Joint Optimization Across Timescales: Resource Placement and Task Dispatching in Edge Clouds

Xinliang Wei, *Student Member, IEEE*, A B M Mohaimenur Rahman,
Dazhao Cheng, *Member, IEEE*, and Yu Wang, *Fellow, IEEE*

Abstract—The proliferation of Internet of Things (IoT) data and innovative mobile services has promoted an increasing need for low-latency access to resources such as data and computing services. Mobile edge computing has become an effective computing paradigm to meet the requirement for low-latency access by placing resources and dispatching tasks at the edge clouds near mobile users. The key challenge of such solution is how to efficiently place resources and dispatch tasks in the edge clouds to meet the QoS of mobile users or maximize the platform's utility. In this article, we study the joint optimization problem of resource placement and task dispatching in mobile edge clouds across multiple timescales under the dynamic status of edge servers. We first propose a two-stage iterative algorithm to solve the joint optimization problem in different timescales, which can handle the varieties among the dynamic of edge resources and/or tasks. We then propose a reinforcement learning (RL) based algorithm which leverages the learning capability of Deep Deterministic Policy Gradient (DDPG) technique to tackle the network variation and dynamic as well. The results from our trace-driven simulations demonstrate that both proposed approaches can effectively place resources and dispatching tasks across two timescales to maximize the total utility of all scheduled tasks.

Index Terms—Resource placement, task dispatching, reinforcement learning, optimization, edge computing

1 Introduction

Recently, there has been a tremendous growth of the new computing paradigm - *mobile edge computing* [1], [2], [3] in both academia and industry due to its advances over traditional cloud computing (e.g., low-latency, agility, privacy). Especially as the increasing amount of data and services offered by diverse applications and IoT/smart devices, network operators and service providers are likely to build and deploy computing resources (such as data, models, services) at the edge of the network near users so as to shorten the response time and support real-time intelligence applications.

As shown in Fig. 1, a typical edge computing environment consists of mobile users, edge clouds (including multiple edge servers connected by the edge network), and a remote cloud (usually within data centers). Each edge server is generally deployed at the network edge near mobile users and owns specific storage, CPU, and memory capacity. Mobile users can generate a couple of computation tasks at any location which request to be dispatched at edge servers with sufficient resources (i.e., internal computation resources such as CPU, memory, storage) and may also require certain data or services (i.e., external resources such as training data or machine learning services). Note that the types of computing tasks from mobile users/devices are heterogeneous due to diverse settings and applications. For example, some tasks may only request data (e.g., image, video) or machine learning (ML) model from the edge network, and then process it locally or perform ML computation based on the model at local edge server. Some tasks may request computation at other edge servers with certain computation services, such as video analysis, speech recognition, 3D rendering. Some tasks may need a combination of data, services and computation resources, such as distributed federated learning or interactive augmented reality. Fig. 1 shows some examples where tasks from mobile users request either data/services or both. Note multiple user tasks can be served by the same edge server and the deployment of multiple copies of resources can usually reduce the accessing cost or balance loads among servers. The diverse types of tasks from mobile users and dynamic available resources at edge servers introduce new challenges in resource management and task dispatching in such a complex edge computing system.

Resource management and computation task offloading in edge computing has been widely studied. For example, [4] and [5] have studied the data placement strategy for workflows in edge computing considering workflow's dependency, reliability, and user cooperation. Xie *et al.* [6], [7] and Wei *et al.* [8], [9] have proposed different virtual space-based data placement methods, where both data and servers are mapped into a virtual space and the data placement decision is based on the virtual distance in the space. Li *et al.* [10] and Breitbach *et al.* [11] have investigated both

- *Xinliang Wei and Yu Wang are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19112 USA. E-mail: {xinliang.wei, wangyu}@temple.edu.*
- *A B M Mohaimenur Rahman and Dazhao Cheng are with the Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC 28223 USA. E-mail: {arahman3, dazhao.cheng}@uncc.edu.*

Manuscript received 8 July 2021; revised 9 Sept. 2021; accepted 11 Sept. 2021. Date of publication 20 Sept. 2021; date of current version 8 Mar. 2023.
This work was supported by the National Science Foundation under Grants CCF-1908843 and CNS-2006604.
(Corresponding author: Yu Wang.)
Recommended for acceptance by S. Guo.
Digital Object Identifier no. 10.1109/TCC.2021.3113605

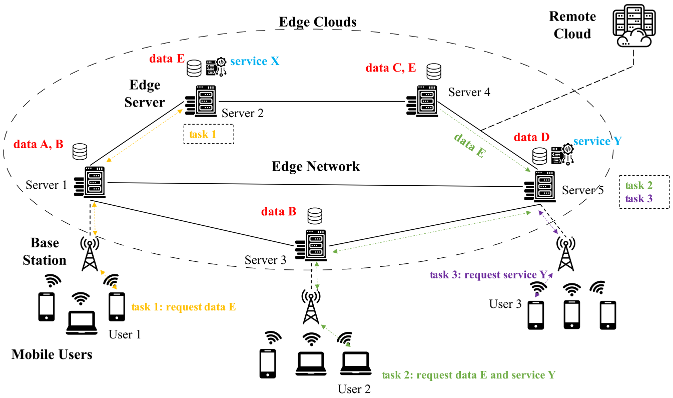


Fig. 1. A typical mobile edge computing environment where data and services are placed at edge clouds and tasks are dispatched to different edge servers. In this example, *task 1* is performed at *server 2*, while *task 2* and *task 3* are performed at *server 5*.

data and task placement in edge computing. While [10] adopted a tabu search based algorithm to solve a joint optimization, [11] considered data/task placement with multiple context dimensions and proposed a context-aware replication strategy. Beside of data placement, service placement has also attracted researchers' attention. Ouyang *et al.* [12] proposed an adaptive user-managed service placement algorithm to jointly optimize the latency and service migration cost. Xu *et al.* [13] studied the service caching in mobile edge clouds with multiple service providers and proposed a distributed caching mechanism for resource sharing. Pasteris *et al.* [14] also studied a multiple-service placement problem and proposed an approximation algorithm to maximize the total reward. There are also recent studies [15], [16], [17], [18] on resource allocation in edge computing. Zhang *et al.* [16] proposed a decentralized multi-provider resource allocation scheme to maximize the overall benefit of all providers, while Meskar and Liang [15] proposed a resource allocation rule retaining fairness properties among multiple access points. Kim *et al.* [17] designed a joint optimization of wireless MIMO signal design and network resource allocation to maximize energy efficiency. Eshraghi and Liang [18] considered the joint optimization of resource allocation and offloading decision for mobile clouds. Similarly, service placement and task/computation offloading has been considered jointly in [19], [20], [21], [22], [23]. However, most of these works consider a kind of joint optimization at a single timescale, thus may not handle the dynamic among tasks, resources, and computation facilities in the edge computing environment.

In a real dynamic edge computing environment, tasks from mobile users generally have a small size and can be easily moved around and distributed at different edge servers for processing. However, the resources, such as data and services, may not be adjusted fast enough to meet the dynamic requirements of tasks. For example, it takes time to reconfigure a service in a new edge server. Similarly, migrating large amount of data also involves additional costs. Therefore, it is nature to manage resources and tasks at two different time scales, i.e., task dispatching can be performed in a fast timescale, while resource placement can occur in a slow timescale. Such multi-timescale solutions have been shown to be more efficient than single timescale methods in edge computing [24], [25]. In addition, a critical factor that has been overlooked is the dynamic status of edge servers. Edge servers are not always running due to regular maintenance or certain events (e.g., power outage and system error). If the status of an edge server is changed, the overall topology of the edge network is changed and this further affects the performance of the entire edge system. Therefore, it is important to taking server status into the resource placement and task dispatching.

In this paper, we jointly study the resource placement and task dispatching problems in mobile edge computing with the aim of maximizing the total utility of performed tasks. We first formulate the problem as an joint optimization problem under the storage, CPU, and memory constraints and take the status of edge servers into account. The overall problem is a nonlinear programming, and thus hard to solve due to its high complexity. In addition, the dynamics of tasks, resources, and the edge environment also make solving this problem much harder. In this paper, we then design two alternative approaches: *two-stage optimization* method and *deep reinforcement learning* method. The two-stage optimization method decomposes the joint optimization problem to two sub-problems (resource placement and task dispatching), and then solves them respectively and iteratively. One nice property of this two-stage optimization method is that it can be performed across two timescales, i.e., performing the joint optimization in each time frame (at a slow timescale) and task dispatching sub-problem only in each time slot (at a fast timescale). To handle the dynamics in edge environment and the complexity of the optimization, we also leverage reinforcement learning (RL) techniques to tackle our joint optimization problem. RL has been used as an effective solution in edge computing [26], [27]. The RL agent can improve its policy to achieve a better goal according to the future reward feedback generated by the environment. Our proposed RL method leverages the Deep Deterministic Policy Gradient (DDPG) [28], [29] to solve the joint optimization problem in dynamic edge environment. Moreover, our deep RL method enables more flexible handling of the multi-timescale problem either by controlling the action or by leveraging multiple DDPG models.

In short, the contributions of this papers are three-folds.

- We formulate an joint optimization problem considering the storage, CPU, and memory constraints as well as the edge server status for the resource placement and task dispatching in mobile edge computing.
- We propose two alternative approaches, *two-stage optimization* and *deep reinforcement learning*, to solve the joint optimization problem. Both methods can be applied across two timescales to deal with different dynamics of tasks and resources in mobile edge computing.
- Extensive trace-driven simulations are conducted to evaluate our proposed methods, and results show both methods can effectively improve the total utility.

The outline of this paper is as follows. Section 2 first introduces the system model and our formulated optimization problems. Sections 3 and 4 present our proposed two-stage optimization method and deep reinforcement based method, respectively. Evaluations of our proposed methods via simulations are provided in Section 5. Section 6 presents an overview of related works. Finally, Section 7 concludes the paper with possible future directions. A preliminary version of this paper appears as [30].

2 System Models and the Optimization

In this section, we first introduce our network and system models under a general edge computing architecture. Then we formulate the resource placement problem, the task dispatching problem, and the joint optimization problem, respectively.

