperforms other existing schemes such as Deep Q-Network (DQN).**

***Index Terms*—mobile edge computing, computation offloading, resource allocation, deep reinforcement learning**

I. Introduction

With the ever-growing popularity of smart mobile devices in the emerging 5G era, new mobile applications such as natural language processing, face recognition, online interactive gaming, and augmented or virtual reality are generating great attention [1]. This kind of mobile applications typically require intensive computation and high energy consumption. However, due to the physical size constraint, mobile user equipment (UE) has limited battery life and computational capacity. Therefore, it is difficult for mobile UEs to meet the requirements of these mobile applications. In order to bridge the gap between the limited resources on UEs and the ever-increasing computational demands required by mobile applications, the technology of mobile edge computing (MEC) [2] has been proposed as a promising solution.

MEC is a new paradigm in cloud radio access network (C-RAN). Instead of using the remote public clouds such as Amazon Web Services and Microsoft Azure, MEC enhances the computational capability at the edge of mobile networks by deploying densely distributed high-performance servers close to mobile users [2]. It enables UEs to offload computational tasks to the MEC server associated with a base station (BS) through wireless channels. Through computation offloading, UEs can considerably reduce the latency and energy consumption and thus improve the Quality of Experience (QoE) of mobile applications. Therefore, the topic of computation offloading in MEC system has been attracting a lot of interests among researchers [3].

Computation offloading largely depends on the efficiency of wireless data transmission, which requires MEC systems to manage radio resources and computational resources and to efficiently perform computation tasks. Different strategies of MEC computation offloading and resource allocation for different design objectives have been widely investigated in the literature [4]–[10]. In general, the approaches for computation offloading in MEC can be categorized into two groups, namely, binary computation offloading [4]–[8] and partial computation offloading [9], [10]. Specifically, binary computation offloading means that a UE can either execute its computational task locally using its own central processing unit (CPU) or it can offload that task completely to the MEC server. In [4], the joint optimization problem of the binary computation offloading, resource allocation, and content caching strategy is formulated and solved by applying the alternating direction method of multipliers (ADMM). In partial computation offloading, the UE can offload parts of the task and execute the rest of it locally, which offers more flexibility than its binary counterparts. The latency-minimization problem for a multi-user MEC system is investigated in [10] by exploiting users' partial computation offloading and jointly optimizing the allocations of the computation and communication resources.

All the above-mentioned works intend to solve the complicated joint computation offloading and resource allocation as optimization problems, which are in general non-convex and challenging. Recently, with the explosive growth in the deep neural networks (DNNs), researchers have started exploiting DNNs, specifically Deep Reinforcement Learning (DRL) to solve these problems [7]–[9], [11]. The works in [7], [8], [11] adopts the Deep Q-Network (DQN) method [12], while Deep Deterministic Policy Gradient (DDPG) algorithm [13] is utilized in [9].

To this end, we focus on the design of a DRL-based scheme for the problem of computation offloading and resource al-

The work was supported in part by the U.S. National Science Foundation (NSF) under Award Number ECCS-1711087.

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

location in MEC by addressing three key questions: 1) how should a BS decide between local execution and offloading computation for a particular UE? 2) if offloading computation is decided for a UE, how can the BS choose a proper channel for that UE in order to obtain high wireless access efficiency given the possible co-channel interference among UEs sharing the same offloading channel? and 3) how much computational resources should the MEC server allocate to the offloading UEs? It is assumed that the offloading decision, channel allocation, and computational resource allocation are determined centrally by the BS, and then the results are sent to the UE. Our objective is to develop a DRL-based solution for efficient computation offloading and resource allocation in a multi-user multi-channel MEC system. We assume that the MEC server has a limited computation resource capacity. Specifically, the problem is formulated as a joint optimization of energy consumption and delay in task computation along with the consideration of individual deadline for each task. The problem is solved by employing the DDPG framework, which can deal with the continuous space of optimization variables. Simulation results show that the proposed DDPG-based solution outperforms other existing schemes such as DQN, which requires the discretization of the continuous optimization variables.

