Dynamic Task Allocation and Service Migration in Edge-Cloud IoT System Based on Deep Reinforcement Learning

Yan Chen, Yanjing Sun, *Member, IEEE*, Chenyang Wang, *Member, IEEE*, and Tarik Taleb, *Senior Member, IEEE*

***Abstract*—Edge computing (EC) extends the ability of cloud computing to the network edge to support diverse resource-sensitive and performance-sensitive IoT applications. However, due to the limited capacity of edge servers (ESs) and the dynamic computing requirements, the system needs to dynamically update the task allocation policy according to real-time system states. Service migration is essential to ensure service continuity when implementing dynamic task allocation. Therefore, this article investigates the long-term dynamic task allocation and service migration (DTASM) problem in edge-cloud IoT systems where users' computing requirements and mobility change over time. The DTASM problem is formulated to achieve the long-term performance of minimizing the load forwarded to the cloud while fulfilling the seamless migration constraint and the latency constraint at each time of implementing the DTASM decision. First, the DTASM problem is divided into two subproblems: 1) the user selection problem on each ES and 2) the system task allocation problem. Then, the DTASM problem is formulated as a Markov decision process (MDP) and an approach based on deep reinforcement learning (DRL) is proposed. To tackle the challenge of vast discrete action spaces for DTASM task allocation in the system with a mass of IoT users, a training architecture based on the twin-delayed deep deterministic policy gradient (DDPG) is employed. Meanwhile, each action is divided into a differentiable action for policy training and one mapped action for implementation in the IoT system. Simulation results demonstrate that the proposed DRL-based approach obtains the long-term optimal system performance compared to other benchmarks while satisfying seamless service migration.**

***Index Terms*—Deep deterministic policy gradient (DDPG), deep reinforcement learning (DRL), dynamic task allocation, edge computing (EC), Internet of Things, seamless service migration.**

Manuscript received 28 September 2021; revised 2 December 2021 and 24 January 2022; accepted 20 March 2022. Date of publication 4 April 2022; date of current version 7 September 2022. This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant 2020ZDPY0304; in part by the Chinese Government Scholarship awarded by China Scholarship Council under Grant 202006420096; in part by the European Union's Horizon 2020 Research and Innovation Program under the MonB5G Project under Grant 871780; in part by the Academy of Finland 6Genesis Project under Grant 318927; and in part by IDEA-MILL under Grant 335936. *(Corresponding author: Yanjing Sun.)*

Yan Chen and Yanjing Sun are with the School of Information and Control Engineering, China University of Mining and Technology, Xuzhou 221116, Jiangsu, China (e-mail: chyan@cumt.edu.cn; yjsun@cumt.edu.cn).

Chenyang Wang is with the College of Intelligence and Computing, Tianjin University, Tianjin 300072, China (e-mail: chenyangwang@tju.edu.cn).

Tarik Taleb is with the Center of Wireless Communications, University of Oulu, 90570 Oulu, Finland, and also with the Department of Computer and Information Security, Sejong University, Seoul 05006, South Korea (e-mail: tarik.taleb@oulu.fi).

Digital Object Identifier 10.1109/JIOT.2022.3164441

I. Introduction

EDGE computing (EC) can support both performance-sensitive and resource-sensitive emerging IoT applications by providing powerful computing capacity and deploying necessary edge service functions (SFs) on edge servers (ESs) in the proximity of IoT users, making it one of the key technologies to extend current cloud-based systems to support various future IoT scenarios [1], [2]. Moreover, EC is becoming essential to support the much-needed service-aware and intelligent management of networks and services in the upcoming mobile communication systems (e.g., B5G and 6G) [3]. However, ESs are generally resource-constrained due to the limitations of working environments, such as space and power supply. Thus, nearby ESs can cooperate to enhance the capacity of edge systems, and provide performance-guaranteed EC services for various IoT applications [4]–[7].

Intelligent applications in future IoT systems are generally both resource-sensitive and performance-sensitive (i.e., these applications have strict performance requirements, and their execution consumes large amounts of resources), making their Quality of Service (QoS) sensitive to the allocated resources and the distance to the ESs on which their tasks are processed [8], [9]. In IoT systems, the computing requirements of IoT users are usually change dynamically due to mobility and time-varying working conditions. For example, a higher communication delay results when a user moves to another radio access network (RAN) node far away from the original ES processing its computing tasks. Both the communication delay and the computing delay increase when the computing load (e.g., computing tasks) increases. As a result, the QoS of users may decrease if the system follows a stationary task allocation policy. Therefore, to ensure the QoS of users and maintain resource utilization, it is necessary to orchestrate the task allocation policy dynamically according to real-time system states in such an edge-cloud system with heterogeneous resource capacities and dynamic computing requirements.

