Joint Scheduling and Beamforming Design in Traffic-Aware RIS Aided MEC Network

Aichen Li[1], Yang Liu[1], Qingqing Wu[2], Qingjiang Shi[3], and Jun Zhao[4]
1: School of Information and Communication Engineering, Dalian University of Technology
2: Department of Electrical and Computer Engineering, University of Macau, Macau, China
3: School of Software Eng., Tongji Univ. Shanghai, Shenzhen Research Institute of Big Data, Shenzhen, China
4: School of Computer Science and Engineering, Nanyang Technological University
Emails: {lacz@mail.dlut.edu.cn, yangliu_613@dlut.edu.cn, qingqingwu@um.edu.mo, shiqj@tongji.edu.cn, junzhao@ntu.edu.sg}

***Abstract*—This paper considers exploiting reconfigurable intelligent surface (RIS) in a multi-user multi-access mobile computing (MEC) system. To precisely model the traffic behaviors of the tasks, we consider queuing procedures at both mobile users and the cloud server. This consideration leads to concatenation of queues and makes our problem much more difficult than existing literature since analytic expression of the objective is impossible. Our goal is to jointly optimize the task scheduling, RIS configuration and computation resource allocation to improve the MEC network's overall computation rate in the long run while maintaining the stability of the queues. To tackle this highly challenging stochastic problem, we develop a Lyapunov guided deep reinforcement learning (DRL) solution, which borrows the Lyapunov optimization theory to stabilize queues and at the same time can effectively optimize the intractable objective in an online manner. Numerical results will be presented to verify the effectiveness of our solution and the performance boost due to the deployment of RIS.**

***Index Terms*—Multi-access edge computing (MEC), reconfigurable intelligent surface (RIS), deep reinforcement learning (DRL), Lyapunov optimization.**

I. Introduction

Nowadays the Internet of Things (IoT) technology is making rapid progress. In future, ubiquitous IoT devices will connect everything and various novel applications will deeply change our society. At the same time, one major challenge of the developing IoT technology is its exponentially growing demand for computation. Constrained by the battery capacity and/or hardware cost, IoT devices generally have limited computation capability. To overcome this difficulty, the so-called multi-access mobile computing (MEC) paradigm has emerged [1]. The main idea of MEC is simple— offloading the computation tasks at the mobile devices to cloud servers lying at the edge of the radio access network (RAN) and exploiting the computation resources therein.

In fact, there have been a multitude of researches on the MEC technology. For instance, the authors in [2] consider cost and delay tradeoff of the offloading scheme in a multi-user MEC system via introducing competition between mobile users. The work [3] considers energy and delay tradeoff in a single user MEC system equipped with multiple cores. The work [4] considers users' mobility and energy harvesting when designing the offloading scheme of MEC network. Additionally, the authors of [5] consider energy transfer in a MEC system, where the offloading procedure should satisfy the energy supply and delay requirement imposed by the IoT devices.

Recently, the reconfigurable intelligent surface (RIS) technology has been envisioned as a potential key technology for 6G due to its beamforming capability and high energy efficiency [6]. The deployment of RIS in MEC system has also been considered in [5], [7]-[8], where computation tasks' assignments and RIS configurations are jointly designed.

In this paper, we deploy RIS device in a MEC network to boost the communication throughput, which can improve the offloading efficiency. Specifically, we jointly optimize the task scheduling, RIS configuration and computation resource allocation to improve the overall computation rate of the MEC network. Unlike the existing works [9]-[10], where only local queuing procedures at mobile users are present, we consider the concatenations of queues of mobile users and MEC server. This consideration makes our problem very challenging since analytic expression of computation rate is impossible. We combine the Lyapunov optimization and deep reinforcement learning (DRL) method to develop an online optimization solution, which can effectively stabilize the queues and maximize the system's computation rate in the long run.

II. System Model

This paper considers a multi-user MEC system aided by RIS, as shown in Fig. 1. K single-antenna mobile users (MUs) are served by a multi-antenna edge cloud server equipped with a multitude of CPUs. For each MU, computation tasks keep emerging due to its ongoing applications. Considering its battery life and limited computation capability, each MU offloads a portion of the computation tasks onto the MEC server and exploits the computation resource therein. Specifically, each MU has a build-in CPU and a communication circuitry, the former executes the incoming computation tasks and the latter transmits the remaining tasks to the cloud server. The edge cloud server will feedback the results once the offloaded tasks have been accomplished. Besides, the system deploys a RIS device with N reflecting elements to boost the communication throughput. Define the index set of the K MUs as $\mathcal{K} = \{1, \ldots, K\}$. Besides, denote $\mathbf{H}_{\rm br}$, $\mathbf{h}_{{\rm ru},k}$ and $\mathbf{h}_{{\rm d},k}$ as the wireless channels between the server and the RIS, the

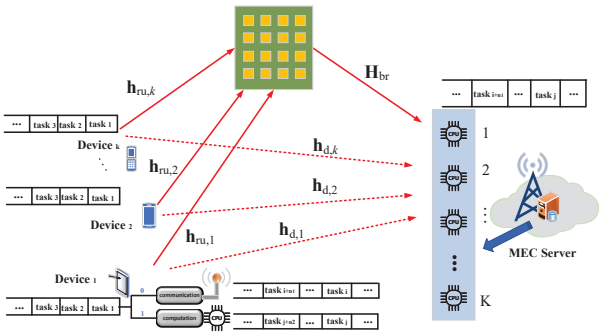


Fig. 1. RIS-aided MEC system with queues.