II. System Model and Problem Formulation

A. Network and Communication Model

A small-cell wireless network with one BS and N UEs is considered. The BS is equipped with an MEC server. Let $\mathcal{N} = \{1, 2, \cdots, N\}$ denote the set of UEs. There are M ($M < N$) wireless access channels and the set of channels is denoted as $\mathcal{M} = \{1, 2, \cdots, M\}$. We assume that each UE has a computation-intensive and time-sensitive task to be completed. Each UE could execute the task locally or offload the task to the MEC server through any of the M wireless channels. The capacity of the MEC server is limited and may not be sufficient for all UEs to offload tasks.

We denote $x_n \in \{0\} \cup \mathcal{M}$ as the computation offloading decision variable of UE n. Specifically, if $x_n \in \mathcal{M}$, then UE n chooses to offload the computation to the MEC server via the wireless channel x_n; if $x_n = 0$, then UE n chooses to execute its task locally. Given the computation offloading decision vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$ of all UEs, we can determine the uplink data rate of UE n that decides to offload the computation to the MEC server via a wireless channel $x_n \in \mathcal{M}$ as [14]

$$r_n(\boldsymbol{x}) = W \log_2 \left[1 + \frac{P_n h_n}{\sigma_0^2 + \sum_{i \in \mathcal{N} \setminus \{n\}: x_i = x_n} P_i h_i} \right], \quad (1)$$

where W is the channel bandwidth, P_n is the transmission power of UE n for uploading data, h_n is the channel gain between UE n and the BS, and σ_0^2 is the background noise power. According to the wireless interference model for urban cellular radio environment [14], we set the channel gain $h_n = l_n^{-\beta}$, where l_n is the distance between UE n and the BS, and β is the path loss exponent. From the communication model in (1), it can be observed that when too many UEs decide to offload computation via the same wireless channel simultaneously, they may incur severe interference, which will yield low data rates. This can negatively affect the performance of MEC computing.

Similar to many previous research works in mobile cloud computing and mobile networking [5], [7], [8], we consider a quasi-static scenario where the set of UEs $\mathcal{N}$ and the channel conditions remain fixed throughout a computation offloading period (e.g., several hundred milliseconds), while they may change across different periods.

B. Task Computation Model

We consider that each UE n has a computation-intensive and time-sensitive task $Z_n \triangleq (b_n, d_n, \tau_n)$, where b_n denotes the size of computation input data (e.g. program codes and input parameters) needed for computing the task, d_n denotes the total number of CPU cycles required to accomplish the computation task, and τ_n denotes the maximum tolerable delay of task Z_n. The CPU cycle d_n is positively related to the task size b_n. All three parameters may differ between different UEs due to different kinds of applications and features.

1) Local Computation: For the local computing approach, UE n executes its computation task Z_n locally using its CPU. Let f_n^l be the computation capability (i.e., CPU cycles per second) of UE n. Computational capabilities may differ among different UEs. The computation execution time of the task Z_n by local computing can then be expressed as

$$T_n^l = \frac{d_n}{f_n^l}. \quad (2)$$

The corresponding energy consumption of task Z_n is

$$E_n^l = \zeta_n d_n, \quad (3)$$

where ζ_n is the coefficient representing the energy consumption per CPU cycle. We set $\zeta_n = 10^{-27}(f_n^l)^2$ according to the measurement results reported in [15].

2) Offloading Computation: For the offloading computing approach, UE n will offload its computation task Z_n to the MEC server via a wireless channel and the MEC server will execute the computation task on behalf of the UE. This approach will be divided into three steps. First, UE n uploads input data (i.e., program codes and parameters) of size b_n to the BS through wireless access network and BS forwards data to the MEC server. Then the MEC server allocates part of its computational resource to execute the task Z_n, and finally the MEC server returns the execution output data to UE n.