2.1 Network and System Models

Without loss of generality, we construct a typical mobile edge computing architecture as shown in Fig. 1. The edge network topology is defined as graph $G(V,E)$, consists of N *edge servers* and M *direct links* among them. Here, $V=\{v_1,\dots,v_N\}$ and $E=\{e_1,\dots,e_M\}$ are the set of edge servers and the set of links, respectively. For each *server* $v_i \in V$, it has a maximal storage capacity c_i, a CPU frequency f_i, a memory capacity m_i, and the current remaining storage capacity cc_i. For each *link* $e_l \in E$, it has a propagation delay p_l and a network bandwidth b_l.

Assume that there are X *data items* ($D=\{d_1,\dots,d_X\}$), Y *services* ($S=\{s_1,\dots,s_Y\}$) and Z *computing tasks* ($U=\{u_1,\dots,u_Z\}$). Since both data items and services can be considered as needed *resources* for computing tasks, we treat them as $O=X+Y$ resources in total, i.e., $Q=\{q_1=d_1,\dots,q_X=d_X,q_{X+1}=s_1,\dots,q_O=s_Y\}$. For each *resource* q_j, it has a storage size of o_j, a download cost ϖ_j from the cloud, a CPU requirement ζ_j and a memory requirement η_j, respectively. Note that for the data resource, its CPU and memory request are set to 0. Each *task* u_k has a requested resource set Ω_k, a CPU requirement γ_k, a memory requirement δ_k, a size of expected output data β_k, the arriving server Ψ_k, and a benefit ρ_k.

To define the requested resources for task u_k, we introduce a binary variable $\omega_{k,j}$ as the indicator whether resource q_j is required by task u_k.

$$\omega_{k,j}=\begin{cases}1, & \text{resource } q_j \text{ is required by task } u_k,\\ 0, & \text{otherwise.}\end{cases}$$

Then, the requested resource set $\Omega_k=\{q_j|\omega_{k,j}=1\}$, and its input resource size α_k can be calculated as $\alpha_k=\sum_{j=1}^{O}\omega_{k,j}o_j$. Note that the resource requested by task u_k cloud be either data items or specific services.

We assume that tasks arrive at discrete time unit t. The duration of such time unit is τ. Later, we will discuss the case where multiple time scales are used (Sections 3.2 and 4.3). For each server v_i, we also assume there is a status indicator st_i^t to represent whether this server is available at time t (available when $st_i^t=1$, not available when $st_i^t=0$). There are two possible causes to unavailability: predictable (such as scheduled update or maintenance) or sudden events (such as power-outage). Here we mainly consider the first type of cases. For the latter case, different back-up strategies should be considered.

Table 1 summarizes the key symbols used in our paper.

2.2 Resource Placement

We first consider a resource placement problem where a placement decision is needed for each resource q_j at time t. A binary variable $x_{j,i}^t$ is defined as the placement decision in time t where resource q_j is placed in edge server v_i.

$$x_{j,i}^t=\begin{cases}1, & q_j \text{ will be placed in } v_i \text{ at } t,\\ 0, & \text{otherwise.}\end{cases} \tag{1}$$

Here, we assume that data items or services can have replicas in edge cloud (i.e. $\sum_{i=1}^{N}x_{j,i}^t$ can be larger than 1). In addition, an edge server may store multiple data and services, but the total storage size placed in edge server v_i cannot exceed its current remaining storage capacity:

$$\sum_{j=1}^{O}x_{j,i}^t o_j \le st_i^t\cdot cc_i, \text{ for all } v_i. \tag{2}$$

For services, there are also specific CPU and memory requirements on the placed server.

$$x_{j,i}^t\zeta_r \le st_i^t\cdot f_i, \text{ for all } v_i, q_j. \tag{3}$$

$$x_{j,i}^t\eta_r \le st_i^t\cdot m_i, \text{ for all } v_i, q_j. \tag{4}$$

The resource placement aims to maximize the total benefit minus the total cost from all serving tasks, while satisfying resource constraints. Here, we consider two types of costs from serving tasks: *placement cost* and *accessing cost*.

For the *placement cost* of a resource item q_j to a server v_i during the placement, we consider two possible ways: (a) directly downloading from the cloud with a cost of ϖ_j, or (b) transferring from a nearby server v_k, which holds a copy of q_j at $t-1$, with a cost of $f(q_j,v_k,v_i)$. Here, assume that P_j is the shortest path in G^t connecting v_k and v_i[1], then the cost $f(q_j,v_k,v_i)$ can be defined as follow.

$$f(q_j,v_k,v_i)=\begin{cases}0, & \text{if } v_i=v_k\\ \sum_{e_l\in P_j}(\frac{o_j}{b_l}+p_l), & \text{otherwise.}\end{cases} \tag{5}$$

Thus, the placement cost of q_j to v_i at t is the minimal among all these, i.e.,

$$pc_{j,i}^t=\begin{cases}0, & \text{if } x_{j,i}^{t-1}=1\\ \min(\varpi_j, \min_{k\neq i}(x_{j,k}^{t-1}f(q_j,v_k,v_i))), & \text{otherwise.}\end{cases} \tag{6}$$

Note that if q_j is already in v_i at $t-1$, no cost is needed. Then, the placement cost for data q_j at t can be defined as

$$\nu_j^t=\sum_{i=1}^{N}x_{j,i}^t\cdot pc_{j,i}^t. \tag{7}$$

We also define a variable to indicate whether resource q_j is requested by any task:

$$\omega_j=\begin{cases}1, & \text{if } \sum_k\omega_{k,j}\ge 1\\ 0, & \text{otherwise.}\end{cases} \tag{8}$$

[1] Here G^t represents the edge network formed by all available servers at time t. The shortest path is defined regarding the summation of propagation and transmission delays of q_j over the path.

TABLE 1
Symbols Used in the Paper

Symbol	Notation
G, V, E, U, Q, D, S	the edge network, the set of edge servers, direct links, tasks, resources, data items, services
v_i, e_l, u_k, q_j	a edge server, a direct link, a task, and a resource
N, M, Z, O	the number of edge servers/links/tasks/resources
$c_i/cc_i, f_i, m_i$	the maximal/current storage, CPU frequency, memory capacity of edge server v_i
p_l, b_l	the propagation delay and network bandwidth of link e_l
o_j, ϖ_j, ζ_j/η_j	the storage size, download cost from the cloud, CPU/memory requirement of resource q_j
γ_k/δ_k, β_k, Ψ_k, ρ_k	the CPU/memory requirement, output data size, arriving server, and benefit of task u_k
$w_{k,j}$, Ω_k, α_k	an indicator whether resource q_j is required by u_k, the requested resource set and input size of task u_k
t/τ, χ	the index/duration of time unit (time slot), the number of time slots per time frame in two timescales
st_i^t	an indicator whether server v_i is available at time t
$x_{j,i}^t$	the data placement decision at t whether resource q_j is placed at server v_i, here $x_{j,i}^t \in \{0, 1\}$
$f(q_j, v_k, v_i)$	the placement cost of a resource item q_j from v_k to v_i
$pc_{j,i}^t$, ν_j^t	the placement cost of resource q_j to v_i at t, and the placement cost of resource q_j at t
ω_j	the indicator whether resource q_j is requested by any task
$\sigma_{j,k}^t$, $\bar{\sigma}_{j,i}^t$	the accessing cost of task u_k for q_j at t, and the accessing cost of q_j from server v_i at t
$\Upsilon_k = \Upsilon(u_k)$	the server assigned by the tasking dispatching of u_k
$y_{k,i}^t$	the task dispatching decision at t whether task u_k is dispatched to server v_i, here $y_{k,i}^t \in \{0, 1\}$
$\xi_k(z)$	the CPU cycles to process task u_k with the input data/service size z
$C_{k,i}^{input}/C_{k,i}^{output}, C_{k,i}^{comp}$	the accessing cost of resource/output and the computation cost of task u_k at server v_i
$\varsigma_{k,i}^t$	the completion cost of u_k at server v_i at t
$x_{j,i}^{t,\iota}$, $y_{k,i}^{t,\iota}$	the data placement of resource q_j and dispatching decision of task u_k to server v_i at t in ι-th round
cr_i	the available computing resources (e.g., storage, CPU, memory) of server v_i
$RP_j = \{rp_{j,i}\}, TD_k = \{td_{k,i}\}$	the placement decision of resource q_j and the dispatching targets of task u_k over each server v_i
$ss_\iota \in SS$, $aa_\iota \in AA$	the system state at step ι in the state space SS, the system action at step ι in the action space AA
rr_ι	the award obtained given the agent's action aa_ι at step ι
$\mu(s\|\theta^\mu)$, $\mu'(s\|\theta^{\mu'})$	the actor evaluation and target networks
$Q(ss, aa\|\theta^Q)$, $Q'(ss, aa\|\theta^{Q'})$	the critic evaluation and target networks
θ^μ, θ^Q, $\theta^{\mu'}$, $\theta^{Q'}$	the evaluation network parameters and target network parameters
D, K	the replay buffer, the number of sampled data from D
z_i, γ, ε	the expected value/reward, discount factor of future reward, and update rate for target networks

For the *accessing cost* of resource after the data/service is placed, let $\sigma_{j,k}^t$ be the accessing cost for resource q_j required by task u_k. Note that the accessing cost depends on which edge server task u_k is processed at. Let $\Upsilon_k = \Upsilon(u_k)$ be the server assigned by the tasking dispatching of u_k. The accessing cost of q_j can be defined as

$$\sigma_{j,k}^t = \min_{v_i \neq \Upsilon_k} x_{j,i}^t f(q_j, \Upsilon_k, v_i). \tag{9}$$

If without task dispatching, we assume that task u_k is processed at its arriving server Ψ_k, then the accessing cost is

$$\sigma_{j,k}^t = \min_{v_i \neq \Psi_k} x_{j,i}^t f(q_j, \Psi_k, v_i). \tag{10}$$

In general, we define the accessing cost of q_j from any edge server v_i is

$$\bar{\sigma}_{j,i}^t = \min_{l \neq i} x_{j,l}^t f(q_j, v_l, v_i). \tag{11}$$

Since each of serving tasks has benefit of ρ_k, the utility of each task u_k can be defined as $\sum_j (\rho_k - \omega_{k,j} \cdot \sigma_{j,k}^t)$.

