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

Samrat Nath and Jingxian Wu
Department of Electrical Engineering, University of Arkansas, Fayetteville, AR 72701, USA
Email: {snath, wuj}@uark.edu

GLOBECOM 2020 - 2020 IEEE Global Communications Conference | 978-1-7281-8298-8/20/$31.00 ©2020 IEEE | DOI: 10.1109/GLOBECOM42002.2020.9348161

***Abstract*—We study the problem of dynamic computation offloading and resource allocation in mobile edge computing (MEC) systems consisting of multiple mobile users (MUs) with stochastic task arrivals and wireless channels. Each MU can execute its task either locally or remotely in an MEC server. The objective is to identify the optimum scheduling scheme that can minimize the long-term average weighted sum of energy consumption and delay of all MUs, under the constraints of limited transmission power per MU and limited computation resources at the MEC server. The optimum design is performed with respect to three decision parameters: whether to offload a given task, how much transmission power to be allocated for offloading, and how much MEC resources to be allocated for an offloaded task. We propose to solve the problem by developing a dynamic scheduling strategy based on deep reinforcement learning (DRL) with deep deterministic policy gradient (DDPG). Simulation results show that the proposed algorithm outperforms other existing strategies such as deep Q-network (DQN).**

***Index Terms*—mobile edge computing, computation offloading, deep reinforcement learning, deep deterministic policy gradient**

I. Introduction

The ever-growing popularity of smart mobile devices (SMDs) and the emergence of the Internet-of-Things (IoT) are driving the development of many new applications such as online interactive gaming, face recognition, virtual/augmented reality, etc. These applications typically require intensive computation and high energy consumption. However, an SMD has limited battery life and computational capacity (processing speed), which makes it difficult for the SMD to meet the stringent requirements of these mobile applications. Mobile edge computing (MEC) has recently emerged as a promising technology to bridge the gap between the resource-limited SMDs and the computation-intensive applications [1].

Unlike conventional cloud computing systems, which rely on remote public clouds with high transmission latency, MEC offers computational capability within the radio access network by deploying densely distributed high-performance servers in proximity to mobile users (MUs) [1]. It allows MUs to offload computational tasks to the MEC server connected to a base station (BS) through the wireless network. MUs can significantly reduce the computation latency and energy consumption through computation offloading and thus improve the Quality of Experience (QoE) of mobile applications. Therefore, there have been growing interests on computation offloading in MEC systems [1].

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

The efficiency of computation offloading relies critically on how the limited communication, power, and computational resources are managed in an MEC system. Various computation offloading strategies with different design objectives and resource allocation schemes have been studied extensively in the literature [2]–[9]. Generally, the computation offloading approaches in MEC can be classified into two types, namely, partial computation offloading [2], [9], and binary computation offloading [3]–[8]. Specifically, in binary computation offloading, an MU can either execute its computational task on the local device or offload that task entirely to the MEC server. On the other hand, in partial computation offloading, the MU can offload fractional parts of the task and execute the rest of it locally, which offers more flexibility. In [2], an online algorithm based on Lyapunov optimization is developed for joint radio and computational resource management for multi-user MEC systems. The alternating direction method of multipliers (ADMM) is applied to solve the joint optimization problem of computation offloading, resource allocation, and content caching strategy in [3]. An iterative search algorithm is proposed to study the energy-latency tradeoff for energy-aware offloading in [4]. The scheme designed in [8] jointly minimizes energy consumption, delay, and deadline penalty of all the users in a multi-channel MEC system. However, most of these problems do not consider dynamic channel conditions and/or dynamic task arrivals. In practice, the MEC systems have time-varying stochastic channel conditions and task arrivals.

The complicated joint computation offloading and resource allocation in MEC is usually formulated as non-convex optimization problems, which are in general very challenging to solve. With the explosive growth of interest in deep neural networks (DNNs), researchers have recently started adopting Deep Reinforcement Learning (DRL) algorithms to solve these problems [5]–[9]. The proposed solutions in [6], [7] exploits the Deep Q-Network (DQN) method [10], while [8], [9] utilizes the Deep Deterministic Policy Gradient (DDPG) algorithm [11]. However, the work in [8] assumes the channel conditions to be quasi-static, and no constraint on the computational capacity of the MEC server is considered in [9].