In the first step, the transmission delay of UE n for offloading the input data of size b_n can be computed as

$$T_{n,t}^o(\boldsymbol{x}) = \frac{b_n}{r_n(\boldsymbol{x})}, \quad (4)$$

where $r_n(\boldsymbol{x})$ stands for the uplink data rate of UE n, and it depends on the computation offloading decision vector $\boldsymbol{x}$ as shown in the communication model (1).

The corresponding energy consumption of the first step is

$$E^o_{n,t}(\boldsymbol{x}) = P_n T^o_{n,t}(\boldsymbol{x}) = \frac{P_n b_n}{r_n(\boldsymbol{x})}. \quad (5)$$

In the second step of executing the computation task Z_n, the MEC server will incur processing delay, which can be expressed as

$$T^o_{n,p} = \frac{d_n}{f_n}, \quad (6)$$

where f_n denotes the computational resource (i.e., CPU cycles per second) allocated to UE n by the MEC server.

Denote $\boldsymbol{f} = (f_1, f_2, \cdots, f_N)$ as the MEC computational resource allocation vector for all the UEs, and F as the total resource of the MEC server. Therefore, the total amount of allocated resources can not exceed the entire computational resource of the MEC server, i.e.,

$$\sum\nolimits_{i=1}^{N} \mathbf{1}(x_i \neq 0) f_i \leq F, \quad (7)$$

where $\mathbf{1}(\kappa)$ is the indicator function with $\mathbf{1}(\kappa) = 1$ if the event κ is true and 0 otherwise.

During the computation at the MEC server, the UE n is assumed to be idle and the corresponding energy consumption is defined as

$$E^o_{n,p} = P^i_n T^o_{n,p} = \frac{P^i_n d_n}{f_n}, \quad (8)$$

where P^i_n denotes the power consumption by UE n during the idle state.

In the final step, UE n downloads the output data from the MEC server. Similar to many studies such as [5]–[7], we neglect the delay and energy consumption during this step. This is because the size of the computation output data is much smaller than the size of computation input data for many applications and the download data rate is also very high generally.

The total execution delay of UE n for task Z_n in the offloading computation approach can be computed by combining the transmission delay (4) and processing delay (6) as

$$T^o_n(\boldsymbol{x}, \boldsymbol{f}) = \frac{b_n}{r_n(\boldsymbol{x})} + \frac{d_n}{f_n}. \quad (9)$$

Similarly, the energy consumption of UE n is computed by combining the energy consumption during transmission (5) and idle state (8) as

$$E^o_n(\boldsymbol{x}, \boldsymbol{f}) = \frac{P_n b_n}{r_n(\boldsymbol{x})} + \frac{P^i_n d_n}{f_n}. \quad (10)$$

C. Problem Formulation

Given the computation offloading decision vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_N)$ and the MEC computational resource allocation vector $\boldsymbol{f} = (f_1, f_2, \cdots, f_N)$, the delay and energy consumption for UE n can be expressed respectively as

$$T_n(\boldsymbol{x}, \boldsymbol{f}) = \mathbf{1}(x_n = 0) T^l_n + \mathbf{1}(x_n \neq 0) T^o_n(\boldsymbol{x}, \boldsymbol{f}), \quad (11)$$

$$E_n(\boldsymbol{x}, \boldsymbol{f}) = \mathbf{1}(x_n = 0) E^l_n + \mathbf{1}(x_n \neq 0) E^o_n(\boldsymbol{x}, \boldsymbol{f}). \quad (12)$$

The objective of this paper is to minimize the weighted sum cost of delay and energy consumption of all UEs in the MEC system, which is defined as

$$C(\boldsymbol{x}, \boldsymbol{f}) = \sum\nolimits_{n=1}^{N} E_n(\boldsymbol{x}, \boldsymbol{f}) + \sum\nolimits_{n=1}^{N} \omega_n T_n(\boldsymbol{x}, \boldsymbol{f}), \quad (13)$$

where ω_n denotes the weight parameter for the delay experienced by UE n. Different UEs may have different requirements for delay depending on the task. For UEs dealing with time-sensitive tasks, ω_n can be set to large values to prioritize faster execution.