Now we can formulate the resource placement problem as an optimization problem. The objective is to maximize total utilities from all serving tasks minus the summation of accessing costs for all resource at time t.

$$\begin{aligned}
\max \quad & \sum_k \sum_j (\rho_k - \omega_{k,j} \cdot \sigma_{j,k}^t) - \sum_j \omega_j \cdot \nu_j^t \\
\text{s.t.} \quad & \sum_j x_{j,i}^t o_j \leq st_i^t \cdot cc_i, && \forall i \\
& x_{j,i}^t \zeta_r \leq st_i^t \cdot f_i, && \forall i, j \\
& x_{j,i}^t \eta_r \leq st_i^t \cdot m_i, && \forall i, j \\
& x_{j,i}^t \in \{0, 1\}, && \forall i, j \\
& i \in (1, 2, \ldots, N), j \in (1, 2, \ldots, O).
\end{aligned} \tag{12}$$

2.3 Task Dispatching

In terms of task dispatching, we assume all tasks arrive in edge network in an arbitrary order. At time t, the goal of task dispatching is to find an optimal edge server v_i to process each task u_k in order to minimize the total completion cost of the task. Specifically, the total completion cost of a task u_k mainly consists of three parts: (a) the accessing cost of resources required by u_k, (b) the computation cost of u_k, and (c) the transmission cost of output data of u_k.

We denote $y_{k,i}^t$ as the task dispatching decision at t whether task u_k is dispatched to edge server v_i.

$$y_{k,i}^t = \begin{cases} 1, & \text{task } u_k \text{ is dispatched to server } v_i \text{ at } t, \\ 0, & \text{otherwise.} \end{cases} \tag{13}$$

Here we assume that each task is at most dispatched to a single server, i.e., $\sum_{i=1}^N y_{k,i}^t \leq 1$.

Note that there are different types of tasks: some only need data from the edge network, some only need to perform general computation at any server either with data or not, some need to perform specific computation with certain service at the available server. Our formulation can model all these task types. If task u_k only needs data, $\gamma_k = 0$, $\delta_k = 0$ while $\alpha_k > 0$. If u_k only needs general computation without specific service or data, $\gamma_k > 0$, $\delta_k > 0$ while $\alpha_k = 0$.

Assume that task u_k is dispatched to edge server v_i, i.e., $y^t_{k,i} = 1$, then its associated costs are defined as follows.

Accessing cost of resources: The transmission cost of input data and needed service for task u_k is defined as $C^{input}_{k,i} = \sum_{j=1}^{O} \omega_{k,j} \cdot \bar{\sigma}^t_{j,i}$.

Computation cost:. Let $\xi_k(z)$ be the function to define CPU cycles to process task u_k with the input data/service size z. So the computation cost of task u_k processed in edge server v_i is defined as $C^{comp}_{k,i} = \sum_{j=1}^{O} \omega_{k,j} \cdot \frac{\xi_k(o_j)}{f_i}$.

Transmission cost of output: The total transmission cost of output data for task u_k from edge server v_i to arriving edge server Ψ_k is $C^{output}_{k,i} = f(\beta_k, v_i, \Psi_k)$.

Therefore, the completion cost of task u_k is calculated as

$$\varsigma^t_{k,i} = C^{input}_{k,i} + C^{comp}_{k,i} + C^{output}_{k,i}. \tag{14}$$

Recall each task has a benefit ρ_k. We then can formulate the task dispatching decision as an optimization problem whose goal is to maximize the total task utility if task u_k is running on server v_i at t.

$$\begin{aligned}
\max \quad & \sum_k \sum_i y^t_{k,i}(\rho_k - \varsigma^t_{k,i}) & \\
\text{s.t.} \quad & \sum_k y^t_{k,i}\varsigma^t_{k,i} \le \tau, & \forall i \\
& y^t_{k,i}\alpha_k \le st^t_i \cdot cc_i, & \forall i,k \\
& y^t_{k,i}\gamma_k \le st^t_i \cdot f_i, & \forall i.k \\
& y^t_{k,i}\delta_k \le st^t_i \cdot m_i, & \forall i,k \\
& \sum_i y^t_{k,i} \le 1, & \forall k \\
& z^t_{k,i} \in \{0,1\}, & \forall k, \forall i \\
& i \in (1,2,\ldots,N), k \in (1,2,\ldots,Z). &
\end{aligned} \tag{15}$$

Note that the constraint of $\sum_k y^t_{k,i}\varsigma^t_{k,i} \le \tau$ makes sure that the dispatched tasks can be completed within the duration of a time scale τ.

2.4 Joint Optimization Problem

We now consider a joint resource placement and task dispatching problem as a nonlinear program problem:

$$\max \quad \sum_k \sum_i y^t_{k,i}(\rho_k - \varsigma^t_{k,i}) - \sum_j \omega_j \cdot \nu^t_j \tag{16}$$

$$\text{s.t.} \quad \sum_j x^t_{j,i} o_j + y^t_{k,i}\alpha_k \le st^t_i \cdot cc_i, \qquad \forall i,k \tag{17}$$

$$x^t_{j,i}\zeta_j \le st^t_i \cdot f_i, \qquad \forall i,j \tag{18}$$

$$x^t_{j,i}\eta_j \le st^t_i \cdot m_i, \qquad \forall i,j \tag{19}$$

$$y^t_{k,i}\gamma_k \le st^t_i \cdot f_i, \qquad \forall i,k \tag{20}$$

$$y^t_{k,i}\delta_k \le st^t_i \cdot m_i, \qquad \forall i,k \tag{21}$$

$$\sum_i y^t_{k,i} \le 1, \qquad \forall k \tag{22}$$

$$\sum_k y^t_{k,i}\varsigma^t_{k,i} \le \tau, \qquad \forall i \tag{23}$$

$$x^t_{j,i} \in \{0,1\}, \quad y^t_{k,i} \in \{0,1\} \tag{24}$$

$$i \in (1,\ldots,N), j \in (1,\ldots,O), \tag{25}$$

$$k \in (1,\ldots,Z). \tag{26}$$

Since there is a nonlinear term inside $y^t_{k,i}\varsigma^t_{k,i}$, the overall problem is a nonlinear integer program problem which is known difficult to solve due to its high computational complexity.

3 Two-Stage Optimization Method

To solving the challenging joint optimization problem, we propose a two-stage algorithm to decompose the problem and solve it via multiple iterations. One of the advantages of this proposed two-stage method, it can be easily adopt to perform the joint optimization across different timescales.

3.1 Two-Stage Optimization

The main idea of this algorithm is as follows. First, we randomly generate a feasible task dispatching decision $y^{t,0}_{k,i}$, then formulate and solve the resource placement problem (obtaining $x^{t,1}_{j,i}$) to maximize the total task utilities. Next, we take the resource placement decision $x^{t,1}_{j,i}$ as input, and formulate and solve the task dispatching problem (obtaining $y^{t,1}_{k,i}$). This finishes the first round of two-stage optimization, then we repeat the two steps, i.e., iteratively taking the latest resource placement or task dispatching decision as an input to optimize the other decision within the overall joint problem, until it satisfies a specific condition.

2-Stage Decomposition: The detail of decomposition of ι-th round is as follows.

Stage 1: Solving resource placement problem with fixed task dispatching. In this stage, our goal is to determine resource placement for each data and service in order to maximize the total task utilities with the last task dispatching decision $y^{t,\iota-1}_{k,i}$. The problem can be formulated as **P1**:

$$\begin{aligned}
\max \quad & \sum_k \sum_i y^{t,\iota-1}_{k,i}(\rho_k - \varsigma^t_{k,i}) - \sum_j \varrho^t_j \\
\text{s.t.} \quad & (17), (18), (19), (23), (24), (25), (26).
\end{aligned} \tag{27}$$

The solution of this problem is $x^{t,\iota}_{j,i}$.

Stage 2: Solving task dispatching problem with fixed resource placement. In this stage, we take the resource placement decision $x^{t,\iota}_{j,i}$ generated in the first stage as input and determine the task dispatching for each task $y^{t,\iota}_{k,i}$ to maximize the total utility. The problem can be formulated as **P2**:

$$\begin{aligned}
\max \quad & \sum_k \sum_i y^{t,\iota}_{k,i}(\rho_k - \varsigma^t_{k,i}) - \sum_j \varrho^t_j \\
\text{s.t.} \quad & (17), (20) - (26).
\end{aligned} \tag{28}$$

The solution of this stage is $y^{t,\iota}_{k,i}$.

Algorithm 1. Two Stages Optimization Method

```
Input: Status of all servers $V$ and the network $G$, resources $Q$ and tasks $U$ for time $t$.
Output: Resource placement and task dispatching decisions $x^t_{j,i}$ and $y^t_{k,i}$.
1: Initialize $max\_itr$, $max\_occur$, $bound\_val$
2: Generate an random initial task dispatching decision $y^{t,0}_{k,i}$ which is feasible (i.e., satisfying constraints in P2)
3: $\iota = 1$ and $count\_num = 0$;
4: repeat
5:    Stage1: Calculate $x^{t,\iota}_{j,i}$ by solving P1 with $y^{t,\iota-1}_{k,i}$ as the fixed task dispatching
6:    Stage2: Calculate $y^{t,\iota}_{k,i}$ by solving P2 with $x^{t,l}_{j,i}$ as the fixed resource placement, let $obj\_val$ be the achieved objective value (total utility from tasks)
7:    if $obj\_val > bound\_val$ then
8:       $bound\_val = obj\_val; count\_num = 1$
9:       $x^t_{j,i} = x^{t,\iota}_{j,i}; y^t_{k,i} = y^{t,\iota}_{k,i}$
10:   else if $obj\_val = bound\_val$ then
11:      $count\_num = count\_num + 1$
12:   $\iota = \iota + 1$
13: until $count\_num = max\_occur$ or $\iota = max\_itr$
14: return $x^t_{j,i}$ and $y^t_{k,i}$
```

After the decomposition, in each round, both **P1** and **P2** are linear integer programming problems, and thus can be solved by the classical linear programming methods (e.g., branch and bound, dynamic programming).