In this paper, we propose to develop an online DRL-based scheme for dynamic computation offloading and resource allocation in a resource-constrained multi-user MEC system by addressing three key questions: 1) whether a given task should be executed locally at an MU or offloaded to MEC? 2) how much transmission power should be allocated to a

978-1-7281-8298-8/20/$31.00 ©2020 IEEE

given MU for task offloading? and 3) how much computational resources should be allocated by the MEC server for a given task? In the proposed MEC framework, the time is divided into slots, and the channel conditions and task arrivals are assumed to be time-varying and stochastic. The offloading decision, power allocation, and computational resource allocation are determined centrally by the BS at the beginning of each time slot, and then the results are forwarded to the MU. Our objective is to design an online DRL-based solution for efficient computation offloading and resource allocation. We assume that the MEC server has a limited computation resource capacity and all the MUs have individual constraints on transmission power. Specifically, we formulate an optimization problem to minimize the long-term average of a cost function, which is a weighted sum of energy consumption and delay of all MUs. We propose to solve the problem by using a DDPG-based method, which can deal with the continuous space of optimization variables. Simulation results demonstrate that the proposed DDPG-based solution outperforms other existing strategies such as DQN, which requires the discretization of the optimization variables.

II. System Model and Problem Formulation

Consider a multi-user MEC system with one 5G small-cell BS with M antennas, one MEC server, and a set of $N \leq M$ mobile users denoted by $\mathcal{N} = \{1, 2, \cdots, N\}$. The BS is connected directly with the MEC server with a total computation capacity of F (in CPU cycles per second). The system bandwidth is W (in Hz). We adopt a discrete-time model, where the time domain is slotted with equal length T_s (in seconds) and indexed by $\mathcal{T} = \{0, 1, \cdots\}$.

A. Task Model

Assume there are K computation-intensive heterogeneous tasks denoted by the set $\mathcal{K} \triangleq \{1, 2, \cdots, K\}$. Each task $k \in \mathcal{K}$ is characterized by two parameters; b_k (in bits) denotes the size of computation input data (e.g. program codes and input parameters) and d_k (in cycles per second) denotes the amount of computing resources required for the task. At the beginning of each time slot, each MU requests a single task randomly from the set $\mathcal{K}$, where one task may be requested by multiple users simultaneously. The popularity of each task $\phi_{k,t}$ is dynamic and follows Zipf distribution [12]. The popularity profile vector is defined as $\boldsymbol{\phi}_t \triangleq \{\phi_{k,t}\}_{k\in\mathcal{K}}$. Given the popularity rank of task k during slot t is $z_{k,t} \in \mathcal{K}$, the popularity of the corresponding task can be expressed as

$$\boldsymbol{\phi}_{k,t} = \frac{z_{k,t}^{-\eta}}{\sum_{l=1}^{K} z_{l,t}^{-\eta}}, \tag{1}$$

where Zipf parameter $\eta \geq 0$ controls the skewness of popularity. As the value of η increases, the popularity difference among the tasks becomes larger.

B. Communication Model

The BS and the MUs form a multi-user multiple-input multiple-output (MIMO) system, with M antennas at the BS serving N single-antenna MUs. The channel conditions between the BS and MUs are described by the $M \times N$ channel matrix $\boldsymbol{H}_t = [\boldsymbol{h}_{1,t}, \cdots, \boldsymbol{h}_{N,t}]$, where $\boldsymbol{h}_{n,t} \in \mathbb{C}^{M\times 1}$ is the channel vector of the n-th MU. The Gaussian Markov block fading autoregressive model [13] is adopted to characterize the temporal channel correlation between consecutive slots. The channel vector for the n-th MU can be expressed as

$$\boldsymbol{h}_{n,t} = \rho_n \boldsymbol{h}_{n,t-1} + \sqrt{1-\rho_n^2}\boldsymbol{e}_t, \tag{2}$$

where ρ_n is the normalized channel correlation coefficient for the n-th MU, the error vector $\boldsymbol{e}_t \in \mathbb{C}^{M\times 1}$ is uncorrelated with $\boldsymbol{h}_{n,t}$, and it is distributed such that $\boldsymbol{e}_t \sim \mathcal{CN}(\boldsymbol{0}, \sigma_e^2 \boldsymbol{I}_M)$, with $\boldsymbol{I}_M$ being a size M identity matrix.