The optimization problem is formulated as follows

$$\begin{aligned} \text{(P1)} \ \min_{\boldsymbol{x},\boldsymbol{f}} \quad & C(\boldsymbol{x}, \boldsymbol{f}) \\ \text{s.t.} \ \ \text{(C1)} \quad & x_n \in \{0\} \cup \mathcal{M}, \ \forall n \in \mathcal{N} \\ \text{(C2)} \quad & 0 \leq f_n \leq F, \ \forall n \in \mathcal{N} \\ \text{(C3)} \quad & \sum\nolimits_{i=1}^{N} \mathbf{1}(a_i \neq 0) f_i \leq F \\ \text{(C4)} \quad & T_n(\boldsymbol{x}, \boldsymbol{f}) \leq \tau_n \ \forall n \in \mathcal{N} \end{aligned}$$

The resource capacity constraint (C3) is regarded as a hard constraint. However, it may be possible that there is no feasible set of decision variables which satisfy both the constraints (C3) and (C4) simultaneously. So, the deadline constraint is treated as a soft constraint for solution feasibility. We introduce a penalty term for the deadline constraint such that

$$C_\tau(\boldsymbol{x}, \boldsymbol{f}) = \sum\nolimits_{n \in \mathcal{N}} \max\left[T_n(\boldsymbol{x}, \boldsymbol{f}) - \tau_n, 0\right]. \quad (14)$$

Then, the problem formulation is redefined as

$$\begin{aligned} \text{(P2)} \ \min_{\boldsymbol{x},\boldsymbol{f}} \quad & J(\boldsymbol{x}, \boldsymbol{f}) = C(\boldsymbol{x}, \boldsymbol{f}) + \omega_\tau C_\tau(\boldsymbol{x}, \boldsymbol{f}) \\ \text{s.t.} \quad & \text{(C1), (C2), and (C3)} \end{aligned}$$

where ω_τ is weight for deadline penalty.

Problem (P2) can be solved by finding optimal values of the computation offloading decision vector $\boldsymbol{x}$ and the MEC computational resource allocation vector $\boldsymbol{f}$. However, the feasible set and objective function of problem (P2) is not convex. Moreover, the size of problem (P2) increases exponentially with the increasing number of UEs. Instead of applying conventional optimization methods to solve the NP-hard problem (P2), we propose a DRL-based method to find the optimal $\boldsymbol{x}$ and $\boldsymbol{f}$.

III. DRL-based Solution for Computation Offloading and Resource Allocation

Since DRL is considered as an advanced Reinforcement Learning (RL) method implemented with DNNs, we first review RL briefly and then present the proposed DRL-based solution.

A. RL Framework

Generally, the RL framework is well-suited for providing solutions to complicated decision-making processes [16]. The framework consists of three key elements, state, action, and reward. An RL agent interacts with its environment in discrete time steps. At each time t, the agent's behavior is defined by

a policy μ, which maps states to actions $\mu : s_t \rightarrow a_t$. After the RL agent selects an action a_t according to the policy μ, the environment then returns a scalar reward r_t and makes a transition from state s_t to s_{t+1}. The RL algorithm aims at learning an optimal policy in order to maximize the expected long-term accumulative reward.

To interpret problem (P2) in the RL framework, we define the key elements according to the system model as follows:

1) State: The system state at an arbitrary time index t is defined as

$$\begin{aligned} \boldsymbol{s}_t &\triangleq \{\boldsymbol{x}(t), \boldsymbol{f}(t)\} \\ &= \{x_1(t), x_2(t), \cdots, x_N(t), f_1(t), f_2(t), \cdots, f_N(t)\}. \end{aligned} \tag{15}$$

In particular, the dimension of system state vector is $2N$, which includes the optimization variables i.e., the computation offloading decisions $x_n(t) \in \{0\} \cup \mathcal{M}$ and resource allocation $f_n(t) \in [0, F]$ for all UEs.