Overall Iteration, Initialization and Termination: Algorithm 1 shows the overall algorithm. Initially, a feasible random task dispatching $y^{t,0}_{k,i}$ is generated (Line 2). Then, in each round (Lines 5-12), we solve the **P1** and **P2** with the previous decision as the input. The resource placement and task dispatching decisions ($x^{t,\iota}_{j,i}$ and $y^{t,\iota}_{k,i}$) are optimized iteratively. Finally, the iteration terminates (Line 13) when either of the following metric met: (1) the number of iteration reach certain threshold max_itr, or (2) the current objective value (total task utility) has occurred more than a specified threshold max_occur. These two thresholds can be set via experiments. Obviously, larger threshold values lead to longer iteration but improved results. In Section 5, we will show the improvement is limited after certain round of iterations.

3.2 Joint Optimization Across Two Timescales

So far, we only discuss our two-stage algorithm in a one-time slice. In edge computing systems, the workload (i.e., computing tasks) and the resources (e.g., data or services) to serve such workload need to be managed on different timescales [24], [25]. Usually the computing tasks could be distributed more frequently at a fast timescale in the edge network, while the resource placement could be adjusted (such as redeploying or migrating services) less frequently on a slow timescale. Compared with the single timescale method, multi-timescale solutions [24], [25] can achieve better performance with more flexible management, thus gain significant attractions recently from the research community.

Our proposed two-stage algorithm can be easily to adopt to a two-timescale solution. As illustrated in Fig. 2, we can make task dispatching decisions along with the fast timescale (at the starting point of each time slot) and make resource placement decisions along with the slow timescale (at the starting point of each time frame). Here, we assume that each time frame includes χ time slots. More specifically, at the beginning of each time frame, we run our proposed iterative two-stage algorithm (Algorithm 1), and at the beginning of each time slot (except for the first time slot), we only solving the Stage 2 problem (**P2**) where the resource placement is fixed. By doing so, not only we can handle diverse dynamics among workload and resources, but also the running time of overall algorithm is reduced since the iterative algorithm is only performed once at each time frame and solving **P2** at each time slot is relevantly simpler. Thus, it leads to greater flexibility with more cost savings.

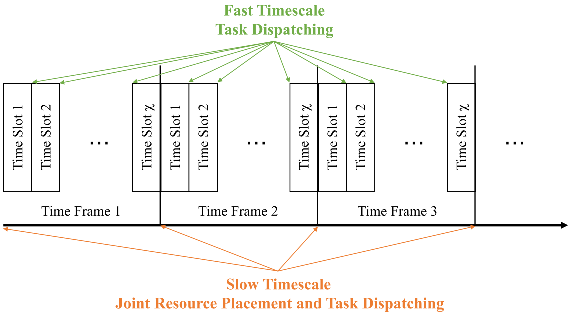


Fig. 2. Illustration of joint resource placement and task dispatching across two timescales.

4 Reinforcement Learning Based Method

In this section, we consider an alternative method to solve the joint optimization by leveraging the emerging deep reinforcement learning technique. Reinforcement learning (RL) has a great capability to attack complex optimization problems in a dynamic system. The characteristic of RL framework is that the decision is made by RL agents and the feedback generated by the environment is used to improve the decision of the agent. There are three key elements in the RL frameworks: state, action and reward.

Generally, RL algorithms can be classified as the category of value-based and policy-based methods. Value-based RL methods (e.g., Q-learning, Deep Q-network (DQN) [31], Double DQN [32]) can select and evaluate the optimal value function with lower variance. The value function measures the goodness of the state (state-value) or how good is to perform an action from the given state (action-value). However, it is difficult for value-based methods to handle the problem of continuous action spaces. If it calculates the value in an infinite number of actions, it will be time-consuming.

On the other hand, policy-based methods, such as policy gradient [29], are effective in high-dimensional or continuous action spaces. It can learn stochastic policies and has better convergence properties. The main idea is to able to determine at a state which action to take in order to maximize the reward. The way to achieve this objective is to find tune a vector of parameters (θ) so as to select the best action to take for policy π. The policy π is the probability of taking action a when at state s and the parameters are θ. There are some disadvantages for policy-based methods: (1) it

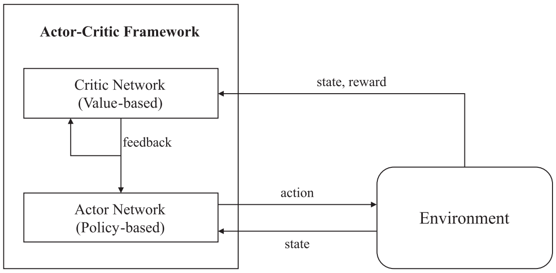


Fig. 3. The architecture of Actor-Critic RL framework.

typically converges to a local rather than global optimum; (2) evaluating a policy is typically inefficient and high variance.

Actor-Critic RL method [33] is proposed to combine the basic idea of value-based and policy-based algorithms. The actor uses policy-based methods to select the action while the critic uses value-based methods. As shown in Fig. 3, the actor takes the state as input and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy (policy-based). The critic, on the other hand, evaluates the action by computing the value function (value based). And the feedback (such as error) will tell the actor how good its action was and how it should adjust. However, since the actor-critic method involves two neural networks, each time the parameters are updated in a continuous state and there is a correlation before and after each parameter update, which causes the neural network to only look at the problem one-sidedly, and even causes the neural network to learn nothing. To avoid such problem in our problem, we leverage Deep Deterministic Policy Gradient (DDPG) RL technique [28], [29] to solve the joint optimization problem.

4.1 RL Framework: State, Action and Reward

We first define the specific state vector, action vector and reward for our system model to enable the proposed RF framework.

State Vector: At each step ι, the agent collects the edge network information and parameters defined below to form the system state.

- M: the number of links among edge servers.
- N: the number of edge servers.
- b_l: available network bandwidth of each link.
- cr_i: available computing resources (e.g., storage, CPU, memory) of each edge server.

Let SS be the state space, the system state $ss_\iota \in SS$ at step ι can be defined as

$$ss_\iota = \{b_1, b_2, \ldots, b_M, cr_1, cr_2, \ldots, cr_N\}_\iota.$$

Action Vector: In terms of action vector, the agent will make decisions for both resource placement and task dispatching. The decision mainly consist of where to place resources and where to dispatch tasks. Therefore, the action vector includes two parts.

- $RP_j = \{rp_{j,1}, rp_{j,2}, \ldots, rp_{j,N}\}$: resource placement of each external resource q_j (data, service).

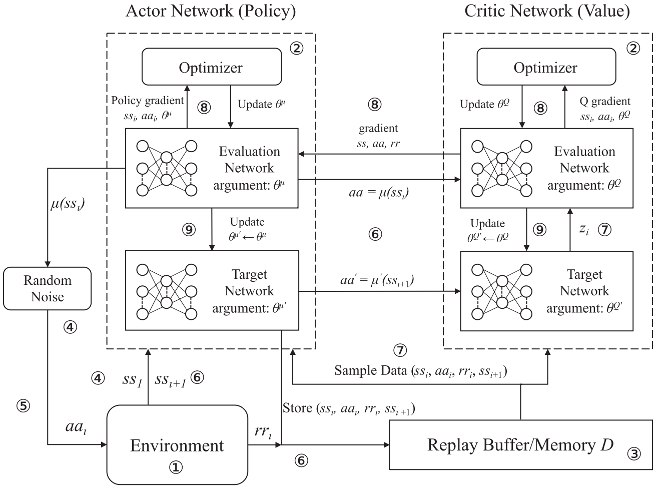


Fig. 4. The architecture of DDPG RL Algorithm. The circled numbers are the corresponding steps.

- $TD_k = \{td_{k,1}, td_{k,2}, \ldots, td_{k,N}\}$: dispatching target of each task u_k (released by mobile user).

Let AA be the action space, the system action $aa_\iota \in AA$ at step ι can be defined as

$$aa_\iota = \{RP_1, RP_2, \ldots, RP_R, TD_1, TD_2, \ldots, TD_Z\}_\iota.$$

Reward: For each step, the agent will get the reward rr_ι from the environment after taking a possible action aa_ι. Generally, the reward function is related to the objective function in the optimization problem. Fortunately, the objective of our optimization problem is to maximize the total utility of all tasks, so the award of RL agent is to set as following.

$$rr_\iota = \sum_k \sum_i (\rho_k - \varsigma^t_{k,i}) - \sum_j \varrho^t_j. \tag{29}$$

Notice that the reward rr_ι can be obtained given the agent's action aa_ι, which includes the solution of both resource placement and task dispatching, and the environment.

4.2 DDPG RL Algorithm

The main goal of RL algorithm is to tune the learning model's parameters (θ) so as to select the best action aa to take based on the given state. We adopt Deep Deterministic Policy Gradient (DDPG) technique [28], [29] to perform the RL. Actually, DDPG integrates the essential idea of the actor-critic and DQN. DQN uses a replay memory and two sets of neural networks with the same structure but different parameter update frequencies, which can effectively promote learning. DDPG has a similar idea but the neural network is a bit complicated. As aforementioned, compared with other RL methods, policy gradient can be used to filter actions in continuous action spaces. Moreover, the screening is performed randomly based on the learned action distribution. However, the screening in DDPG is deterministic but not random. In terms of the architecture of neural networks in DDPG, it is similar to that of Actor-Critic, both need the policy-based neural networks and the value-based neural networks as shown in Fig. 4. Each kind of neural network also includes two types of neural networks: the evaluation network and the target network. The target networks are time-delayed copies of their original networks that slowly track the learned networks. Using these target networks greatly improve stability in learning.