The task processing of edge IoT applications requires computing capacity and relies on the support of the underlying environments and state contexts maintained by the hosting operating system (OS) and SFs [10]–[12]. Therefore, in the edge-cloud system that requires dynamic task allocation, a user's state context should be migrated apart from switching task traffic from the current server to the target server

2327-4662 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

when the task processing is reallocated. The migration of the user-dependent state context is known as service migration, i.e., resuming service for the user on the target ES with the migrated context. Although dynamic task allocation and service migration (DTASM) can accommodate users' dynamic computing requirements and mobility to maintain stable and reliable QoS, overheads are also definitely introduced when implementing DTASM operations, e.g., the service downtime during which users' EC service is interrupted, the transmission cost, and the service migration management cost. Besides, executing more service migrations would generally result in higher overheads. Overheads introduced by service migration can be dramatically reduced by only executing the necessary service migrations. Service continuity (i.e., ensuring a contracted service downtime) is vital for IoT applications [13], [14], especially in IoT scenarios with strict reliability demands, e.g., Industrial IoTs. Service migration for each user in those systems must not result in a service downtime that exceeds a precontracted threshold as severe accidents may result otherwise. Therefore, providing seamless service migration is essential for the EC service controller when generating new task allocation and implementing DTASM.

Based on the above backgrounds, this article investigates the DTASM problem in an edge-cloud IoT system with dynamic computing requirements and user mobility. Unlike existing works, we investigate DTASM in an edge-cloud system with large numbers of IoT users and also comprehensively consider the seamless service migration constraint and the service latency constraint. Moreover, a deep reinforcement learning (DRL) approach based on a twin-delayed deep deterministic policy gradient (DDPG) is proposed to address the difficult DTASM problem under limited prior knowledge and the challenge of vast discrete action space. The main contributions of this article are summarized as follows.

1) We investigate the DTASM problem in a heterogeneous edge-cloud IoT system requiring dynamic task allocation and in which seamless service migration is critical to ensure service continuity. As the system EC service performance only depends on real-time system state and implemented policy, the long-term DTASM problem is formulated as optimizing the task allocation policy, which can minimize the load forwarded to the cloud server under any observed system state while satisfying constraints, including seamless migration, service latency, and computing capacity.
2) We first decompose the problem into user selection and task allocation subproblems. Then, we formulate the DTASM problem as a Markov decision process (MDP) and propose a DRL-based approach since it can learn to adapt to systems without prior knowledge. To handle the enormous DTASM action space challenges in real IoT systems with numerous IoT users, we employ a twin-delayed DDPG architecture to train the agent. To exploit the DDPG method in the DTASM problem with discrete action space, we split a task allocation action into a differentiable action for agent training and a mapped action for implementation.
3) Extensive simulations are conducted to evaluate the proposed DRL-based DTASM approach. The performance is evaluated in terms of the load forwarded to the cloud, service downtime, the number of migrated users, and the number of migration-failed users. We also investigate the convergence of the DRL-based algorithm over different discount factors. The impact of the number of IoT users and seamless migration constraints is also studied. Simulation results demonstrate that our DRL-based algorithm can maintain a stable QoS for implementing DTASM and performs significantly better than other benchmarks.

The remainder of this article is organized as follows. Related works are discussed in Section II. Section III details the system model and formulates the DTASM problem. The proposed DRL-based DTASM approach is illustrated in Section IV. Simulations are conducted and the results are discussed in Section V. This article concludes in Section VI.

II. Related Works

The collaborative task allocation problem in the EC community has been explored in different studies, and some recent works consider the service migration problem.

A. Collaborative Task Allocation

Collaborative task allocation in edge-cloud systems has been explored with keywords, such as task offloading, server selection, and service placement, which indicates allocating the computing tasks of each user to an appropriate ES in the system. Meanwhile, a cloud server is generally deployed to process overloaded tasks offloaded from the edge system [15].