Denote the transmission power of the n-th MU at the t-th slot as $p_{n,t} \in [0, P_n^{\max}]$, where $P_n^{\max}$ is the maximum transmission power of the n-th MU. The BS manages the uplink transmissions of multiple single-antenna MUs by employing the linear detection algorithm zero-forcing (ZF) [9], [14]. With the ZF detector at the BS, the signal-to-interference-plus-noise ratio (SINR) for the signal from the n-th MU is

$$\gamma_{n,t} = \frac{p_{n,t}}{\sigma^2[(\sqrt{\boldsymbol{P}_t}^T \boldsymbol{H}_t^H \boldsymbol{H}_t \sqrt{\boldsymbol{P}_t})^{-1}]_{nn}}, \tag{3}$$

where σ^2 is the noise power, $\boldsymbol{P}_t = \text{diag}\{\boldsymbol{p}_t\}$ is a diagonal matrix with $\boldsymbol{p}_t = [p_{1,t}, \cdots, p_{N,t}]^T$ on its main diagonal, the operators $\boldsymbol{A}^T$ and $\boldsymbol{A}^H$ respectively represent the matrix transpose and Hermitian operations, and $[\boldsymbol{A}]_{mn}$ denotes the (m, n)-th element of the matrix $\boldsymbol{A}$. The transmission data rate from the n-th MU to the BS at slot t can be expressed as [9]

$$r_{n,t} = W \log_2(1 + \gamma_{n,t}). \tag{4}$$

C. Computation Model

Denote $x_{n,t} \in \{0, 1\}$ as the computation offloading decision variable of the n-th MU at slot t. Specifically, if $x_{n,t} = 1$, then the n-th MU chooses to offload its current computation task to the MEC server via the wireless link; if $x_{n,t} = 0$, then the n-th MU chooses to execute its task locally. The computation offloading decision vector for all MUs is represented by $\boldsymbol{x}_t \triangleq [x_{1,t}, \cdots, x_{N,t}]^T$. Denote $k_t^n \in \mathcal{K}$ as the index of the task requested by the n-th MU at the beginning of time slot t, and $\boldsymbol{k}_t \triangleq [k_t^1, \cdots, k_t^N]^T$ as the task request vector.

1) Local Execution: In the local execution approach, the n-th MU executes its computation task k_t^n locally using its own CPU. Let f_n^l be the computation capability (in CPU cycles per second) of the n-th MU. Different MUs may have different computational capabilities. The computation time of task k_t^n by local execution can then be expressed as

$$T_{n,t}^l = \frac{d_{k_t^n}}{f_n^l}. \tag{5}$$

The corresponding energy consumption is

$$E_{n,t}^l = \zeta_n d_{k_t^n}, \tag{6}$$

where the coefficient ζ_n denotes the energy consumption per CPU cycle, which depends on the chip architecture at the MU device. We set $\zeta_n = 10^{-27}(f_n^l)^2$ in this paper [15].

2) MEC Server Execution: In this approach, the MEC server will execute the computation task on behalf of the MU. This approach is divided into three steps. First, the n-th MU uploads its task data of size $b_{k_t^n}$ to the BS through the wireless channel, and the BS forwards that data to the MEC server. Second, the MEC server allocates part of its computational resources to execute the task. Finally, the MEC server returns the execution results to the n-th MU.

In the first step, the transmission delay of task offloading by the n-th MU during slot t can be computed as

$$T_{n,t}^x = \frac{b_{k_t^n}}{r_{n,t}}. \tag{7}$$

The corresponding energy consumption of the first step is

$$E_{n,t}^x = p_{n,t} T_{n,t}^x = \frac{p_{n,t} b_{k_t^n}}{r_{n,t}}. \tag{8}$$

In the second step during task execution, the processing delay incurred by the MEC server is

$$T_{n,t}^p = \frac{d_{k_t^n}}{f_{n,t}}, \tag{9}$$

where $f_{n,t}$ denotes the computational resource (in CPU cycles per second) allocated to the n-th MU by the MEC server during slot t. Denote $\boldsymbol{f}_t \triangleq [f_{1,t}, \cdots, f_{N,t}]^T$ as the MEC computational resource allocation vector for all the MUs.

In the final step, the n-th MU downloads the output data from the MEC server. For many applications, the size of the computation output data is much smaller than that of the input data, and the download data rate is also much higher than the upload rate. Therefore, we do not consider the delay and energy consumption during this step [2]–[4], [6]–[9].

D. Problem Formulation

Given the computation offloading decision vector $\boldsymbol{x}$, the energy consumption and computation delay for the n-th MU during slot t can be computed, respectively, as

$$E_{n,t} = \mathbf{1}(x_{n,t} = 0) E_{n,t}^l + \mathbf{1}(x_{n,t} = 1) E_{n,t}^x, \tag{10}$$

$$T_{n,t} = \mathbf{1}(x_{n,t} = 0) T_{n,t}^l + \mathbf{1}(x_{n,t} = 1)(T_{n,t}^x + T_{n,t}^p), \tag{11}$$

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

Define the overall cost of all MUs in the MEC system as

$$C_t = \sum\nolimits_{n=1}^{N} E_{n,t} + \sum\nolimits_{n=1}^{N} \omega_n T_{n,t}, \tag{12}$$

where the weight coefficient ω_n (in W/sec) controls the tradeoff between energy and delay for the n-th MU. Different MUs might have different delay requirements depending on the task. For MUs dealing with time-sensitive tasks, ω_n can be set to larger values to prioritize faster execution.