RIS and the k-th MU and the server between the k-th MU, respectively.

A. Offloading and Queuing Procedures

The entire MEC system operates in a time-slot-based manner. That is, the whole time axes is divided into consecutive non-overlapping equal-length intervals, with each interval having a duration of $\triangle$ and being indexed by $t = 1, 2, \cdots$. At the beginning of t-th slot, $m_k(t)$ new computation tasks are generated from the MU-k's application layer. The number of arriving tasks $\{m_k(t)\}_{t=1}^{\infty}$ can be regarded as a random process and its underlying distribution is related to the applications being experienced by MU-k. We do not impose any restrictions on the task generating process distribution, which is generally assumed to be Poisson in the existing literature, e.g. [11].

Each emerging task is characterized by a 2D tuple (D_k, C_k) with D_k indicating the amount of data (in bits) of the task and C_k indicating the amount of CPU cycles per bit required to finish its computation. Each computation task should be executed either locally at the MU side or remotely at the MEC server side via offloading. Therefore, all the incoming computation tasks are divided into two groups—one group is forwarded to MU-k's local CPU and the other group is fed to its communication circuitry. To effectively balance the workloads, MU-k offloads $\lfloor m_k(t)v_k \rfloor$ incoming tasks to the MEC server and assigns the remaining $m_k(t) - \lfloor m_k(t)v_k \rfloor$ tasks to the local CPU, with v_k being the adjustable offloading factor. Here we assume that each task is indivisible and therefore the floor operation $\lfloor \cdot \rfloor$ guarantees that each task shall be processed as a whole either locally or remotely. Each group of scheduled tasks forms a queue and will be consecutively transmitted or computed in a first-in-first-out (FIFO) manner, as shown in Fig. 1.

• *Local Computation Queue*: Let $Q_k^l(t)$ denote the amount of CPU cycles needed to finish the backlogged tasks in the queue for MU-k's local computing at the end of the t-th slot. The dynamics of $Q_k^l(t)$ is given as

$$Q_k^l(t) = \max\left\{Q_k^l(t-1) + (m_k(t) - \lfloor m_k(t)v_k \rfloor) D_k C_k - f_k \triangle, 0\right\}, \quad (1)$$

where f_k is MU-k's CPU frequency (in cycles/s).

• *Transmission Queue*: The group of tasks to be offloaded also forms a queue. Denote the amount of data in this communication queue as $Q_k^T(t)$ (in bits). Then the dynamics of $Q_k^T(t)$ is described as

$$Q_k^T(t) = \max\left\{Q_k^T(t-1) + \lfloor m_k(t)v_k \rfloor D_k - R_k(t)\triangle, 0\right\}, \quad (2)$$

where $R_k(t)$ is the MU-k's communication rate at the t-th slot. Utilizing orthogonal multiple access (OMA) scheme, e.g. frequency division multiple access (FDMA) or code division multiple access (CDMA), MU-k's transmission rate can be expressed as

$$R_k(t) = B \log_2\left(1 + \frac{p_k ||\mathbf{h}_k||_2^2}{N_0}\right), \forall k \in \mathcal{K}, t \in \mathcal{T}. \quad (3)$$

where p_k is the transmit power, $\mathbf{h}_k$ denotes the effective channel from MU-k and B is the bandwidth of sub-channel dedicated for each MU.

The effective channel $\mathbf{h}_k$ is expressed as $\mathbf{h}_k^H = \mathbf{h}_{\mathrm{ru},k}^H \mathbf{\Phi} \mathbf{H}_{\mathrm{br}} + \mathbf{h}_{\mathrm{d},k}^H$, with $\mathbf{\Phi} = \mathrm{diag}(e^{j\theta_1}, \cdots, e^{j\theta_N})$ denoting the reflection coefficients of the RIS.

• *MEC Server's Queue*: The MEC cloud server assigns one dedicated CPU to execute each MU's tasks. The offloaded tasks from MU-k forms a queue and the amount of cycles required to finish the queuing task at the end of t-th slot is denoted as $Q_k^o(t)$. The dynamics of this queue can be depicted as

$$Q_k^o(t) = \max\left\{\left(Q_k^o(t-1) - g_k \triangle + C_k \min\left\{Q_k^T(t-1) + \lfloor m_k(t)v_k \rfloor D_k, R_k(t)\triangle\right\}\right), 0\right\}, \quad (4)$$

where g_k is the frequency of the CPU serving MU-k. In (4), the "min" term considers the fact that the amount of offloaded data could not exceed that in MU-k's communication queue in the t-th slot.

B. Computation Rate of the System

We aim to improve the whole MEC system's computation rate, which is defined as the total cycles executed by the CPUs of all MUs and the MEC server within unit time. In the following, we will analyze the computation rates contributed by various components of the system.