Some previous works have studied one-shot operations based on traditional approaches, such as convex optimization and heuristic algorithms, to obtain optimal or near-optimal performance for a studied stable system. In [16], a joint task offloading and resource allocation problem was investigated. The NP-hard optimization problem was addressed separately by a designed heuristic task offloading algorithm and a resource allocation solution based on convex and quasi-convex optimization. The joint task offloading decision and resource allocation was formulated as a mixed-integer nonlinear programming problem to minimize the energy consumption of users, and algorithms based on genetic algorithm and particle swarm optimization are proposed in [17] and [18]. In [19], a collaborative task offloading scheme based on game theory was proposed for an edge-cloud system with a hybrid fiber-wireless network to minimize the energy consumption of mobile devices. In [20], based on game theory, a collaborative computation offloading and resource allocation optimization algorithm was designed to maximize the designed utility of system latency and cost.

Recent works have also studied dynamic task allocation algorithms, and DRL and its variations are usually employed in these works to deal with this challenging problem. An optimal ES selection problem in the system with heterogeneous RANs and ESs was studied in [8] to minimize the system service latency, which was formulated as an MDP and

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

addressed by a value iteration algorithm. A DQN-based edge service placement algorithm was proposed to minimize the long-term average response time of all users [21]. A DRL-based task offloading algorithm was proposed [22], in which the dynamic load in the edge network was considered, and a double-Q network method was employed to obtain a policy that can minimize the long-term cost.

However, state contexts packaged with user-specific information and the real-time states of tasks are necessary for task processing. For a long-term running edge-cloud IoT system, dynamic task allocation is coupled with the service migration that synchronizes users' state contexts to maintain service continuity. Moreover, the service manager cannot ignore the service migration procedure in edge-cloud systems since it significantly impacts the overall QoS.

B. Service Migration

Some previous works have studied the service migration from the view of a single user to ensure its QoS. Zhang *et al.* [23] proposed a QoS-aware active edge service migration method, which enables the service migration to be actively performed by one user based on MDP incorporated with network and server state. In [9], a single-user system was considered and a DQN-based service migration algorithm was designed to make the service migration strategies. The work of [24] introduced a service migration method for one user based on DQN to maximize the designed reward. In their work, the state is the real-time distance from the user to the ES, which processes its tasks, and the reward is the difference between a constant maximum achievable QoS and migration cost. Sun *et al.* [25] proposed a user-centric service migration algorithm in which a mobile user made migration decisions based on its own QoS requirement. The algorithm was designed to minimize the total delay of all computing tasks and to satisfy the total energy consumption constraint of the user by considering its real-time distances to all ESs. However, IoT systems generally support multiple users simultaneously. Thus, the migration of a user may interfere with the users on the target ES.

In [26], a service migration and resource allocation algorithm based on the relaxation-and-rounding approach was proposed to maximize the weighted reward between system offloading rate and migration cost. However, they considered a constant migration cost for each user. Furthermore, the dynamic nature of the system state and necessary environment dependency of edge applications were not considered. Besides, the computing capacity of an ES is defined as the number of users it can support rather than the actual computing capacity (e.g., CPU cycles) that is heterogeneous among users and changes dynamically over time. Gao *et al.* [27] proposed a two-phase dynamic service placement algorithm based on matching and game theory and aiming to minimize the sum of communication, computing latency, and migration cost. In [28] and [29], dynamic edge service migration algorithms were proposed based on MDP, in which the service migration is based on the real-time location of users to minimize the long-term discounted migration cost. However, the dynamic computing load in real multiuser IoT systems has not been investigated in the above works.

Similarly, deep Q-learning-based service migration approaches were proposed to maximize the weighted sum reward of load capacity and service migration cost [30], [31]. The algorithm selects the optimal action with the maximum Q-value from all candidates at each time. However, such approaches can only be used in systems with few users since estimating and comparing Q-values of all candidate actions are difficult in systems with a large number of IoT users. A dynamic service placement and service migration problem was investigated in [32]. The author assumed the computing capacity allocation among IoT users to be homogeneous, and the migration constraint was set as long-term energy consumption in their work. Then, the problem of seeking long-term average system service latency was transformed into a Lyapunov optimization problem. Different from their work, we consider the dynamic capacity requirements based on heterogeneous QoS requirements of users and consider the seamless migration constraint.