The objective of this paper is to minimize the long-term average cost of the MEC system, which is defined as

$$\bar{C} = \mathbb{E}\left[\lim_{|\mathcal{T}| \to \infty} \frac{1}{|\mathcal{T}|} \sum\nolimits_{t \in \mathcal{T}} C_t \right], \tag{13}$$

where $\mathbb{E}(\cdot)$ denotes mathematical expectation. The optimization problem is formulated as follows.

$$\begin{aligned}
\mathbf{P1}: \min_{\boldsymbol{x}, \boldsymbol{p}, \boldsymbol{f}} \quad & \bar{C} \\
\text{s.t.} \quad & \text{(C1)} \quad x_{n,t} \in \{0, 1\}, \ \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T} \\
& \text{(C2)} \quad p_{n,t} \le P_n^{\max}, \ \forall n \in \{n : x_{n,t} = 1\}, \ \forall t \in \mathcal{T} \\
& \text{(C3)} \quad \sum\nolimits_{n=1}^{N} \mathbf{1}(x_{n,t} = 1) f_{n,t} \le F, \ \forall t \in \mathcal{T} \\
& \text{(C4)} \quad T_{n,t} \le T_s, \ \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}
\end{aligned}$$

Here, (C3) represents the constraint that the total amount of allocated resources can not exceed the total computational resource of the MEC server, and (C4) represents the constraint that each MU must execute its task either locally or in the MEC server within one time slot.

The optimal solution of **P1** requires knowledge of the statistical distribution of the channel condition and task requests, which are in general not available in a practical system. Moreover, **P1** is a mixed-integer nonlinear programming and it is very challenging to solve even if the statistical information is available. One possible way to overcome these challenges is to adopt an online approach that can efficiently make the decisions regarding computation offloading and resource allocation in real-time by learning from past observations. Hence, instead of applying conventional optimization methods to solve the NP-hard problem **P1**, we propose a DRL-based method to find the optimal $\boldsymbol{x}, \boldsymbol{p}$, and $\boldsymbol{f}$.

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

DRL is regarded as a combination of reinforcement learning (RL) and DNN. We first formulate **P1** in the RL framework, and then present the proposed DRL-based solution.

A. RL Framework

Generally, the RL framework is well-suited for solving complicated decision-making problems in real-time [16]. The framework consists of three key elements, state, action, and reward. An RL agent interacts with the environment in discrete time domain. At each time step t, the agent's behavior is defined by a policy μ, which maps states to actions $\mu : s_t \to a_t$. After the RL agent selects an action a_t according to the policy μ, the environment returns a scalar reward r_t and makes a transition from state s_t to s_{t+1}. The action-value function $Q^\mu(\boldsymbol{s}, \boldsymbol{a})$ (also known as Q-function) represents the expected infinite-horizon discounted accumulative reward under the policy μ with initial state $\boldsymbol{s}$ and initial action $\boldsymbol{a}$:

$$Q^\mu(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\left[\lim_{|\mathcal{T}| \to \infty} \sum\nolimits_{t \in \mathcal{T}} \gamma^t r_t | \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_0 = \boldsymbol{a} \right], \tag{14}$$

where $\gamma \in [0, 1]$ is the discount factor. The goal of the RL agent is to learn the optimal policy μ^* such that

$$\mu^*(\boldsymbol{s}) = \operatorname*{argmax}_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a}), \tag{15}$$

where $Q^*(\boldsymbol{s}, \boldsymbol{a})$ is the optimal Q-function.

To interpret problem **P1** in the RL framework, we define the key elements according to the system model as follows.

1) State: The state of a system is a set of parameters that can be used to describe a system. Hence, the system state at an arbitrary time slot t is defined as

$$\boldsymbol{s}_t \triangleq \{\boldsymbol{k}_t, \boldsymbol{H}_t\}, \tag{16}$$

where the task request vector $\boldsymbol{k}_t$ and the channel matrix $\boldsymbol{H}_t$ determine the stochasticity of system. At the start of each slot, $\boldsymbol{k}_t$ is known to the system, and the channel reciprocity can be used to estimate $\boldsymbol{H}_t$ for the upcoming uplink transmission [9].