Algorithm 2. DDPG method

```
Input: The edge network $G(V, E)$, set of external resource $Q$, set of task $U$. Remained storage capacity $cc_i$, CPU frequency $f_i$, memory capacity $m_i$ of each edge server. Propagation delay $p_j$, network bandwidth $b_j$ of each link. Storage size $o_j$, download cost $\varpi_j$, CPU requirement $\zeta_j$ and memory requirement $\eta_j$ of each resource.
1:  Initialize the environment with all input information.
2:  Initialize Actor and Critic evaluation network $\mu(s|\theta^{\mu})$ and
    $Q(ss, aa|\theta^{Q})$ with parameters $\theta^{\mu}$ and $\theta^{Q}$, respectively.
3:  Initialize Actor and Critic target network $\mu'(s|\theta^{\mu'})$ and
    $Q'(ss, aa|\theta^{Q'})$ with parameters $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$.
4:  Initialize empty replay buffer $D$, the maximum episodes
    $max\_ep$ and the maximum steps per episode $max\_st$.
5:  for $episode = 1, episode < max\_ep$ do
6:    Initialize the random exploration noise for action.
7:    Generate the initial observation state $ss_1$ from
      environment.
8:    for each step $\iota = 1, \iota < max\_st$ do
9:      Calculate action $aa_{\iota}$ based on the current policy and
        random noise.
10:     Execute action $aa_{\iota}$ in the environment and observe
        reward $rr_{\iota}$ and new state $ss_{\iota+1}$.
11:     Store transition $(ss_{\iota}, aa_{\iota}, rr_{\iota}, ss_{\iota+1})$ to replay buffer $D$.
12:     Send the action from Actor evaluation and target
        network to Critic evaluation and target network,
        respectively.
13:     Randomly sample a batch of transitions $(ss_i, aa_i, rr_i,$
        $ss_{i+1})$ from $D$ to Actor and Critic network.
14:     Calculate $z_i = rr_i + \gamma Q'(ss_{i+1}, \mu'(ss_{i+1}|\theta^{\mu'})|\theta^{Q'})$, where $\gamma$
        is the discount factor for future rewards.
15:     Update Critic evaluation network by minimizing the
        loss: $\frac{1}{K}\sum_i (z_i - Q(ss_i, aa_i|\theta^{Q}))^2$, where $K$ is the number
        of sampled data from $D$.
16:     Send gradient parameters to Actor evaluation network.
17:     Update Actor evaluation network by using the sampled
        policy gradient:
        $$\frac{1}{K}\sum_i \nabla_{aa} Q(ss, aa|\theta^{Q})|_{ss=ss_i, aa=\mu(ss_i)} \nabla_{\theta^{\mu}} \mu(ss|\theta^{\mu})|_{ss_i}.$$
18:     Update Actor and Critic target network by using the
        evaluation network arguments:
        $$\theta^{\mu'} \leftarrow \varepsilon\theta^{\mu} + (1-\varepsilon)\theta^{\mu'}, \quad \theta^{Q'} \leftarrow \varepsilon\theta^{Q} + (1-\varepsilon)\theta^{Q'}.$$
19:   end for
20: end for
```

Algorithm 2 is the detail of DDPG algorithm. The main steps of DDPG algorithm (with corresponding lines in Algorithm 2) are as follows.

1) Initialize the system and environment based on the edge network G, and set of external resource Q and set of task U as well as other network information (Line 1).
2) Initialize Actor evaluation network $\mu(s|\theta^{\mu})$ and target network $\mu'(s|\theta^{\mu'})$ as well as Critic evaluation network $Q(ss, aa|\theta^{Q})$ and target network $Q'(ss, aa|\theta^{Q'})$, where θ^{μ} and θ^{Q} are evaluation network parameters, $\theta^{\mu'}$ and $\theta^{Q'}$ are target network parameters (Lines 2-3).
3) Initialize replay buffer D, the maximum number of episodes max_ep and the maximum number of steps per episode max_st (Line 4). D is used to sample experience to update neural network parameters.
4) At the beginning of each episode, initialize the random exploration noise and generate the initial state ss_1 (Lines 5-7).
5) For each step ι, the actor selects an action aa_{ι} based on the current policy and random noise (Lines 8-9).
6) The environment executes action aa_{ι} and get the reward rr_{ι} and observe new state $ss_{\iota+1}$. Then it stores the transition $(ss_{\iota}, aa_{\iota}, rr_{\iota}, ss_{\iota+1})$ to D. At the same time, the actor send the action to critic network (Lines 10-12).
7) Randomly sample a batch of data $(ss_i, aa_i, rr_i, ss_{i+1})$ from D. Then calculate the expected value/reward z_i (Lines 13-14).
8) Update Critic and Actor evaluation network with the sampled data (Lines 15-17).
9) Update Actor and Critic target network with the rate ε (Lines 18).
10) This process is done until it reaches the maximum number of episode.

Fig. 4 also shows these steps with the circled numbers.

4.3 RL Method Across Two Timescales

While RL technique can handle network dynamics, it is also flexible to deal with the complexity in multiple timescales scenario. We now further extend our proposed DDPG to work across two timescales. There are two different ways to extend the proposed DDPG method. A straightforward way is to build another separate DDPG for task dispatching problem **P2**, and run both DDPG models in different timescales (joint one for each time frame and **P2** one for each time slot). The other way is to use the same DDPG model, but force the action policy to not adjust the resource placement during the fast timescale. With either way, the agent can still learn the best decision based on the environment and the current state vector. In this paper, we adopt the first method, as shown in Fig. 5. We use two DDPG networks, one for resource placement (RP DDPG) and the other for task dispatching (TD DDPG). Resource placement (RP DDPG) is performed every specific time frame while task dispatching (TD DDPG) is executed every time slot. In each time slot, the environment sends current network state (available network bandwidth and computing resources) to task dispatching agent (TD agent), the TD agent will output the task dispatching decision to the environment. Our experimental results (Section 5.5) show that by leveraging two DDPG agents, the proposed RL method can handle the system dynamic and manage the resource/task effectively.

5 Performance Evaluation

This section reports the results from our trace-based simulations to evaluate our proposed strategies.

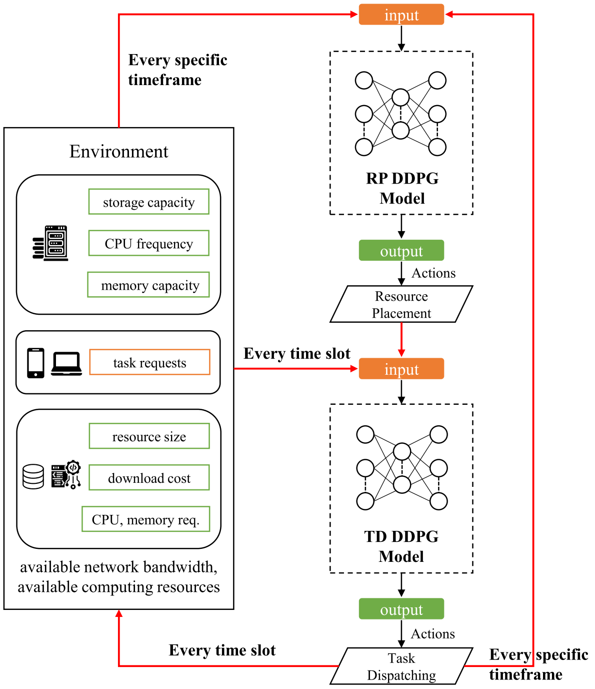


Fig. 5. Resource placement and task dispatching via deep reinforcement learning across two timescales with two DDPG models.

5.1 Simulation Setup

In our simulation, we randomly construct edge networks G with 10 to 50 edge servers whose degree satisfies a binomial distribution. The propagation and bandwidth for each network link are randomly generated. Each edge server has a limited storage capacity ranges from 512MB to 1,024MB. To simulate the CPU, memory and status of edge servers, we make use of the Google Cluster Data (ClusterData 2011 traces) [34]. For the external resources (data and services), we randomly generate 100 data items and 20 services where the size of each resource are from 10MB to 200MB. To simulate the tasks from mobile users, we leverage the user mobility data from the CRAWDAD dataset kaist/wibro [35], developed by a Korean Team, which collected the CBR and VoIP traffic from the WiBro network in Seoul, Korea. We randomly sample from this dataset to generate the random tasks from mobile user to perform our simulation. We run our experiments on a DELL Precision 3630 Tower with i7-9700 CPU, 16GB RAM and NVIDIA GeForce RTX 2060 GPU. For our proposed RL based method, the detail of hyper parameters configuration is reported in Table 2. The parameters are initialized by general value that used in most RL experiments. We test multiple values for each parameter and select the value that has better performance.

We compare our proposed *Two Stage Optimization (OPT)* and *Deep Reinforcement Learning (RL)* solutions with two baselines: a random strategy and a greedy strategy.

- *Random (RAND)*. At each time slice, it randomly generates a feasible resource placement and task dispatching decision which satisfies those constraints.
- *Greedy (GRD)*. It greedily determines its resource placement and task dispatching decision to maximize total utility in each round. It gives the priority to resources/tasks based on their popularity/benefits. Specifically, GRD first sorts resources based on their popularity and processes them from the most popular one. It iteratively selects an edge server to place this resource which maximizes the total utility in each round. Similarly, for tasking dispatching, GRD sorts all tasks based on their benefits and processes the most beneficial task first. Likewise, it greedily selects an edge server to dispatch the task to get the maximal task utility in each round iteratively.

TABLE 2
RL Hyper Parameters

Parameter	Value	Parameter	Value
Max Episode	100	Reward Discount	0.9
Max Step per Episode	3,000	Batch Size	32
Learning Max Episode	10	Soft Replacement	0.01
Actor Learning rate	0.0001	Replay buffer Capacity	10,000
Critic Learning rate	0.0002		

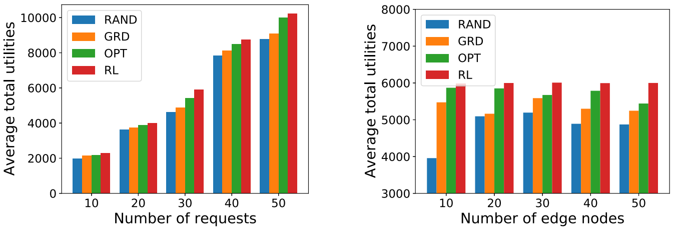
We evaluate the performance of all methods based on average total utility (i.e., the objective function in our formulated optimization problems). Obviously, the larger utility value the better resource placement and task dispatching performance. All parameters required to calculate the objective function (such as network topology, bandwidth, task requirements, server capacity, download cost) are known to all methods as inputs at each time unit. For RL methods, those are used to calculate the reward at each time unit.