Different from the above works, we investigate the DTASM problem in an edge-cloud IoT system with a mass of IoT users. Besides, we comprehensively take the heterogeneous and dynamic computing requirements and user mobility into consideration. Meanwhile, the seamless service migration constraint at each time of implementing DTASM is considered as well. In addition, an approach based on the twin-delayed DDPG method is proposed to address the enormous discrete action space challenge that is hardly addressed by traditional DQN-based methods.

III. System Model and Problem Formulation

A. System Model

As shown in Fig. 1, we consider an edge-cloud collaborative IoT system, where a set of RAN nodes can support communications of a set of IoT users $\mathcal{U} = \{1, 2, \ldots, U\}$ in the system. Meanwhile, each RAN node (i.e., base station) is associated with an ES that can support the task processing of kinds of intelligent IoT applications $\mathcal{F} = \{1, 2, \ldots, F\}$ installed on IoT users. We use $\mathcal{H} = \{1, 2, \ldots, H\}$ to represent the set of ESs and RAN nodes. We assume that each IoT user can only be installed with one application, and the type of the application is one included in $\mathcal{F}$. Those IoT users running multiple applications in the real world can be virtualized into multiple users executing one application. Besides, a cloud server $\mathcal{C}$ is deployed to manage the whole IoT system, and application tasks that cannot be satisfied in the edge network are forwarded to the cloud server for processing. For simplification, we use $\mathcal{H}' = \mathcal{H} \cup \mathcal{C}$ to represent all servers in the system.

Generally, every IoT user is connected to one RAN node at any time, and its tasks are offloaded to the access point of the RAN via the communication link established between them (D2R). Then, its tasks are further forwarded from its currently connected RAN to one ES for processing. The tasks and results are transmitted via the communication link between related RANs (R2R) when a user's tasks are allocated to be processed by another ES that is not directly associated

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

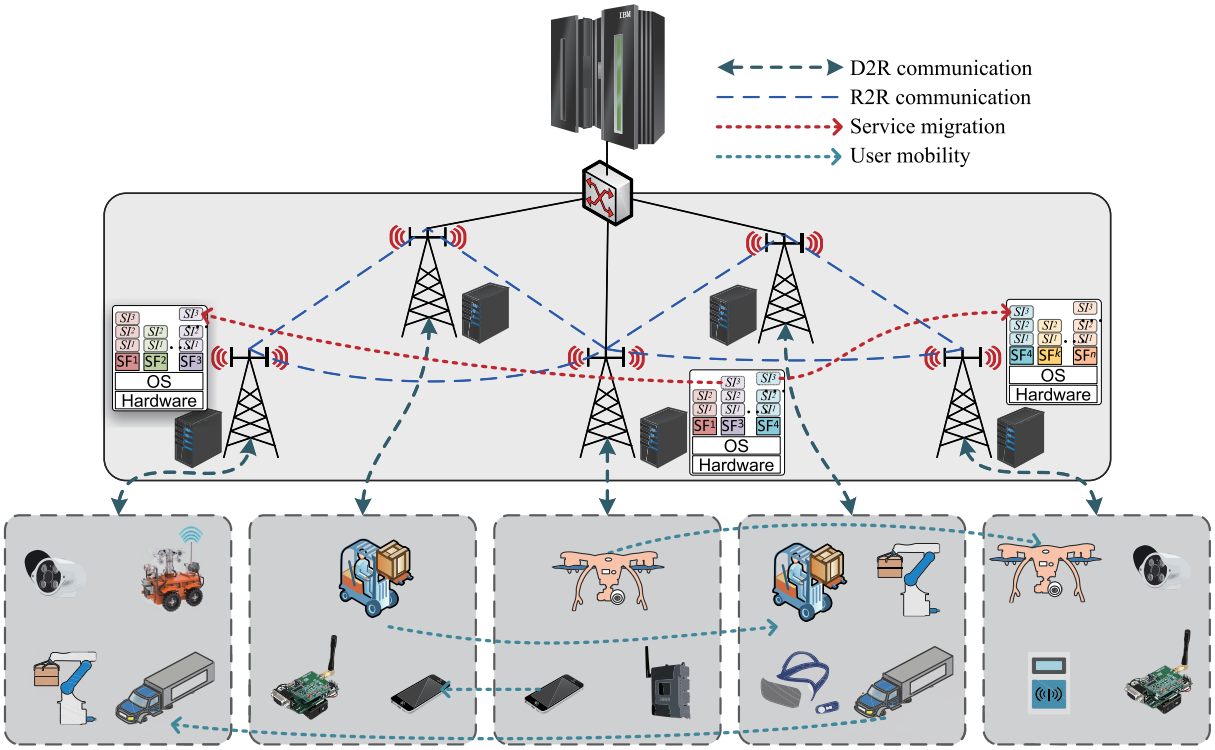


Fig. 1. Edge-cloud IoT system requiring DTASM.