2) Action: Based on the observed state $\boldsymbol{s}_t$, the RL agent will select actions $\boldsymbol{a}_t$ based on the decision variables in **P1** as

$$\boldsymbol{a}_t \triangleq \{\boldsymbol{x}_t, \boldsymbol{p}_t, \boldsymbol{f}_t\}. \tag{17}$$

which includes the computation offloading decision vector $\boldsymbol{x}_t$, the transmission power vector $\boldsymbol{p}_t$, and the MEC computational resource allocation vector $\boldsymbol{f}_t$ for each slot $t \in \mathcal{T}$.

3) Reward: Given a particular state $\boldsymbol{s}_t$ and an action $\boldsymbol{a}_t$ at time slot t, it is evident that the overall system cost C_t in (12) can be expressed by the cost function C, which maps the state-action pair to a scalar reward r_t such that

$$r_t \triangleq -C(\boldsymbol{s}_t, \boldsymbol{a}_t) = -C_t. \tag{18}$$

Please note that although RL algorithms maximize the expected discounted long-term reward, these algorithms can also approximate the true expected long-term undiscounted reward when $\gamma \to 1$ [17]. So, the average system cost in (13) is minimized by applying the policy learned via the RL agent.

It is difficult to obtain the exact solution of the RL problem in high-dimensional state and action spaces by directly maximizing the Q-function. We propose to tackle this challenge by obtaining an approximate solution of the RL problem by using DRL with DDPG. Details are given in the next subsection.

B. DRL with DDPG

A feasible method to solve the RL problem is the well-known Q-learning algorithm [16], which solves the optimal Q-function through a value iteration update approach as

$$\begin{aligned} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \leftarrow\ & Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \\ & + \alpha \left[r_t + \gamma \max_{\boldsymbol{a}_{t+1}} Q(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \right], \end{aligned} \tag{19}$$

where α is the learning rate. However, as the dimensions of the state space and action space increase, the complexity of solving (19) grows exponentially. DQN provides an efficient method to address this issue [10]. DQN exploits the architecture of DNN in order to approximate the Q function with a finite number of parameters in the neural network. However, DQN can only handle discrete and low-dimensional action spaces, because finding the optimal value according to (15) is relatively simple with low-dimensional spaces. For problems with continuous action and state spaces like **P1**, the action and state spaces have to be discretized before applying DQN. The complexity grows exponentially with discretization levels. The discretization in DQN also causes loss of precision.

We propose to address this challenge by applying DDPG [11], which is appropriate for problems with continuous action and state spaces. In DDPG, an actor-critic approach is adopted by using two separate DNNs, where the actor network $\mu(\boldsymbol{s}|\boldsymbol{\theta}^{\mu})$ approximates the policy function μ, and the critic network $Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^{Q})$ approximates the Q-function. Here, $\boldsymbol{\theta}^{\mu}$ and $\boldsymbol{\theta}^{Q}$ are the parameters of the actor and critic networks, respectively.

In the proposed DDPG framework, a four-layer fully connected neural network with two hidden layers is considered for both the actor and critic networks. The dimensions of two hidden layers are $8N$ and $4N$, respectively. The neural networks use the ReLu as the activation function for all hidden layers, while the final output layer of the actor uses a sigmoid layer to bound the actions. Ornstein-Uhlenbeck process [18] is adopted to generate random noise $\Delta\mu$ for action exploration, while the adaptive moment estimation (Adam) method [19] is used for updating the neural network parameters. Details of the proposed solution are described in Algorithm 1.

IV. Simulation Results

Simulation results are presented in this section to demonstrate the performance of the proposed algorithm with DDPG. Unless specified otherwise, the default settings of the MEC system are set as follows: the number of MUs $N = 6$, the number of antennas in BS $M = 8$, the coverage radius of BS $d_m = 50$ m, the channel bandwidth $W = 10$ MHz, the computational resource of MEC server $F = 5$ GHz, the CPU frequency of each MU $f_n^l = 1$ GHz, and the duration of time slot $T_s = 1$s.