2) Action: We use the action to describe how we move between two consecutive system states. Specifically, we denote $x'_n(t) = x_n(t+1) \in \{0\} \cup \mathcal{M}$ as the new computation offloading decision for UE n and $\Delta f_n(t) = f_n(t+1) - f_n(t)$ as the change in the allocated computational resource by the MEC server for UE n. The system action is then defined as

$$\boldsymbol{a}_t \triangleq \{x'_1(t), \cdots, x'_N(t), \Delta f_1(t), \cdots, \Delta f_N(t)\}. \tag{16}$$

Let $\mathcal{N}_t^o$ denote the set of UEs who decide to offloading their computation to the MEC server at time $t+1$, i.e., $\mathcal{N}_t^o = \{n : x'_n(t) \neq 0\}$. $|\mathcal{N}_t^o|$ denotes the total number of offloading UEs at time $t+1$. In order to satisfy the constraint (7) regarding the computational resource capacity in the MEC server, the support for the variable $\Delta f_n(t)\ \forall\ n \in \mathcal{N}$ is defined such that

$$\Delta f_n(t) \in \begin{cases} \{-f_n(t)\}, & \text{if } x'_n(t) = 0, \\ (-f_n(t), K_t(F - f_n(t))], & \text{otherwise}, \end{cases} \tag{17}$$

where $K_t = \frac{F - \sum_{n \in \mathcal{N}_t^o} f_n(t)}{|\mathcal{N}_t^o| F - \sum_{n \in \mathcal{N}_t^o} f_n(t)}$.

3) Reward: Given a particular state $\boldsymbol{s}_t$ and an action $\boldsymbol{a}_t$, we can determine the transition of state from $\boldsymbol{s}_t$ to $\boldsymbol{s}_{t+1}$. Moreover, at an arbitrary time index t, the objective function in (P2) can be expressed as a function of the state variable.

$$J(\boldsymbol{s}_t) = J\{\boldsymbol{x}(t), \boldsymbol{f}(t)\} \tag{18}$$

Based on $J(\boldsymbol{s}_t)$ and $J(\boldsymbol{s}_{t+1})$, we define the reward of the state-action pair $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ as

$$r_t \triangleq J(\boldsymbol{s}_t) - J(\boldsymbol{s}_{t+1}). \tag{19}$$

B. DRL-based Solution

The well-known Q-learning algorithm [16] provides a viable way to solve the RL problem. Specifically, the algorithm solves the Q function; a function of the state-action pair which quantifies the long-term accumulative reward. The Q function is given by

$$\begin{aligned} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) &\leftarrow Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ &+ \alpha \left[r_t + \gamma \max_{\boldsymbol{a}_{t+1}} Q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \right], \end{aligned} \tag{20}$$

where α is the learning rate, and γ is the discount factor. We consider $\gamma = 1$ in this paper.

The complexity in solving (20) grows exponentially as the sizes of the state space and action space increase. A promising approach to address this issue is to approximate the Q function with a finite number of parameters. DQN [12] proves to be an efficient solution methodology, which exploits the architecture of DNN in order to approximate the Q function and thus to facilitate solving (20).

Even though DQN can successfully solve problems in high-dimensional state spaces, it can handle only discrete and low-dimensional action spaces. For problems with continuous action space, the action space has to be discretized first before applying DQN. This will result in loss of precision and increase in complexity.

We propose to address this problem by applying DDPG [13], which can be applied to problems with continuous action space and state space. In DDPG, an actor-critic approach is adopted by using two separate DNNs, where the actor network $\mu(\boldsymbol{s}|\theta^\mu)$ approximates the policy function μ, and the critic network approximates the Q function $Q(\boldsymbol{s}, \boldsymbol{a}|\theta^Q)$, where θ^μ and θ^Q are the neural network weights of the actor and critic networks, respectively. Details of the proposed solution is described in Algorithm 1, where mod() is the remainder operation and τ denotes the weight update rates for target actor network $\mu'(\boldsymbol{s}|\theta^{\mu'})$ and target critic network $Q'(\boldsymbol{s}, \boldsymbol{a}|\theta^{Q'})$.