5.2 Overall Performance

In the first set of simulations, we test all four methods within a fixed time period (in the single timescale) over different numbers of tasks or edge servers.

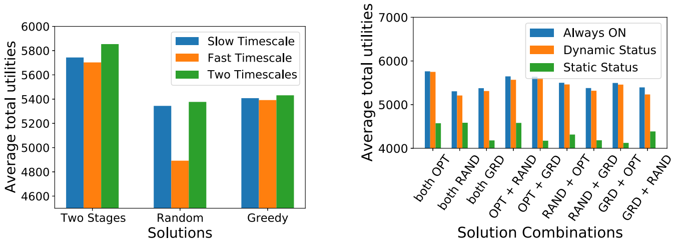
Fig. 6a displays the performances for the four solutions under different number of task requests (from 10 to 50 in each time unit). The number of edge servers is fixed at 30. It is obvious to see that the average total benefits of four solutions increase as the number of task requests increases. Our proposed two stage optimization algorithm (OPT) and Reinforcement Learning (RL) outperform the other two algorithms (RAND and GRD) in all cases. In addition, when the number of requests is low (e.g., 10 or 20), the difference of average total utilities between OPT and RL is small. However, as the number of requests increases, the difference becomes larger. So, in the real scenario, we can select either OPT or RL if the number of requests is low. If the number of request is large, we prefer to use RL to make the decision.

We then fix the number of tasks at 30 and investigate the impact of the number of edge servers (changing from 10 to 50). As shown in Fig. 6b, the average total utility of RAND increases in the beginning and then less varies as the number of edge servers increases. For other three solutions, OPT and GRD vary a little as the number of servers increases while RL keeps stable all of the way. Overall, the performance of most of the solutions are relevantly stable, especially RL. For all cases, RL and OPT perform much better than GRD and RAND. This once again confirms the advantage of our two proposed methods.



(a) number of task requests (b) number of edge servers

Fig. 6. *Overall performance of four methods in one timescale:* Comparison of proposed solutions (OPT, RL) with Random (RAND) and Greedy (GRD) strategy with different numbers of tasks or edge servers.



(a) single vs two timescales (b) dynamic vs static status

Fig. 8. *Performance of OPT across two timescales with dynamic status:* (a) different methods at fast, slow or two timescales; (b) with dynamic status from real-world traces.

5.3 Running Time and Convergence of OPT

We first investigate the running time and convergence of our proposed two-stage optimization (OPT) method.

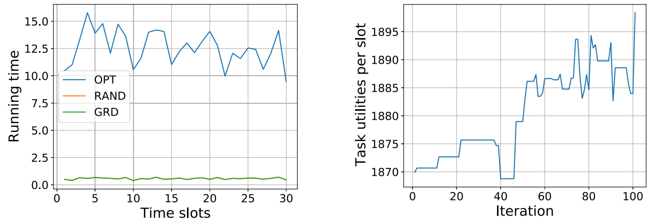
Fig. 7a shows the running time of OPT, GRD and RAND at different time slots. The running time is defined as the time duration when the algorithm is executing. We can find that GRD and RAND has the least running time since their placement/dispatching can be done in a polynomial time. Our OPT method spends more time to solve the challenging optimization problem, but remember that it generates much better solution (better total utilities) than GRD/RAND as shown in Fig. 6.

Recall that our two-stage optimization algorithm (Algorithm 1) iteratively optimizes the objective value under a max iteration. Fig. 7b displays the total task utility per slot under different iterations. It is clear that with more iterations the overall trend of performance increases, even though there is an drop in early iteration and some variety in each iteration. Therefore, it is necessary to select an appropriate max iteration (max_itr) to achieve a decent performance (total utility). It is a trade-off between the max iteration and the running time as well as the optimization objective value since more iterations cost more running time.

5.4 OPT Across Two Timescales

We further investigate our proposed methods across two timescales where the joint resource placement and task dispatching decisions are made at different timescales (time frame versus time slot). Here we mainly focus on our two-stage optimization solution (OPT).

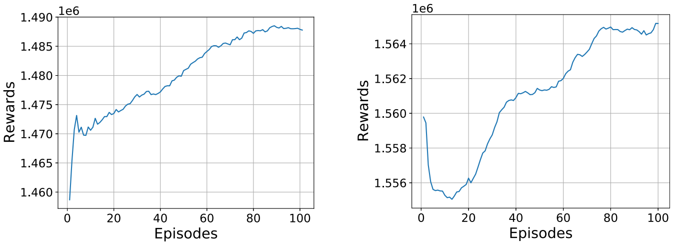
In the first set of experiments, we perform our proposed OPT method against RAND/GRD in three different scenarios: (1) *Slow Timescale*: all methods perform joint resource placement and task dispatching at the beginning of each time frame; (2) *Fast Timescale*: all methods perform placement and dispatching at the beginning of each time slot; (3) *Two Timescales*: all methods perform task dispatching in each time slot while joint resource placement and task dispatching performing only at the beginning of each time frame. In this set of experiments, each time frame has 5 time slots (i.e., $\chi = 5$), and we fix the number of request per time frame at 30 and the number of edge server at 30. Fig. 8a displays the performances of three methods (OPT, RAND, GRD) under three scenarios. First, our proposed two-stage OPT method achieve better performance than RAND and GRD in all setting. Second, for all three solutions, running at slow timescale achieves larger utilities than running at fast timescale. This is mainly due to running at slow timescale takes the advantage having better global information over longer time duration. In addition, fast timescale solution also suffers from frequent resource placement changes which might be costly. Third, when the solutions are performed across two timescales, the performances can be further improved. This might due to performing task dispatching at the time slot can find sufficient server to perform the task and quickly release the server for other tasks. Overall, the results from this set show that multi-timescale solution can achieve better performance compared with the single timescale method, which echos the similar discovery from [24], [25] (though the studied problems and network models are different).



(a) running time (b) total utility per slot

Fig. 7. *Running time and convergence of OPT:* (a) Comparison of running time (OPT versus RAND/GRD); (b) Convergence of OPT.

Finally, we evaluate our proposed two-timescale solutions over edge servers with dynamic status by leveraging the status trace-driven data from the Google Cluster Data (ClusterData 2011 traces) [34]. We use the trace data to generate the server status at different time slots. Other parameters are similar to previous experiments. For two-timescale solutions, we use different combinations of OPT/GRD/RAND to solve data placement and task dispatching problems respectively. As shown in Fig. 8b, there are nine combinations in total. For example, OPT+RAND means optimization based method is used for data placement, while task dispatching is done randomly. Fig. 8b reports the results of these methods under three different scenarios: (1) *Always On*: assume that all edge servers are always running and available for serving tasks; (2) *Dynamic Status*: the status of the edge node varies along with the time slot, while a server is down at a time slot no task can be dispatched to it; (3) *Static Status*: our method completely ignore the server



(a) single timescale (b) across two timescales

Fig. 9. *Convergence of RL under different timescales:* Running joint resource placement and task dispatching under single timescale and across two timescales.

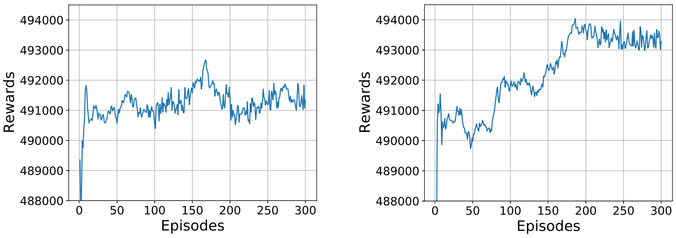
status during solving the data placement and task dispatching. Obviously, all combinations with dynamic status have lower total utility than those of always on, since some server may be unavailable in certain time slots. In addition, if ignoring the status, the performance (of static status) will be significantly reduced, since the dispatched tasks may not be completed due to the server is unavailable. Clearly, our solutions which considers dynamic status can achieve a comparative performance to the case where every server is on. Last, among all nine combinations, using our optimization based solution for both resource placement and task dispatching across two-timescales has higher performance than other combinations. This indirectly illustrates the effectiveness of the two-stage algorithm under two-timescales to handle real dynamics in edge computing, which is the major contribution of this paper.

5.5 Performance and Convergence of RL

In this subsection, we study the performance and convergence of our proposed deep RL methods. The default number of edge servers is set to 10.

Convergence Performance of RL Under Single Timescale and different timescales. Fig. 9a displays the convergence result of our RL solutions that jointly determine the resource placement and task dispatching decision in a single timescale. As we can see, the reward gets higher as the number of episodes increases and it converges at around the 80th episode. On the other hand, Fig. 9b shows the convergence of our RL solutions across two timescales where makes the task dispatching decision in the fast timescale and the resource placement decision in the slow timescale. We can find that the reward drops in the beginning and then increases when the training episode increases. We also observe that the reward in Fig. 9b is higher than that in Fig. 9a. This further confirms the benefit of making resource placement and task dispatching across two timescales.

Convergence Performance of Resource Placement and Task Dispatching. We further show the convergence result of resource placement and task dispatching, respectively. We first fix the task dispatching decision in each episode and make the resource placement decision. Similarly, we then fix the resource placement decision in each episode and make the task dispatching decision. As shown in Fig. 10, the results of resource placement and task dispatching are similar since they use the same RL model and can both converge while working on different optimization decisions. The number of edge server is not large and the resource placement problem is less complex than the task dispatching one, thus RL can learn faster in resource placement. The task dispatching initially has larger variation since the tasks are more sensitive to the user mobility. With more training data, the convergence becomes better.



(a) resource placement (b) task dispatching

Fig. 10. *Convergence of resource placement and task dispatching:* Training resource placement and task dispatching respectively.