• Local Computation Rates

Firstly, we consider the MU-k's CPU. At the beginning of t-th slot, the workloads in the queue for MU-k's local computing is $Q_k^l(t-1) + (m_k(t) - \lfloor m_k(t)v_k \rfloor) D_k C_k$ cycles. When provisioned with sufficiently high frequency, all the queuing tasks would be completed before the end of the t-th time slot and the local CPU would go into idle state. Otherwise, the CPU would keep working during the whole slot. Therefore, within the t-th slot the working time $\tau_k^l(t)$ of MU-k in the t-th slot is given as (5) and hence its contribution to the system's computation rate is $f_k \tau_k^l(t)$.

$$\tau_k^l(t) = \min\left\{\frac{Q_k^l(t-1) + (m_k(t) - \lfloor m_k(t)v_k \rfloor) D_k C_k}{f_k}, \triangle\right\}. \quad (5)$$

• MEC Server's Computation Rate

Next we proceed to discuss the computation rates conducted by the MEC server. At the beginning of the t-th slot, MU-k keeps transmitting the tasks in its communication queue to the MEC server. At the same time, the remaining tasks in the queue for MU-k at the MEC server is $Q_k^o(t-1)$. The CPU serving MU-k will firstly try to finish all the residual workloads $Q_k^o(t-1)$ in the queue. Assuming that the system's first slot starts from 0 in time, the instant completing all these residual tasks is $s_{k,0}(t) = Q_k^o(t-1)/g_k + t\Delta$. After that, provided there still exists time left before the end of the current slot, the CPU will proceed to process the tasks newly uploaded from MU-k within this slot. Therefore, depending on $s_{k,0}(t)$,

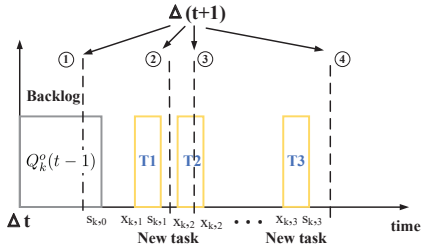


Fig. 2. The state of task queue and computation moment.

two possible cases could occur and they are elaborated in the following:

1) Case-1: $\triangle(t+1) \leq s_{k,0}(t)$. In this case, the residual workload $Q_k^o(t-1)$ cannot be finished within the current slot, as illustrated by the case ① in Fig. 2. Therefore, the CPU serving MU-k will keep working until the end of the current slot. In this case, the working time $\tau_k^o(t)$ can be readily determined as

$$\tau_k^o(t) = \triangle. \tag{6}$$

2) Case-2: $s_{k,0}(t) < \triangle(t+1)$. After finishing $Q_k^o(t-1)$, the CPU will continue to compute newly offloaded tasks. At this time, the computation rate calculation can be complicated since there may be waiting time before each transmitted task from MU-k. For instance, $s_{k,0}(t)$ may be earlier than the arrival of the first offloaded task from MU-k. Therefore the CPU will experience an idle interval due to waiting for the next task, which undermines the computation rate. Similarly, idle intervals may also occur for the subsequent newly arriving offloaded tasks, as illustrated by case 2, 3,and 4 in Fig. 2. Next, we will iteratively analyze and calculate idle and working time for the CPU. To simplify notations, we denote the i-th offloaded task from the MU-k to the MEC server within the t-th slot as $T_{k,i}(t)$, with $i = 1, 2, ..., N_k(t)$, and $N_k(t)$ being the maximal number of completed tasks newly offloaded to the MEC server within the t-th slot, which can be formally determined as

$$N_k(t) = \underset{\{i=1,2,\cdots\}}{\arg\max} \{s_{k,i}(t) \leq \triangle(t+1)\}. \tag{7}$$

The arriving time of $T_{k,1}(t)$ can be determined as follows:

$$u_{k,1}(t) = \begin{cases} \max\left\{\frac{Q_k^T(t-1)}{D_k} - \lfloor\frac{Q_k^T(t-1)}{D_k}\rfloor, 1\right\}\frac{D_k}{R_k(t)} + t\Delta, \\ \qquad\qquad \text{if } Q_k^T(t-1) \geq D_k, \\ \frac{Q_k^T(t-1)}{R_k(t)} + t\Delta, \qquad \text{if } 0 < Q_k^T(t-1) < D_k, \\ (t+1)\Delta, \qquad\quad \text{if } Q_{k-1}^T(t) = 0. \end{cases} \tag{8}$$

where the third case in the above corresponds to the case where the communication queue of MU-k in the t-th slot is empty and therefore no fresh task will be offloaded to the MEC server indeed. Denote the moment that $T_{k,1}(t)$ starts being executed by the MEC's CPU as $x_{k,1}(t)$, which can be determined as

$$x_{k,1}(t) = \max\{s_{k,0}(t), u_{k,1}(t)\}. \tag{9}$$

Assume that $T_{k,1}(t)$ is finished at the time instant $s_{k,1}(t)$ and it can be expressed explicitly as

$$s_{k,1}(t) = x_{k,1}(t) + D_k C_k / g_k. \tag{10}$$

Depending on the order in time of $u_{k,1}(t)$ and $s_{k,0}(t)$, there may be an idle interval for the CPU before executing $T_{k,1}(t)$. That is, we always have $x_{k,1}(t) \geq s_{k,0}(t)$ and the CPU's waiting time for $T_{k,1}(t)$ is $x_{k,1}(t) - s_{k,0}(t)$ when $x_{k,1}(t) > s_{k,0}(t)$. Denote the idle time before executing $T_{k,1}(t)$ as $\delta_{k,1}(t)$, then we have

$$\delta_{k,1}(t) = x_{k,1}(t) - s_{k,0}(t). \tag{11}$$

For $i = 2, 3, ..., N_k(t)$, we define $u_{k,i}(t), x_{k,i}(t), s_{k,i}(t)$ and $\delta_{k,i}(t)$ as the time instant when $T_{k,i}(t)$'s arrives at the MEC server, the instant when $T_{k,i}(t)$ starts to be processed, the instant when $T_{k,i}(t)$ is completed and the waiting time for $T_{k,i}(t)$, respectively. Then following similar arguments as above, the quantities $\{u_{k,i}(t), x_{k,i}(t), s_{k,i}(t), \delta_{k,i}(t)\}$ can be recursively determined as follows:

$$\begin{cases} u_{k,i}(t) = u_{k,i-1}(t) + \frac{D_k}{R_k(t)}, \\ x_{k,i}(t) = \max\{s_{k,i-1}(t), u_{k,i}(t)\}, \quad i = 2, 3, \cdots, N_k(t). \\ s_{k,i}(t) = x_{k,i}(t) + \frac{D_k C_k}{g_k}, \\ \delta_{k,i}(t) = x_{k,i}(t) - s_{k,i-1}(t). \end{cases} \tag{12}$$

Recall that $T_{k,N_k(t)}$ is the last newly offloaded task which has been finished within the t-th time slot. Then there may still exist an idle interval lying between $s_{k,N_k(t)}$ and $\triangle(t+1)$, as illustrated by the sub-cases ②, ③, ④ in Fig.2 and we denote it as $\delta_{k,N_k(t)+1}(t)$. Specifically,② corresponds to the scenario that $T_{k,N_k(t)+1}(t)$ will not arrive before $\triangle(t+1)$, ③ means $T_{k,N_k(t)+1}(t)$ does arrive but cannot be completed before $\triangle(t+1)$, and ④ suggests that $T_{k,N_k(t)}$ is indeed the last task. Then, according to the sub-cases ②-④, $\delta_{k,N_k(t)+1}(t)$ can be evaluated as follows:

$$\begin{cases} ②: \triangle(t+1) - s_{k,N_k(t)}(t), \\ \qquad \text{if } s_{k,N_k(t)}(t) \leq \triangle(t+1) < x_{k,N_k(t)+1}(t), \\ ③: x_{k,N_k(t)+1}(t) - s_{k,N_k(t)}(t), \\ \qquad \text{if } x_{k,N_k(t)+1}(t) \leq \triangle(t+1) < s_{k,N_k(t)+1}(t), \\ ④: \triangle(t+1) - s_{k,N_k(t)}(t), \\ \qquad \text{if } s_{k,N_k(t)}(t) \leq \triangle(t+1). \end{cases} \tag{13}$$

The total idle interval of the server's CPU within the t-th slot is given as

$$\delta_k^o(t) = \sum\nolimits_{j=1}^{N_k(t)+1} \delta_{k,j}(t). \tag{14}$$

and the working interval of the CPU is obtained by

$$\tau_k^o(t) = \triangle - \delta_k^o(t). \tag{15}$$

Therefore, the computation rate contributed by the server's CPU for MU-k is $g_k \tau_k^o(t)$.

III. Problem Formulation

The total computation rate of the t-th time slot of the aforementioned MEC system is given as

$$\sum\nolimits_{k=1}^{K} f_k \tau_k^l(t) + g_k \tau_k^o(t) \tag{16}$$

with $\tau_k^l(t)$ and $\tau_k^o(t)$ being determined in (5) and (15), respectively. According to the energy modeling proposed in [3], the power consumption of CPU working at frequency f can be modeled as $P = \kappa f^3$, where κ is capacitance switching coefficient. Therefore, the average power consumption of MU-k and server is given as $\kappa_k^l f_k^3 + p_k$ and $\kappa_k^o g_k^3$, respectively, with κ_k^l being the capacitance switching coefficient of MU-k and κ_k^o being that of the server. To simplify notations, we define $\mathbf{v} \triangleq [v_1, \cdots, v_K]$, $\mathbf{f} \triangleq [f_1, \cdots, f_K]$, $\mathbf{g} \triangleq [g_1, \cdots, g_K]$, $\mathbf{p} \triangleq [p_1, \cdots, p_K]$ and $\boldsymbol{\phi} \triangleq [\theta_1, \cdots, \theta_N]$. Our goal is to jointly optimize the offloading coefficients $\mathbf{v}$, the local and edge CPU frequencies $\mathbf{f}$ and $\mathbf{g}$, the transmit power $\mathbf{p}$ and the RIS phase shifts $\boldsymbol{\phi}$ to maximize system's computation rate in the long run. Mathematically, this dynamic control problem can be expressed as

$$(\mathsf{P1}): \max_{\mathbf{v},\mathbf{f},\mathbf{p},\mathbf{g},\boldsymbol{\phi}} \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{\infty}\sum_{k=1}^{K}\Big(f_k\tau_k^l(t)+g_k\tau_k^o(t)\Big) \tag{17a}$$

$$\text{s.t. } v_k \in [0,1], \forall k \in \mathcal{K}, \tag{17b}$$

$$\theta_n \in [0,2\pi], \forall n \in \mathcal{N} \triangleq \{1,\cdots,N\}, \tag{17c}$$

$$\lim_{T\to\infty} \frac{1}{T}\sum\nolimits_{t=1}^{T}\Big[Q_k^l(t)\Big] < \infty, \forall k \in \mathcal{K}, \tag{17d}$$

$$\lim_{T\to\infty} \frac{1}{T}\sum\nolimits_{t=1}^{T}\Big[Q_k^T(t)\Big] < \infty, \forall k \in \mathcal{K}, \tag{17e}$$

$$\lim_{T\to\infty} \frac{1}{T}\sum\nolimits_{t=1}^{T}[Q_k^o(t)] < \infty, \forall k \in \mathcal{K}, \tag{17f}$$

$$\mathbb{E}\Big[\kappa_k^l f_k^3 + p_k\Big] \le \tilde{e}_k^l, \forall k \in \mathcal{K}, \tag{17g}$$

$$\sum\nolimits_{k=1}^{K}\kappa_k^o g_k^3 \le \tilde{e}_k^o, \forall k \in \mathcal{K}. \tag{17h}$$

where the constraints (17d)-(17f) require that the all the queues should be stable in long term and (17g)-(17h) indicate that the average power consumption at MUs and the edge server should not exceed their power supplies $\tilde{e}_k^l$, $\tilde{e}_k^o$, respectively.