Compared to the vanilla DDPG algorithm [13], we propose modifications at the start of every training episode in Algorithm 1. In order to facilitate faster convergence, we set the initial state at the beginning of every 100 episodes as the best state from all the previously observed states. A similar approach with DQN is applied in [8]. In our design of the DNN architecture for both the actor and critic networks, two hidden layers are considered. The dimensions of the first and second hidden layers are $4N$ and $3N$, respectively. Adaptive parameter noise approach [17] in the actor network is employed for action exploration, while adaptive moment estimation (Adam) method [18] is adopted for learning the neural network parameters.

IV. Simulation Results

Simulation results are presented in this section to demonstrate the performance of the proposed algorithm obtained through DDPG. In the simulations, we consider a wireless small cell BS with a coverage radius of 50m. There are N = 5 UEs randomly scattered over the coverage region. The BS consists of M = 2 channels and the channel bandwidth is W = 5 MHz. The UE's transmission power and idle power are set to be P_n = 500 mW and P_n^i = 100 mW [19]. The background noise power is set as $\sigma_0^2 = 10^{-9}$ W [9] and β = 3 is the path loss exponent.

For the computation task, we assume the data size of the computation offloading b_n (in kbits) is uniformly distributed between $[400, 600]$, the number of CPU cycles d_n (in Megacycles) is uniformly distributed between $[900, 1100]$, and the maximum tolerable delay τ_n (in seconds) is uniformly

Algorithm 1 Proposed Solution using DDPG

Input: System environment parameters, experience replay buffer size $|\mathcal{R}|$, mini-batch size B, soft update rate for target networks τ, number of training episodes $K_{\max}$, number of time steps in each episode $T_{\max}$.

Initialization: Randomly initialize actor network $\mu(\boldsymbol{s}|\theta^{\mu})$ and critic network $Q(\boldsymbol{s},\boldsymbol{a}|\theta^{Q})$ with weights θ^{μ} and θ^{Q}, respectively.
Initialize associated target networks μ' and Q' with weights $\theta^{\mu'} \leftarrow \theta^{\mu}, \theta^{Q'} \leftarrow \theta^{Q}$.
Initialize the experience replay buffer $\mathcal{R}$.
Initialize $J_{\min}$ to a very large value.
for each episode $k = 1, 2, \cdots, K_{\max}$ **do**
  Randomly generate an initial state $\boldsymbol{s}_1$
  if mod(k,100) = 0 **then**
    Change initial state to current best result: $\boldsymbol{s}_1 \leftarrow \boldsymbol{s}^*$
  end if
  for each episode $t = 1, 2, \cdots, T_{\max}$ **do**
    Determine the computation offloading decision and resource allocation vector by selecting an action $\boldsymbol{a}_t = \mu(\boldsymbol{s}_t|\theta^{\mu}) + \Delta\mu$ using the current policy θ^{μ} and generating exploration noise $\Delta\mu$.
    Execute action $\boldsymbol{a}_t$ and observe the new state $\boldsymbol{s}_{t+1}$ and reward $r_t = J(\boldsymbol{s}_t) - J(\boldsymbol{s}_{t+1})$ from the simulation environment.
    if $J(\boldsymbol{s}_{t+1}) < J_{\min}$ **then**
      Update the result for best state observed so far $\boldsymbol{s}^* \leftarrow \boldsymbol{s}_{t+1}$ and $J_{\min} \leftarrow J(\boldsymbol{s}_{t+1})$
    end if
    Save the transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ into the the replay buffer $\mathcal{R}$.
    Randomly sample a mini-batch of B transitions $\{(\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_{i+1})\}_{i=1}^{B}$ from $\mathcal{R}$.
    Update the critic network $Q(\boldsymbol{s},\boldsymbol{a}|\theta^{Q})$ by minimizing the loss L_B obtained from the samples:

$$L_B = \frac{1}{B}\sum_{i=1}^{B}\left[r_i + Q'\big(\boldsymbol{s}_{i+1}, \mu'(\boldsymbol{s}_{i+1}|\theta^{\mu'})|\theta^{Q'}\big) - Q(\boldsymbol{s}_i, \boldsymbol{a}_i|\theta^{Q})\right]^2 \quad (21)$$