At the beginning of every episode, the channel vector of each MU is initialized as $\boldsymbol{h}_{n,0} \sim \mathcal{CN}(\boldsymbol{0}, h_0(d_0/d_n)^{\beta}\boldsymbol{I}_M)$, where $h_= - 30$ dB, $d_0 = 1$ m, the path-loss exponent $\beta = 3$ [9], d_n (in meters) denotes the distance from MU n to the BS. In each episode, the locations of MUs are randomly set such that they are uniformly scattered throughout the coverage region, and the locations are independent in different episodes. The channel vectors $\boldsymbol{h}_{n,t}$, $\forall n \in \mathcal{N}$ are updated according to (2), where the channel correlation coefficient $\rho_n = 0.95$ and the error vector $\boldsymbol{e}_t \sim \mathcal{CN}(\boldsymbol{0}, h_0(d_0/d_m)^{\beta}\boldsymbol{I}_M)$. The MU's maximum allowed transmission power $P_n^{\max} = 2$ W, $\forall n \in \mathcal{N}$, and the background noise power is $\sigma^2 = 10^{-9}$ W [9]. The energy-delay tradeoff parameters are $\omega_n = 1$ for all MUs.

There are $K = 4$ computation tasks. We assume the data sizes of the computation tasks b_k (in MB) is uniformly distributed between $[50, 100]$ and the number of CPU cycles required to complete the tasks d_k (in Gigacycles) is uniformly distributed between $[0.1, 0.5]$. Moreover, the popularity profile $\boldsymbol{\phi}_t$ is modeled via a three-state Markov chain, represented by three different popularity profiles $\boldsymbol{\phi}^{(1)}, \boldsymbol{\phi}^{(2)}$, and $\boldsymbol{\phi}^{(3)}$ [20]. These profiles are modeled by Zipf distributions with parameters $\eta_1 = 1, \eta_2 = 1.2$, and $\eta_3 = 1.5$, respectively. So, at each time slot t, the popularity profile $\boldsymbol{\phi}_t$ will follow one of the three states and each task $k \in \mathcal{K}$ will be assigned popularity ranks $z_{k,t}$ randomly. The Markov transition probabilities are

Algorithm 1 Proposed Solution using DDPG

Input: System model parameters, number of episodes $K_{\max}$, number of time steps in each episode $T_{\max}$, empty replay buffer $\mathcal{R}$, mini-batch size B, update rate for target networks τ, learning rate for critic network α^Q and actor network α^μ.

Initialization:

Initialize actor network $\mu(\boldsymbol{s}|\boldsymbol{\theta}^\mu)$ and critic network $Q(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}^Q)$ with random weights $\boldsymbol{\theta}^\mu$ and $\boldsymbol{\theta}^Q$, respectively, drawn from a uniform distribution $\mathcal{U}[-3\times10^{-3}, 3\times10^{-3}]$.

Initialize associated target networks μ' and Q' with weights $\boldsymbol{\theta}^{\mu'} \leftarrow \boldsymbol{\theta}^\mu, \boldsymbol{\theta}^{Q'} \leftarrow \boldsymbol{\theta}^Q$.

for each episode $k = 1, 2, \cdots, K_{\max}$ **do**

Randomly generate an initial state $\boldsymbol{s}_1$

for each episode $t = 1, 2, \cdots, T_{\max}$ **do**

Determine the decision vectors by selecting an action $\boldsymbol{a}_t = \mu(\boldsymbol{s}_t|\boldsymbol{\theta}^\mu) + \Delta\mu$ using the current policy μ and exploration noise $\Delta\mu$, which is generated by following the Ornstein-Uhlenbeck process [18].

Execute action $\boldsymbol{a}_t$ and observe the reward $r_t = -C(\boldsymbol{s}_t, \boldsymbol{a}_t) = -C_t$ and the new state $\boldsymbol{s}_{t+1}$.

Save the transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ into the replay buffer $\mathcal{R}$.

Randomly sample a mini-batch of B transitions $\{(\boldsymbol{s}_i^{(b)}, \boldsymbol{a}_i^{(b)}, r_i^{(b)}, \boldsymbol{s}_{i+1}^{(b)})\}_{b=1}^B$ from $\mathcal{R}$.

Update the critic network $Q(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}^Q)$ by one-step gradient descent as $\boldsymbol{\theta}^Q \leftarrow \boldsymbol{\theta}^Q - \alpha^Q \nabla_{\boldsymbol{\theta}^Q} L^Q$, where the loss L^Q is

$$L^Q = \frac{1}{B}\sum_{b=1}^{B}\big[r_i^{(b)} + \gamma Q'\big(\boldsymbol{s}_{i+1}^{(b)}, \mu'(\boldsymbol{s}_{i+1}^{(b)}|\boldsymbol{\theta}^{\mu'})|\boldsymbol{\theta}^{Q'}\big) - Q(\boldsymbol{s}_i^{(b)}, \boldsymbol{a}_i^{(b)}|\boldsymbol{\theta}^Q)\big]^2. \quad (20)$$