IV. Algorithm Design

A. Lyapunov Optimization

To deal with the queuing stability constraints (17d)-(17f), we adopt the Lyapunov optimization methodology to tackle (P1). To this end, define the total backlogs of the entire MEC system as $\boldsymbol{\Theta}_t \triangleq \{Q_k^l(t), Q_k^T(t), Q_k^o(t)\}_k$. The perturbed Lyapunov function $L(\boldsymbol{\Theta}_t)$ and Lyapunov drift $\triangle L(\boldsymbol{\Theta}_t)$ [9], are defined as

$$L(\boldsymbol{\Theta}_t) \triangleq \frac{1}{2}\left\{\sum_{k\in\mathcal{K}}(Q_k^l(t))^2+\sum_{k\in\mathcal{K}}(Q_k^T(t))^2+\sum_{k\in\mathcal{K}}(Q_k^o(t))^2\right\}, \tag{18}$$

$$\triangle L(\boldsymbol{\Theta}_t) \triangleq \mathbb{E}\left[L(\boldsymbol{\Theta}_{t+1}) - L(\boldsymbol{\Theta}_t)|\boldsymbol{\Theta}_t\right]. \tag{19}$$

According to the Lyapunov optimization framework, to maximize the total computation rate while stabilizing all the queues $\boldsymbol{\Theta}_t$, we turn to optimize the drift-plus-penalty of the queuing system, which is indeed the objective penalized by Lyapunov drift [9]. Specifically, for each specific slot t, the drift-plus-penalty can be expressed as

$$\Lambda(\boldsymbol{\Theta}_t) \triangleq \triangle L(\boldsymbol{\Theta}_t) - V\sum\nolimits_{k=1}^{K}\Big(f_k\tau_k^l(t)+g_k\tau_k^o(t)\Big). \tag{20}$$

where $V>0$ is a predefined "importance" weight to balance the drift and the genuine objective. It can be proved [1] that $\triangle L(\boldsymbol{\Theta}_t)$ is upper bounded by the following quantity:

$$\begin{aligned}\triangle L(\boldsymbol{\Theta}_t) \le \hat{B} &+ \sum\nolimits_{k=1}^{K}\mathbb{E}\left[Q_k^T(t)\Big(\lfloor m_k(t)v_k\rfloor D_k - R_k(t)\triangle\Big)\middle|\boldsymbol{\Theta}_t\right]\\ &+\sum\nolimits_{k=1}^{K}\mathbb{E}\left[Q_k^l(t)\Big((m_k(t)]-\lfloor m_k(t)v_k\rfloor)D_kC_k - f_k\triangle\Big)\middle|\boldsymbol{\Theta}_t\right]\\ &+\sum\nolimits_{k=1}^{K}\mathbb{E}\left[Q_k^o(t)\Big(R_k(t)C_k\triangle - g_k\triangle\Big)\middle|\boldsymbol{\Theta}_t\right].\end{aligned} \tag{21}$$

with $\hat{B}$ being some constant upper bounding the second order terms of the backlog $\boldsymbol{\Theta}_t$. According to [9], the drift-plus-penalty approach usually optimizes an upper bound of $\Lambda(\boldsymbol{\Theta}_t)$ obtained by substituting the $\triangle L(\boldsymbol{\Theta}_t)$ in (20) with its upper bound in (21).

Lyapunov optimization framework solves the original problem (P1) in an online manner with each time slot t solving the following problem

$$\begin{aligned}(\mathsf{P2}): \max_{\mathbf{v},\mathbf{f},\mathbf{p},\mathbf{g},\boldsymbol{\phi}} \sum_{k=1}^{K}\Big[V\Big(f_k\tau_k^l(t)+g_k\tau_k^o(t)\Big) + Q_k^T(t)R_k(t)\triangle\\ + Q_k^l(t)f_k\triangle + Q_k^o(t)g_k\triangle\Big]\end{aligned} \tag{22a}$$

$$\text{s.t. } v_k \in [0,1], \forall k \in \mathcal{K} \tag{22b}$$

$$\theta_n \in [0,2\pi], \forall n \in \mathcal{N}, \tag{22c}$$

$$\mathbb{E}\Big[\kappa_k^l f_k^3 + p_k\Big] \le \tilde{e}_k^l, \forall k \in \mathcal{K}, \tag{22d}$$

$$\sum\nolimits_{k=1}^{K}\kappa_k^o g_k^3 \le \tilde{e}_k^o, \forall k \in \mathcal{K}, \tag{22e}$$

So far the problem reduces to solving the instant problem (P2), which is still very challenging. The major difficulty lies in the complicated form of the computation rate that is developed recursively through the derivations (6)-(15). Due to the intractable combinatorial nature of the objective of (P2), we turn to adopt RL method to tackle it.