    Update the actor network $\mu(\boldsymbol{s},\boldsymbol{a}|\theta^{Q})$ by using the sampled policy gradient:

$$\frac{1}{B}\sum_{i=1}^{B}\Delta_{\boldsymbol{a}} Q(\boldsymbol{s}_i, \boldsymbol{a}|\theta^{Q})\,|_{\boldsymbol{a}=\boldsymbol{a}_i}\,\Delta_{\theta^{\mu}}\mu(\boldsymbol{s}_i|\theta^{\mu}) \quad (22)$$

    Update the target networks:

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \quad \text{and} \quad \theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$$

  end for
end for

Output: Decision variable s^* and total system cost $J_{\min}$.

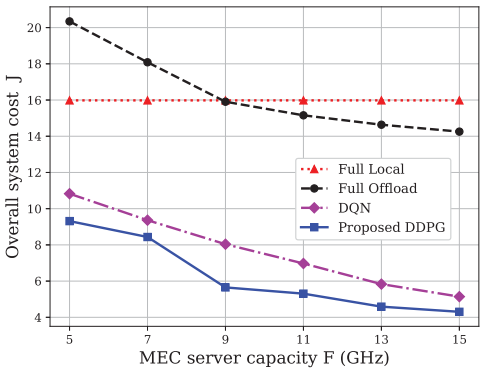

Fig. 1. Overall system cost v.s. the capacity of MEC server.

distributed between $[0.5, 1.5]$. The computation capacity of the MEC server is F = 5 GHz and the CPU frequency of each UE is f_n^l = 0.5 GHz. The delay weight parameters are $\omega_n = 1$ for all UE n and the deadline penalty weight is $\omega_\tau = 10$.

For learning the neural network parameters, the learning rates for the actor and critic networks are 10^{-4} and 10^{-3}, respectively, and the soft update rate for the target networks is $\tau = 10^{-3}$. Moreover, the number of episodes is $K_{\max} = 1000$, the maximum number of steps in each episode is $T_{\max} = 100$, the experience replay buffer size is $|\mathcal{R}| = 50000$, and the mini-batch size is $B = 128$.

For comparing the performance of the proposed DDPG-based solution, the baseline schemes are introduced as follows: (1) *Full Local*: all UEs execute their tasks by local computing. (2) *Full Offload*: all UEs offload their tasks to the MEC server and the whole computational resource F is allocated equally to each UE. For UE $n \in \mathcal{N}$, the selected channel is $x_n = \text{mod}\ (n-1, M) + 1$.
(3) *DQN-based Solution*: The conventional discrete action space based DRL algorithm, DQN [12], is also implemented as a solution approach. The discrete space for $\boldsymbol{x}'$ is $\{0\} \cup \mathcal{M}$, and the support space for $\Delta\boldsymbol{f}$ is discretized into finite L levels. Specifically, given the state variable $f_n(t)$ for each UE n at time index t,

$$\Delta f_n(t) \in \{-f_n(t), -f_n(t) + \delta_{n,t}, -f_n(t) + 2\delta_{n,t}, \cdots, 0, \delta_{n,t}, 2\delta_{n,t}, \cdots, K_t(F - f_n(t))\},$$

where, $\delta_{n,t} = \frac{K_t(F - f_n(t)) + f_n(t)}{L-1}$.

Consequently, for N UEs, the size of the action space becomes $(M+1)^N \times L^N$. We arbitrarily set L = 5 and maintain the same structure for hidden layers as mentioned in Section III-B. Moreover, ϵ-greedy exploration method with $\epsilon = 0.01$ is adopted for exploring the actions during network training.