Update the actor network $\mu(\boldsymbol{s},\boldsymbol{a}|\boldsymbol{\theta}^Q)$ by using one-step sampled policy gradient ascent as $\boldsymbol{\theta}^\mu \leftarrow \boldsymbol{\theta}^\mu + \alpha^\mu \nabla_{\boldsymbol{\theta}^\mu} J^\mu$, where $J^\mu \triangleq \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}} Q^\mu(\boldsymbol{s},\boldsymbol{a})$, and

$$\nabla_{\boldsymbol{\theta}^\mu} J^\mu \approx \frac{1}{B}\sum_{b=1}^{B} \nabla_{\boldsymbol{a}} Q(\boldsymbol{s}_i^{(b)}, \boldsymbol{a}|\boldsymbol{\theta}^Q)\,|_{\boldsymbol{a}=\boldsymbol{a}_i^{(b)}} \times \nabla_{\boldsymbol{\theta}^\mu}\mu(\boldsymbol{s}_i^{(b)}|\boldsymbol{\theta}^\mu). \quad (21)$$

Update the target networks:

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

end for

end for

Output: Optimal policy μ^*.

given by the transition matrix

$$\boldsymbol{\tau} \triangleq \begin{bmatrix} \tau_{1,1} & \tau_{1,2} & \tau_{1,3} \\ \tau_{2,1} & \tau_{2,2} & \tau_{2,3} \\ \tau_{3,1} & \tau_{3,2} & \tau_{3,3} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.1 & 0.6 & 0.3 \\ 0.25 & 0.35 & 0.4 \end{bmatrix}, \quad (22)$$

where $\tau_{i,j}$ indicates transition probability from state i to j, for $i, j \in \{1, 2, 3\}$. Please note that these states are different from the system states defined in our problem formulation.

For learning the neural network parameters, we set the hyper-parameters as follows: the number of training episodes $K_{\max} = 1500$, the number of steps in each episode $T_{\max} = 100$, the experience replay buffer size $|\mathcal{R}| = 50000$, the mini-batch size $B = 128$, the soft update rate for the target networks $\tau = 10^{-3}$, and the learning rate for actor network $\alpha^\mu = 10^{-4}$ and critic network $\alpha^Q = 10^{-3}$.

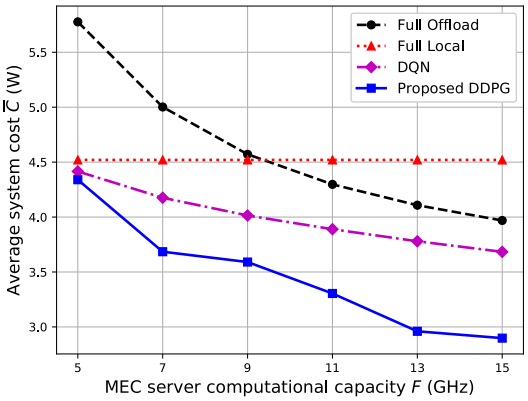

Fig. 1. Average system cost v.s. the capacity of MEC server.

To evaluate the performance of policy μ^* learned by the proposed DDPG-based solution, testing results are averaged from 1000 episodes, with each episode consisting of 100 steps. Results obtained from the proposed DDPG-based algorithm are compared to three baseline strategies described as follows.
(1) *Full Local*: all MUs execute their tasks locally.
(2) *Full Offload*: all MUs offload their tasks to the MEC server. All MUs transmit with the maximum power available and the computational resource F is distributed uniformly to each MU.
(3) *DQN-based Solution*: DQN [10] can only be implemented on systems with discrete state and action spaces. The support spaces for $\boldsymbol{p}$ and $\boldsymbol{f}$ are both discretized uniformly into finite L levels each. Therefore, the size of the action space is $(2L^2)^N$ for N MUs. We arbitrarily set L = 3 and maintain the same neural network architecture as mentioned in Section III-B. In addition, ϵ-greedy exploration method is adopted for exploring the actions during network training with $\epsilon = 0.01$.

Fig. 1 shows the average system cost as a function of the resource capacity of the MEC server (F) with different algorithms. Since the MUs do not utilize the computational resource of the MEC server in the 'Full Local' approach, the performance remains constant regardless of F. The performance of all the other algorithms improves as F increases due to the extra computational resources from MEC. The proposed DDPG-based algorithm achieves the best performance, followed by the DQN approach. The performance gap between the DDPG-based algorithm and DQN algorithm becomes larger as F increases. This means that the DDPG-based algorithm can better utilize the MEC resources. Although it may be possible to get better results by DQN approach with a larger number of discrete levels L, it will yield a very high-dimensional action space with prohibitively high complexity.