B. Lyapunov Guided DRL Algorithm Design

In this subsection, we incorporate the DRL method into the aforementioned Lyapunov optimization framework to tackle the problem (P1).

We need firstly reinterpret our problem in a standard RL context. Specifically, we treat the wireless channels and the queues as the environment and the scheduling control unit (which is usually the MEC server) as the agent. Other key elements are specified as follows

State Space: The environment state s_t is composed of the observations of the system status associated with time slot t, simply, we define $\mathbf{h}_t \triangleq [\mathbf{h}_1,\cdots,\mathbf{h}_K]$, $\mathbf{Q}^l(t) \triangleq [Q_1^l(t),\cdots,Q_K^l(t)]$, $\mathbf{Q}^T(t) \triangleq [Q_1^T(t),\cdots,Q_K^T(t)]$ and $\mathbf{Q}^o(t) \triangleq [Q_1^o(t),\cdots,Q_K^o(t)]$.

$$s_t = \Big\{\mathbf{h}_t, \mathbf{Q}^l(t), \mathbf{Q}^T(t), \mathbf{Q}^o(t)\Big\}. \tag{23}$$

Action Space: Action means all the configurable parameters the agent can adjust. Here the action comprises the offloading coefficients $\mathbf{v}$, the CPU frequencies $\mathbf{f}$ and $\mathbf{g}$, transmit power $\mathbf{p}$ and RIS phase shifting $\boldsymbol{\phi}$.

$$a_t = \{\mathbf{v},\mathbf{f},\mathbf{p},\mathbf{g},\boldsymbol{\phi}\}. \tag{24}$$

Policy: Policy is a mapping from state s_t to the probability of selecting all possible actions. It is usually described as the probability distribution $\pi(a_t|s_t)$ based on s_t.

Reward Function: Reward measures the obtained benefit after taking an action given a specific state. In our problem, the reward is chose as the value of (22a) given the state and actions of the current time slot.

The DRL framework incorporates deep learning into the RL procedure and progressively learns an optimal policy by interacting with the environment. We adopt the deep deterministic policy gradient (DDPG) method to update policy, which integrates the actor-critic (AC) learning process into the policy gradient (PG) updating [13]. Specifically, DDPG algorithm has four components—two evaluation networks and two target networks, as shown in Fig. 3.

The evaluation actor network produces a specific action $a_t = \mu(s_t;\boldsymbol{\theta}_\mu)$ according to the s_t with $\boldsymbol{\theta}_\mu$ being the parameters of the neutral network (NN). The evaluation critic network models the Q value $Q(s_t,a_t;\boldsymbol{\theta}_q)$, which measures the average reward associated with s_t and a_t, with $\boldsymbol{\theta}_q$ being the parameters

[1] the proof indeed follows similar arguments in [12] and is omitted here due to space limitation.

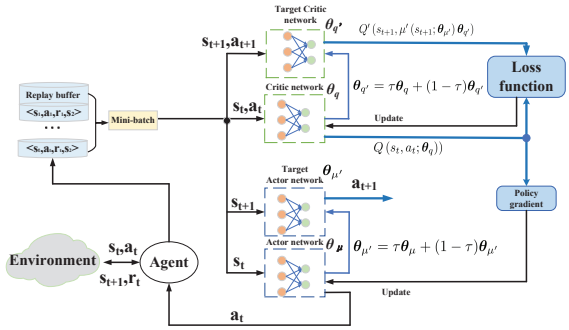


Fig. 3. The architecture of LyDRL.

of NN. Simultaneously, a target actor and a target critic network are also present and are implemented using identical architectures with their actor counterparts but with different parameters $\boldsymbol{\theta}_{q'}$ and $\boldsymbol{\theta}_{\mu'}$, respectively. Moreover, to calculate the corresponding target values, the target critic Q value is modeled as $Q'(s_{t+1}, a_{t+1}; \boldsymbol{\theta}_{q'})$, where $a_{t+1} = \mu'(s_{t+1}; \boldsymbol{\theta}_{\mu'})$. Besides, an experience replay buffer is implemented to reduce the correlation between different training samples.

The working procedure of DDPG is detailed as follows. During each episode, as illustrated in Fig. 3, the actor evaluation network takes the state s_t as input and gives out a corresponding action a_t. After a_t is taken, the associated reward r_t is obtained and the environment's state transfers from s_t to s_{t+1}. The newly obtained quantities $\{s_t, a_t, r_t, s_{t+1}\}$ will be stored in the replay buffer. The critic then samples a mini-batch of N_B experiences to calculate the target Q value y_j [13], which are defined as (25).

$$y_j = \begin{cases} r_j, & j = N_B \\ r_j + \gamma Q'\left(s_{j+1}, \mu'\left(s_{j+1}; \boldsymbol{\theta}_{\mu'}\right); \boldsymbol{\theta}_{q'}\right), & j < N_B \end{cases}. \quad (25)$$

To optimize the evaluation critic network, we calculate the difference between target Q value and Q-function given by the evaluation critic network. Then the evaluation critic network is updated by using stochastic gradient descent (SGD) method to minimize the loss function, which is defined as

$$L\left(\boldsymbol{\theta}_q\right) = \frac{1}{N_B}\sum_{t=1}^{N_B}\left(y_t - Q\left(s_t, a_t; \boldsymbol{\theta}_q\right)\right)^2. \quad (26)$$

The evaluation actor network is updated in a policy gradient ascent manner as follows

$$\Delta_{\boldsymbol{\theta}_\mu} = \frac{1}{N_B}\sum_{t=1}^{N_B}\left(\nabla_a Q\left(s_t, \mu\left(s_t; \boldsymbol{\theta}_\mu\right); \boldsymbol{\theta}_q\right) | \nabla_{\boldsymbol{\theta}_\mu}\mu\left(s_t; \boldsymbol{\theta}_\mu\right)\right). \quad (27)$$

The proposed DDPG algorithm is summarized in Alg.1.