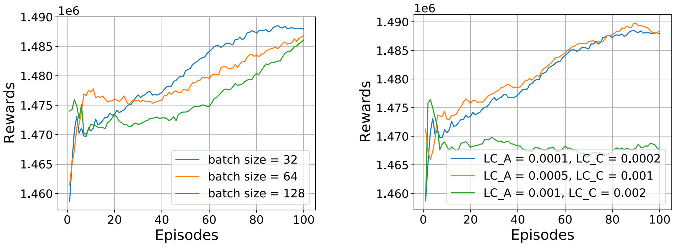
Convergence Performance Under Different Batch Sizes/Learning Rates. Finally, we investigate the convergence of our proposed deep RL method with different batch sizes and learning rates. Fig. 11a shows the performance of RL with a batch size at 32, 64 and 128. The batch size is used to determine the number of experience samples that need to be trained each step. We can find that the result of batch size at 32 gets higher rewards and converges earlier than the other two scenarios. Fig. 11b shows the performance of RL at different learning rates ε, which is used to control the update speed of the weight in the neural network. Here, we use different rates for the actor and critic (denoted by LC_A and LC_C respectively). Obviously, different learning rates will lead to different convergence results so we have to select an appropriate learning rate for our RL model.

6 Related Work

In this section, we briefly review some related works.

6.1 Resource Placement/Management

In this paper, we consider both data placement and service placement as resource placement in edge computing. Note that there are other types of resource management problems in edge computing, such as virtual network function placement [36], [37], virtual machine placement [38], [39], and cloudlet placement [23], [40], [41]. Next, we briefly review existing works on data placement and service placement.



(a) different batch size (b) different learning rate

Fig. 11. *Convergence of RL under different batch size and learning rate:* Running RL under 20 to 30 episodes with 3,000 steps per episode.

Data placement has been an important topic in distributed database/system [42], [43], peer-to-peer networking [44], [45], content delivery network[46], and cloud computing[47], [48]. While similar to all distributed systems, edge computing has its own characteristics [1], thus brings new data placement problems. Shao *et al.* [4] proposed a data replica placement strategy for processing the data-intensive IoT workflows in edge system which aims to minimize the data access costs while meeting the workflow's deadline constraint. The problem is modeled as a 0–1 integer programming problem and solved by an intelligent swarm optimization. Similarly, Lin *et al.* [5] also proposed a self-adaptive discrete particle swarm optimization algorithm to optimize the data transmission time when placing data for a scientific workflow in edge computing. Li *et al.* [10] investigated a joint optimization of data placement and task scheduling in edge computing to reduce the computation delay and response time. Their formulated optimization considers the value, transmission cost, and replacement cost of data blocks, which is then solved by a tabu search algorithm. Breitbach *et al.* [11] have also studied both data placement and task placement in edge computing by considering multiple context dimensions. For its data placement part, the proposed data management scheme adopts a context-aware replication, where the parameters of the replication strategy is tuned based on context information (such as data size, remaining storage, stability, application). Huang *et al.* [49] have studied caching fairness for data sharing in edge computing environments. They formulate the caching fairness problem, where fairness metrics take resources and wireless contention into consideration, and propose both approximation and distributed algorithms. Xie *et al.* [6] also studied the data-sharing problem and proposed a coordinate-based data indexing mechanism to enable the efficient data sharing in edge computing. It maps both switches and data indexes into a virtual space with associated coordinates, and then the index servers are selected for each data based on the virtual coordinates. Xie *et al.* [7] further extended their virtual-space method to handle data placement and retrieval in edge computing with an enhancement based on centrodial Voronoi tesselation to handle load balance among edge servers. Similarly, Wei *et al.* [8], [9] proposed another virtual-space based data placement strategy which takes the data popularity of data items into consideration during the virtual-space mapping, data placement and retrieval. There are solutions [50] for data management issues in edge computing as well.

Similar to data placement, service and resource placement in edge computing has been studied as well. Ouyang *et al.* [12] proposed an adaptive user-managed service placement algorithm to jointly optimize the latency and service migration cost. By formulating the service placement problem as a contextual Multi-armed Bandit problem, they proposed a Thompson-sampling based online learning algorithm to explore make adaptive service placement decisions. Xu *et al.* [13] studied the service caching in mobile edge clouds with multiple service providers completing for both computation and bandwidth resources, and proposed a distributed and stable game-theoretical caching mechanism for resource sharing among the network service providers. Pasteris *et al.* [14] also studied a multiple-service placement problem in a heterogeneous edge system and proposed an approximation algorithm placing multiple services to maximize the total reward. Meskar and Liang [15] proposed a resource allocation rule retaining fairness properties among multiple access points, while Zhang *et al.* [16] proposed a decentralized multi-provider resource allocation scheme to maximize the overall benefit of all providers. Resource placement has also been considered jointly with other design issues in edge networking and computing. For example, Kim *et al.* [17] designed a joint optimization of wireless MIMO signal design and network resource allocation to maximize energy efficiency in wireless D2D edge computing. Eshraghi and Liang [18] considered the joint optimization of computing/communication resource allocation and offloading decision of uncertain tasks in mobile edge networks.

6.2 Task Offloading/Dispatching

Task dispatching, as known as computation offloading [51], is also a critical problem in edge computing, and has been studied recently. In many cases, it is jointly considered with data/resource placement. For example, Breitbach *et al.* [11] also considered task placement in their context-aware solution, where task scheduler allocates tasks according to the current context and observes the state during runtime. Bi *et al.* [19] jointly studied a task offloading, service caching and resource allocation problem in a single edge server that assists a mobile user to perform a sequence of computation tasks. They formulated it as a mixed integer nonlinear programming (MINLP), and then solved it by separately optimizing the resource allocation and transforming the problem to integer linear program. Xu *et al.* [20] proposed an online algorithm to jointly optimize dynamic service caching and task offloading in edge-enabled dense cellular networks. Their solution is based on Lyapunov optimization and Gibbs sampling without knowing future information. Similarly, Poularakis *et al.* [21] investigated the joint service placement and request routing problem in edge-enabled multi-cell networks, and proposed a bi-criteria algorithm with randomized rounding technique that achieves approximation guarantees while violating the resource constraints in a bounded way. Ma *et al.* [22] studied cooperation among edge servers and investigated cooperative service caching and workload scheduling in mobile edge computing environment. They formulated the problem as MINLP and solved it by an iterative algorithm based on Gibbs sampling to achieve near-optimal performance. Yang *et al.* [23] proposed a Benders decomposition-based algorithm to jointly solve the cloudlet placement and task allocation problem while minimizing the total energy consumption.

However, most of these works consider a kind of joint optimization at a single timescale, thus may not handle the dynamic among tasks, resources, and computation facilities in the edge computing environment. Recently, Farhadi *et al.* [24] studied service placement and request scheduling problem in edge cloud environment for data-intensive applications and proposed a two-timescales framework to determine the near-optimal decision under specific constraints. You *et al.* [25] also studied a joint resource provision and workload distribution problem in mobile edge network. They formulated the problem as a nonlinear mixed-integer program to minimize the long-term cost, and proposed online learning based algorithms to solve the problem in two timescales. Our work is inspired by these works, but we consider different joint optimization with different network and edge settings. In addition, we also leverage deep reinforcement learning to solve the joint optimization.

6.3 Deep Reinforcement Learning

Reinforcement learning is one of the basic machine learning paradigms, which has been well-studied and widely applied in many fields. Recent advances in deep reinforcement learning (DRL) [28], [29], [31], [32], [33] have further enhanced its great capability to attack complex optimization problems in real dynamic systems, including edge computing.

Chen *et al.* [26] have studied the computation offloading problem in a dynamic time-varying network, and proposed a DQN-based solution to optimally offload the computation to base stations to maximize the long-term utility performance. Li *et al.* [52] considered the joint offloading and resource allocation in a multi-user edge system, where multiple users can perform computation offloading via wireless channels to an edge server. They proposed a DRL based scheme to tackle the optimization. Huang *et al.* [53] considered a binary task offloading in wireless edge system, and proposed a DRL based online offloading framework to adapts task offloading decisions and wireless resource allocations to the time-varying wireless channel conditions. Wang *et al.* [27] also proposed a DRL based resource allocation approach to adaptively allocate computing and network resources to reduce the average service time and balance resource usages under dynamic edge network. Ning *et al.* [54] solved the joint task scheduling and resource allocation optimization in vehicular edge system to maximize users' Quality of Experience (QoE) by using a two-sided matching scheme for task scheduling and a DRL approach for resource allocation respectively. Nath and Wu [55] considered the computation offloading and resource allocation in a cache-assisted edge system, and proposed a DDPG-based scheduling policy to minimize the long-term average cost including energy consumption, total delays and resource accessing cost. Meanwhile, Rahman *et al.* [56] also studied the joint problem of mode selection, resource allocation, and power allocation to minimize the total delay in the fog radio access networks using DRL methods. While many of these works adopt DRL to successfully optimize task scheduling/offloading and/or resource allocation, they usually use one DRL agent to learn the dynamic. In our work, our DRL method has been extended to work across two timescales.

7 Conclusion

In this study, we have investigated a joint resource placement and task dispatching problem in edge computing across different timescales. We proposed a two-stage optimization algorithm and a deep RL based algorithm to solve this joint optimization within a dynamic edge environment. Both methods can handle the variety of dynamics at two different timescales. Our simulation results showed that (1) both proposed methods perform much better than random and greedy algorithms; (2) the advantage of performing resource placement and task dispatching in different timescales is not only to reduce the placement cost but also does not require much future prediction of the task. The two proposed solutions have their own advantages. On one hand, RL needs more time to train the agent's model while OPT directly solves the optimization problem. On the other hand, RL is more efficient to handle dynamic environment and scales well with larger number of requests/servers.

In future, we plan to further enhance the proposed methods by also considering how to handle and recover from sudden server failure events, and apply the proposed ideas to other joint optimization issues in edge computing and beyond.

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, Oct. 2016.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Oct.–Dec. 2017.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1657–1681, Jul.–Sep. 2017.

[4] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for IoT workflows in collaborative edge and cloud environments," *Comput. Netw.*, vol. 148, pp. 46–59, 2019.

[5] B. Lin *et al.*, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Trans. Ind. Inform.*, vol. 15, no. 7, pp. 4254–4265, Jul. 2019.