Table I shows the effects of the number of MUs (N) on

TABLE I
EFFECT OF NUMBER OF MUS (N) ON AVERAGE SYSTEM COST $\bar{C}$ (W)

N	Full Offload	Full Local	DQN	Proposed DDPG
4	2.42	1.99	1.74	**1.52**
5	3.19	2.39	2.04	**1.70**
6	4.05	2.78	2.63	**2.52**

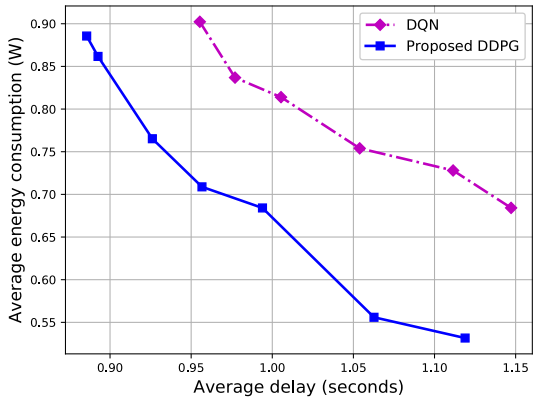

Fig. 2. Tradeoff between average energy consumption and average delay with tradeoff parameter $\omega_n = \omega \in [1, 2]\ \forall n \in \mathcal{N}$.

the system cost averaged over time. As expected, the time-averaged system cost becomes larger as N increases. The proposed DDPG-based solution has the best performance, followed by DQN, full local, and full offload, respectively. The full offload approach has the worst performance because the computation capacity at the MEC ($F = 5$ GHz) is not sufficient to support the computation demands from all MUs, especially when N becomes larger. The DDPG-based algorithm achieves 4%-16.7% performance improvement compared to its DQN-based counterpart.

Fig. 2 illustrates the tradeoff relationship between the average energy consumption and the average delay of the system. Different tradeoff points are obtained by varying the values of ω_n for all $n \in \mathcal{N}$ MUs. It is worth noting that we consider the time-average not the ensemble average across all the MUs. With larger ω_n, the average delay experienced by all the MUS can be decreased at the cost of higher energy consumption. Moreover, the policy learned by DDPG demonstrates better tradeoff performance compared to the policy learned by DQN. Since both the 'Full Offload' and 'Full Local' approaches have fixed policies, ω_n doesn't affect the decisions. Hence, these approaches do not demonstrate tradeoff relationships between energy consumption and delay.

V. CONCLUSION

We have studied the problem of dynamic computation offloading and resource allocation in MEC systems with stochastic wireless channel conditions, where computation-intensive tasks at MUs can be executed either locally or remotely in an MEC server. The problem was formulated to minimize the long term average of the weighted sum of energy consumption and delay of all the MUs. A DDPG-based method has been proposed to solve the problem. Simulation results have demonstrated that the proposed algorithm outperforms existing approaches such as DQN.

REFERENCES

[1] 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, Mar. 2017.

[2] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, Sep. 2017.

[3] 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.

[4] J. Zhang, X. Hu, Z. Ning, E. C.-H. Ngai, L. Zhou, J. Wei, J. Cheng, and B. Hu, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2017.

[5] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing (Early Access)*, Jul. 2019.

[6] 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.

[7] 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.

[8] S. Nath, Y. Li, J. Wu, and P. Fan, "Multi-user multi-channel computation offloading and resource allocation for mobile edge computing," in *Proc. IEEE International Conference on Communications (ICC)*, Dublin, Ireland, Jun. 2020.

[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] 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.

[11] 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.

[12] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[13] H. A. Suraweera, T. A. Tsiftsis, G. K. Karagiannidis, and A. Nallanathan, "Effect of feedback delay on amplify-and-forward relay networks with beamforming," *IEEE Transactions on Vehicular Technology*, vol. 60, no. 3, pp. 1265–1271, Mar. 2011.

[14] H. Q. Ngo, E. G. Larsson, and T. L. Marzetta, "Energy and spectral efficiency of very large multiuser mimo systems," *IEEE Transactions on Communications*, vol. 61, no. 4, pp. 1436–1449, Apr. 2013.

[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] D. Adelman and A. J. Mersereau, "Relaxations of weakly coupled stochastic dynamic programs," *Operations Research*, vol. 56, no. 3, pp. 712–727, 2008.

[18] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal dynamic proactive caching via reinforcement learning," in *19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. Kalamata, Greece: IEEE, 2018.