Fig. 1 shows the overall system cost as a function of the resource capacity of the MEC server (F) with different optimization algorithms. The performance of the full local approach remains the same regardless of the value of F, because the UEs do not utilize the computational resource of the MEC

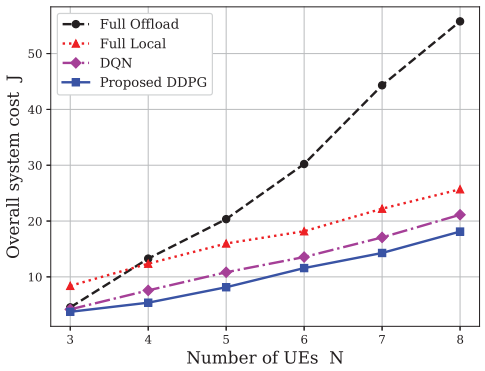

Fig. 2. Overall system cost v.s. the number of UEs.

TABLE I
EFFECT OF NUMBER OF CHANNELS (M) ON SYSTEM COST WITH $N = 8$

M	2	3	4	5
Full Offload	55.77	46.22	40.45	29.21
Full Local	25.69	25.69	25.69	25.69
DQN	22.31	21.21	21.08	20.89
Proposed DDPG	**18.12**	**17.74**	**17.49**	**16.63**

server. All other methods show improvement in performance with increasing values of F due to the extra resources from MEC. The proposed DDPG-based algorithm achieves the best result and it demonstrates at least 9.9% improvement over its DQN-based counterpart. While increasing the number of discrete levels L in the DQN approach may help to get better results for DQN, but it will yield a very high-dimensional action space and thus a much slower convergence than DDPG.

Fig. 2 demonstrates the impacts of the number of UEs (N) on the overall cost of the MEC system. As expected, the overall system cost becomes larger as N increases. While the performance of all the other methods are relatively stable with respect to N, the cost in full offload approach increases much more rapidly with the increase in N. This is because when the number of UEs becomes large, the resource capacity of the MEC server F is not sufficient to support all UEs for offloading computation. The proposed DDPG-based algorithm performs the best, and it achieves at least 10.5% performance improvement compared to the DQN-based solution.

Table I shows the effects of the number of channels (M) on system cost. As M increases, the possibility of interference among the UEs decreases, which results in better uplink data rates and thus lower system cost. Again the proposed DDPG-based algorithm achieves the best performance compared to all other algorithms, and it achieves at least 14.2% performance improvement compared to the DQN-based solution.

V. CONCLUSION

We have studied the problem of computation offloading and resource allocation in multi-user multi-channel MEC systems, where multiple UEs have computation-intensive and time-sensitive tasks that need to be executed either locally or remotely in an MEC server. All the UE tasks have individual deadline constraints and the MEC server has a limited computational resource capacity. The problem is formulated as the joint optimization of total energy consumption, delay, and deadline penalty of all the UEs. DDPG is applied to solve the problem and simulation results have shown that the proposed algorithm outperforms other existing schemes such as DQN.

REFERENCES

[1] W. Zhang, Y. Wen, J. Wu, and H. Li, "Toward a unified elastic computing platform for smartphones with cloud support," *IEEE Network*, vol. 27, no. 5, pp. 34–40, Sep. 2013.
[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] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.
[4] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.
[5] 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, Oct. 2016.
[6] 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.
[7] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *Wireless Communications and Networking Conference (WCNC)*, Barcelona, Spain, Apr. 2018, pp. 1–6.
[8] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digital Communications and Networks*, vol. 5, no. 1, pp. 10–17, Feb. 2019.
[9] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," *arXiv preprint arXiv:1812.07394*, 2018.
[10] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, Aug. 2018.
[11] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Emerging Topics in Computing*, 2019.
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[14] T. S. Rappaport *et al.*, *Wireless communications: principles and practice*. New Jersey, USA: Prentice Hall PTR, 1996, vol. 2.
[15] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE Conference on Computer Communications (INFOCOM)*, Orlando, FL, USA, Mar. 2012, pp. 2716–2720.
[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press, 2018.
[17] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, "Parameter space noise for exploration," in *International Conference on Learning Representations*, Vancouver, Canada, Apr. 2018.
[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[19] 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, Sep. 2014.