[6] J. Xie, C. Qian, D. Guo, M. Wang, S. Shi, and H. Chen, "Efficient indexing mechanism for unstructured data sharing systems in edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 820–828.

[7] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, and H. Chen, "Efficient data placement and retrieval services in edge computing," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1029–1039.

[8] X. Wei, ABM M. Rahman, and Y. Wang, "Data placement strategies for data-intensive computing over edge clouds," in *Proc. IEEE IPCCC*, 2021.

[9] X. Wei and Y. Wang, "Popularity-based data placement with load balancing in edge computing," *IEEE Trans. Cloud Comput.*, early access, Jul. 13, 2021, doi: 10.1109/TCC.2021.3096467.

[10] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *J. Parallel Distrib. Comput.*, vol. 125, pp. 93–105, 2019.

[11] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2019, pp. 1–10.

[12] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in Proc. *IEEE Conf. Comput. Commun.*, 2019, pp. 1468–1476.

[13] Z. Xu, L. Zhou, S. C.-K. Chau, W. Liang, Q. Xia, and P. Zhou, "Collaborate or separate? Distributed service caching in mobile edge clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2066–2075.

[14] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 514–522.

[15] E. Meskar and B. Liang, "Fair multi-resource allocation in mobile edge computing with multiple access points," in *Proc. 21st Int. Symp. Theory, Algorithmic Found. Protocol Des. Mobile Netw. Mobile Comput.*, 2020, pp. 11–20.

[16] C. Zhang, H. Du, Q. Ye, C. Liu, and H. Yuan, “DMRA: A decentralized resource allocation scheme for Multi-SP mobile edge computing,” in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 390–398.

[17] J. Kim, T. Kim, M. Hashemi, C. G. Brinton, and D. J. Love, “Joint optimization of signal design and resource allocation in wireless D2D edge computing,” in *Proc. IEEE Conf. Comput. Commun.*, 2020, 2086–2095.

[18] N. Eshraghi and B. Liang, “Joint offloading decision and resource allocation with uncertain task computing requirement,” in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1414–1422.

[19] S. Bi, L. Huang, and Y.-J. A. Zhang, “Joint optimization of service caching placement and computation offloading in mobile edge computing systems,” *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4947–4963, Jul. 2020.

[20] J. Xu, L. Chen, and P. Zhou, “Joint service caching and task offloading for mobile edge computing in dense networks,” in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.

[21] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, “Joint service placement and request routing in multi-cell mobile edge computing networks,” in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 10–18.

[22] X. Ma, A. Zhou, S. Zhang, and S. Wang, “Cooperative service caching and workload scheduling in mobile edge computing,” in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 2076–2085.

[23] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, “Cloudlet placement and task allocation in mobile edge computing,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5853–5863, Jun. 2019.

[24] V. Farhadi *et al.*, “Service placement and request scheduling for data-intensive applications in edge clouds,” in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1279–1287.

[25] W. You, L. Jiao, S. Bhattacharya, and Y. Zhang, “Dynamic distributed edge resource provisioning via online learning across timescales,” in *Proc. 17th Annu. IEEE Int. Conf. Sens. Commun. Netw.*, 2020, pp. 1–9.

[26] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, “Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning,” *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[27] J. Wang, L. Zhao, J. Liu, and N. Kato, “Smart resource allocation for mobile edge computing: A deep reinforcement learning approach,” *IEEE Trans. Emerging Top. Comput.*, early access, Mar. 04, 2019, doi: 10.1109/TETC.2019.2902661.

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 387–395.

[29] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” 2015, *arXiv:1509.02971*.

[30] X. Wei and Y. Wang, “Joint resource placement and task dispatching in mobile edge computing across timescales,” in *Proc. IEEE/ACM 29th Int. Symp. Qual. Serv.*, 2021, pp. 1–6.

[31] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[32] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning,” in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[33] V. Mnih *et al.*, “Asynchronous methods for deep reinforcement learning,” in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[34] Google Cluster Data (clusterdata 2011 traces). Accessed: Sep. 20, 2020. [Online]. Available: https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md

[35] M. Han, Y. Lee, S. B. Moon, K. Jang, D. Lee, “CRAWDAD dataset kaist/wibro (v. 2008–06–04).” Accessed: Sep. 20, 2020. [Online]. Available: https://crawdad.org/kaist/wibro/20080604

[36] G. Sallam and B. Ji, “Joint placement and allocation of virtual network functions with budget and capacity constraints,” in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 523–531.

[37] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, “Delay-aware virtual network function placement and routing in edge clouds,” *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 445–459, Feb. 2021.

[38] D. Shen, J. Luo, F. Dong, and J. Zhang, “Virtco: Joint coflow scheduling and virtual machine placement in cloud data centers,” *Tsinghua Sci. Technol.*, vol. 24, no. 5, pp. 630–644, 2019.

[39] W. Zhang, X. Chen, and J. Jiang, “A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems,” *Tsinghua Sci. Technol.*, vol. 26, no. 1, pp. 95–111, 2021.

[40] L. Zhao, W. Sun, Y. Shi, and J. Liu, “Optimal placement of cloudlets for access delay minimization in SDN-based internet of things networks,” *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1334–1344, Apr. 2018.

[41] L. Zhao and J. Liu, Y. Shi, and J. Liu, “Optimal placement of virtual machines for supporting multiple applications in mobile edge networks,” *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6533–6545, Jul. 2018.

[42] P. Chundi, D. J. Rosenkrantz, and S. S. Ravi, “Deferred updates and data placement in distributed databases,” in *Proc. 12th Int. Conf. Data Eng.*, 1996, pp. 469–476.

[43] A. Brinkmann, K. Salzwedel, and C. Scheideler, “Efficient, distributed data placement strategies for storage area networks,” in *Proc. 12th Annu. ACM Symp. Parallel Algorithms Architectures*, 2000, pp. 119–128.

[44] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, “Search and replication in unstructured peer-to-peer networks,” in *Proc. 16th Int. Conf. Supercomputing*, 2002, pp. 84–95.

[45] J. Tian, Z. Yang, and Y. Dai, “A data placement scheme with time-related model for P2P storages,” in *Proc. 7th IEEE Int. Conf. Peer-to-Peer Comput.*, 2007, pp. 151–158.

[46] M. Drwal and J. Józefczyk, “Decentralized approximation algorithm for data placement problem in content delivery networks,” in *Proc. Doctoral Conf. Comput. Elect. Ind. Syst.*, 2012, pp. 85–92.

[47] T. Wang, S. Yao, Z. Xu, and S. Jia, “DCCP: An effective data placement strategy for data-intensive computations in distributed cloud computing systems,” *J. Supercomput.*, vol. 72, no. 7, pp. 2537–2564, 2016.

[48] D. Yuan, Y. Yang, X. Liu, and J. Chen, “A data placement strategy in scientific cloud workflows,” *Future Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1200–1214, 2010.

[49] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, “Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments,” *IEEE Trans. Mobile Comput.*, vol. 19, no. 4, pp. 852–864, Apr. 2020.

[50] Q. Meng, K. Wang, X. He, and M. Guo, “QoE-driven big data management in pervasive edge computing environment,” *Big Data Mining Analytics*, vol. 1, no. 3, pp. 222–233, 2018.

[51] R. Bi, Q. Liu, J. Ren, and G. Tan, “Utility aware offloading for mobile-edge computing,” *Tsinghua Sci. Technol.*, vol. 26, no. 2, pp. 239–250, 2021.

[52] J. Li, H. Gao, T. Lv, and Y. Lu, “Deep reinforcement learning based computation offloading and resource allocation for MEC,” in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2018, pp. 1–6.

[53] L. Huang, S. Bi, and Y.-J. A. Zhang, “Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks,” *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[54] Z. Ning, P. Dong, X. Wang, J. J. P. C. Rodrigues, and F. Xia, “Deep reinforcement learning for vehicular edge computing: An intelligent offloading system,” *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, pp. 1–24, 2019.

[55] S. Nath and J. Wu, “Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems,” *Intell. Converged Netw.*, vol. 1, no. 2, pp. 181–198, Sep. 2020.

[56] G. M. S. Rahman, T. Dang, and M. Ahmed, “Deep reinforcement learning based computation offloading and resource allocation for low-latency fog radio access networks,” *Intell. Converged Netw.*, vol. 1, no. 3, pp. 243–257, 2020.

Xinliang Wei (Student Member, IEEE) received the BE and MS degrees in software engineering from SUN Yat-sen University, Guangzhou, China, in 2014 and 2016, respectively. He is currently working toward the PhD degree with the Department of Computer and Information Sciences, Temple University. His research interests include edge computing, Internet of Things, and computer graphs.

A B M Mohaimenur Rahman received the undergraduate degree from the Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology in 2017. He is currently working toward the PhD degree with the Department of Computer Science, University of North Carolina at Charlotte. His research interests include smart sensing, wireless networks, and mobile computing.

Dazhao Cheng (Member, IEEE) received the BS degree in electronic engineering from the Hefei University of Technology in 2006, the MS degree from the University of Science and Technology of China in 2009, and the PhD degree from the University of Colorado, Colorado Springs, in 2016. He is currently an assistant professor with the Department of Computer Science, University of North Carolina at Charlotte. His research interests include big data and cloud computing. He is a member of the ACM.

Yu Wang (Fellow, IEEE) received the BEng and MEng degrees in computer science from Tsinghua University and the PhD degree from the Illinois Institute of Technology. He is currently a professor with the Department of Computer and Information Sciences, Temple University. He has authored or coauthored more than 200 papers in peer reviewed journals and conferences. His research interests include wireless networks, smart sensing, and mobile computing. He was the general chair, the program chair, and a program committee member for many international conferences, including IEEE IPCCC, ACM MobiHoc, IEEE INFOCOM, IEEE GLOBECOM, and IEEE ICC. He was an editorial board member for several international journals, including the *IEEE Transactions on Parallel and Distributed Systems* and *IEEE Transactions on Cloud Computing*. He was the recipient of Ralph E. Powe Junior Faculty Enhancement awards from Oak Ridge Associated Universities in 2006 and the Outstanding Faculty Research Award from the College of Computing and Informatics, University of North Carolina at Charlotte in 2008. He is an ACM distinguished member.