V. Simulation Results

In this section, we test our proposed solution in a RIS-assisted MEC system as shown in Fig. 4. The number of arriving packets per time slot is set as a random integer uniformly distributed over the range $[\bar{\lambda}-\delta_\lambda, \bar{\lambda}+\delta_\lambda]$ with $\bar{\lambda}$ and δ_λ being the average rate and vibrations, respectively. In the experiment, we set $\bar{\lambda} = 60$ and $\delta_\lambda = 6$ with the size of each packet as 1Mb. Other settings of the numerical experiment is specified as follows: The maximum power supplies $\tilde{e}_k^l = 0.21$ W, and $\tilde{e}_k^o = 40$ W, the capacitance switching coefficient $\kappa_k^l = 10^{-22}$, $\kappa_k^o = 10^{-26}$, the number of cycles needed to process one bit of task data $C_k = 100$. The communication bandwidth B is set as 2 MHz uses of channels per second. We set the number of devices as 3 and assume the wireless

Algorithm 1 DDPG Algorithm

Input: Initialize parameters $\boldsymbol{\theta}$, $\boldsymbol{\theta}'$,learning rates, γ, τ, the replacement steps N_r and N_B.
Output: $\mathbf{v}, \mathbf{f}, \mathbf{p}, \mathbf{g}, \boldsymbol{\phi}$.

```
Randomly initialize the weights θ_μ and θ_q.
Initialize the target network with weights θ_μ ← θ_μ', θ_q ← θ_q'.
Initialize the empty replay buffer D.
for episode i = 1, 2, ..., N do
    Reset the environment and obtain the initial state.
    for each time slot t = 1, 2, ..., T do
        Get the action a_t.
        Observe s_{t+1} given action a_t and obtain r_t.
        Store the s_t, a_t, s_t, a_t in D.
        Sample N_B tuples from D.
        Construct target Q values (25).
        Update Q(s, a; θ_q) by minimizing (26).
        Update μ(s_t; θ_μ) in (27).
        Soft update the target networks.
    end for
end for
```

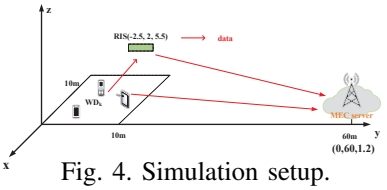


Fig. 4. Simulation setup.

channels are Rician fading. The adopted path-loss model is $P_l = P_{l_0} - 10\delta\log_{10}(d/d_0)$ dB, where $d_0 = 1$ m, P_{l_0} = -30 dB, The path loss exponents of the server-RIS, RIS-device, and server-MU links are set to $\delta_{br} = -2.6$, $\delta_{ru} = -2.6$ and $\delta_{bu} = -3.2$ [14], respectively. Meanwhile, discount parameter for DRL is set as 0.9.

In Fig. 5 we examine the impact of learning rates. In DRL, learning rates of actors/critics can affect the algorithm's converge and should appropriately set. Usually, large learning rate leads to fast convergence while often yielding sub-optimal objective values. On the contrary, too small learning rate generally results in slow convergence. According to our experiments, 0.0001 is generally an appropriate value for both actor and critic. As shown in In Fig. 5, when learning rate is small, the objective iterates often yield an "U" shape trajectory. In fact, before the queues get stablized, the queue length is large and hence much more emphasis is cast on stabilizing the queues (see (22a)) and therefore the original objective is somewhat neglected.

In Fig. 6, we examine our proposed solution's performance. For comparison, we also consider other plain off-loading schemes and learning networks as benchmarks. Specifically, for task scheduling, we consider a basic half-local-half-offloading (HLHO) scheme, where half of the tasks will be sent to the cloud server [2]. For the deep NN structure, we compare our DDPG scheme with the widely used classical actor-critic (AC) structure, which has only a single set of AC network [13]. As reflected by Fig. 6, our proposed joint optimization of task scheduling (i.e. $\mathbf{v}$) outperforms the plain HLHO scheme. Besides, our proposed DDPG based structure also outperforms the widely used AC based structure.

In Fig. 7, we investigate the influence of the penalty parameter V that derives from the Lyapunov optimization.

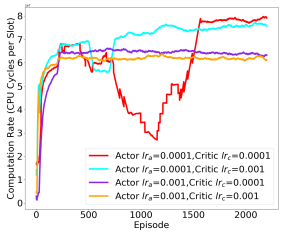


Fig. 5. Impact of learning rate.

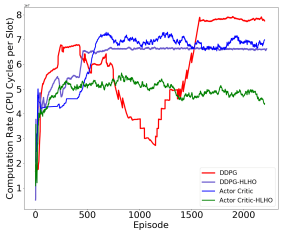


Fig. 6. Performance of different schemes.

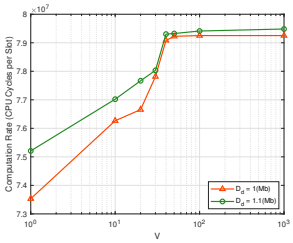


Fig. 7. Impact of the Lyapunov penalty parameter V.

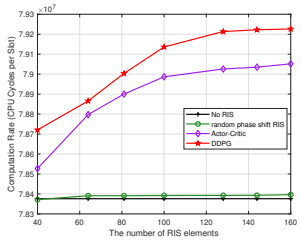
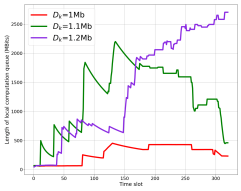
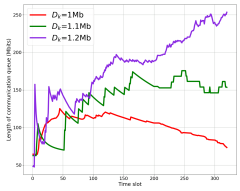


Fig. 8. Impact of the number of RIS elements.

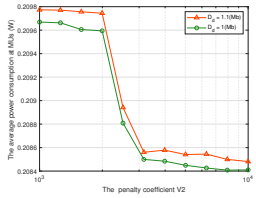


(a) MD-1's computation queue length.

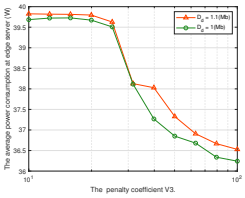


(b) MD-1's communication queue length.

Fig. 9. Average queue length.



(a) The average power consumption at MU versus the penalty coefficient V_2.



(b) The average power consumption at server versus the penalty coefficient V_3.

Fig. 10. The average power consumption versus the penalty coefficient V_2 and V_3.

The weighting factor V determines the priority of the original objective and should not be too small. According to our numerical tests, when the value of V is larger than several tens, the converged objective values becomes rather stable.

In Fig. 8, we illustrate the impact of the RIS size. With other settings unchanged, the system's computation rate can be significantly improved when the number of RIS elements is increased, which boosts the beamforming gain.

Fig. 9 illustrates the queue stabilization effect of the Lyapunov mechanism. With other settings being fixed, the length of the communication and computation queues of MD-1 associated with different arriving task rates (reflected by various D_k) are plotted in Fig. 9. Note when D_k is larger than 1.2Mb, the task rate is indeed out of the capacity region and the system cannot be stable. As can be seen, when the task rate lies within the capacity, our proposed Lyapunov guided solution will finally effectively suppress the growth of queue lengths.

In Fig. 10, we examine the impact of penalty coefficient settings when dealing with the power constraints. Recall that solve (P2), the power constraints (22d) and (22e) should be satisfied. In DRL, we penalizes these constraints with weighting coefficients V_2 and V_3, respectively. As shown by Fig. 10, when V_2 and V_3 are set as several thousands and several tens, respectively, the power constraints can be well satisfied.

VI. Conclusions

In this paper, we consider the joint optimization of task scheduling, RIS reconfiguration and computation resource allocation to improve the long-tern computation rate of an MEC system, where traffic queuing procedures are considered at both users and the cloud server. We develop a Lyapunov guided DRL based solution, which stabilizes the queues and boosts the system's computation rate the in the long run. The benefit of deploying RIS has been verified by numerical results.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge computing: vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637-646, June, 2016.

[2] Y. Kim, J. Kwak, and S. Chong,"Dual-side optimization for cost-delay tradeoff in mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1765-1781, Feb. 2018.

[3] C. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132-4150, June 2019.

[4] H. Hu, Q. Wang, R. Q. Hu and H. Zhu, "Mobility-aware offloading and resource allocation in a MEC-enabled IoT network with energy harvesting," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17541-17556, Dec. 2021.

[5] S. Mao et al., "Computation rate maximization for intelligent reflecting surface enhanced wireless powered mobile edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 10, pp. 10820-10831, Oct. 2021.

[6] Q. Wu, and R. Zhang, "Joint active and passive beamforming optimization for intelligent reflecting surface assisted SWIPT under QoS constraints," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 8, pp. 1735-1748, Aug. 2020.

[7] F. Zhou, C. You and R. Zhang, "Delay-optimal scheduling for IRS-aided mobile edge computing," *IEEE Wireless Commun. Lett.*, vol. 10, no. 4, pp. 740-744, Apr. 2021.

[8] P. D. Lorenzo, M. Merluzzi and E. C. Strinati, "Dynamic mobile edge computing empowered by reconfigurable intelligent surfaces," 2021 IEEE 22nd International Workshop on Signal Processing Advances in Wireless Communications (SPAWC), Lucca, Italy, Nov. 2021.

[9] S. Bi, L. Huang, H. Wang and Y. -J. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519-7537, Nov. 2021.

[10] G. Zhang, Y. Chen, Z. Shen and L. Wang, "Distributed energy management for multiuser mobile-edge computing systems with energy harvesting devices and QoS constraints," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4035-4048, June 2019.

[11] N. Zhang, Y. Liu, X. Mu, and W. Wang, "Queue-aware STAR-RIS assisted NOMA communication systems," Feb. 2022. [Online]. Available: https://arxiv.org/abs/2202.12333

[12] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synth. Lect. Commun. Netw.*, vol. 3, no. 1, pp. 1-211, Jan. 2010.

[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," Jul. 2019. [Online]. Available: https://arxiv.org/abs/1509.02971

[14] F. Zhou, C. You and R. Zhang, "Delay-Optimal Scheduling for IRS-Aided Mobile Edge Computing," *IEEE Trans. Wireless Commun.*, vol. 10, no. 4, pp. 740-744, Apr. 2021.