with the RAN to which the user is currently connected. To satisfy the requirement of isolation and easy management of the running environment of the performance-sensitive and resource-sensitive IoT applications [33], a three-layer service architecture is considered. On every ES, a host OS is running to provide the base support environment for task processing of various applications allocated to it. On top of the OS, SFs are activated to satisfy different applications' dedicated tasks processing requirements. Meanwhile, each SF maintains stateless contexts not relevant to a particular user, such as generic databases and codes. Each SF can only provide service for the users who perform the corresponding type of application. When a user is allocated to one SF activated on one ES for task processing, the SF will create a dedicated running service instance (SI) for the user to satisfy the isolation requirement. Besides, each SI maintains the stateful context encapsulated with user-specific information, such as the real-time running status and private context that is needed in the future.

B. Dynamic Task Allocation and Service Migration

When a user's computing task is reallocated to be processed on a different ES, the necessary stateless and stateful contexts should be ready on the target ES before resuming its task processing. Service migration of the user indicates synchronizing its related contexts from the source server to the target ES before resuming its task processing on the destination ES, which introduces considerable migration costs, such as service interruption and network congestion. However, the stateless context can be prestored on the disk of each ES and activated on demand because the disk resources of the server

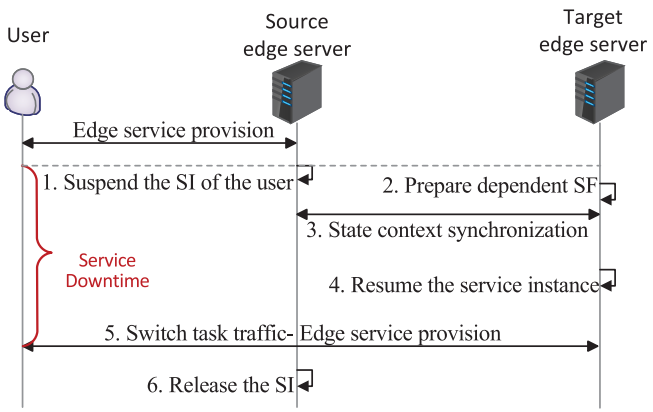


Fig. 2. Necessary procedures of edge service migration.

are usually sufficient nowadays [12], [34]. Then, only stateful contexts maintained by SIs need to be synchronized during service migrations (as shown in Fig. 1), which can significantly reduce service migration costs and is more suitable to support edge applications [35]. As shown in Fig. 2, a service migration performs the following procedures in general [10]. First, suspending the current running SI and packaging the real-time state context. Then, the state context is transmitted to the target ES via the R2R link. Finally, the target ES restores the SI from its checkpoint after the activation of the required SF and provides service in the following time. Meanwhile, the task traffic is switched to the target server.

At each time t, every user offloads its tasks to the corresponding SI instanced on a nearby ES via current associated RAN according to current task allocation result.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

For $\forall u \in \mathcal{U}$, we use a row vector of binary indicators $\mathbf{x}_u(t) = [x_u^1(t), x_u^2(t), \ldots, x_u^{|\mathcal{H}'|}(t)]$ to represent its task allocation, where $x_u^h(t) = 1$ indicates the task of u is allocated to be processed by its SI placed on server h at time t and $x_u^h(t) = 0$, otherwise. Meanwhile, the $\mathbf{x}_u(t)$ should satisfy

$$\mathbf{C1}: \sum_{h \in \mathcal{H}'} x_u^h(t) = 1 \quad \forall u \in \mathcal{U} \quad \forall t = 1, \ldots, +\infty \tag{1}$$

which indicates that each user must be allocated to only one server to process its tasks.

The task reallocation and service migration can be triggered when the controller detects any significant QoS degradation or new requirement of IoT users [23], which is beyond the focus of this article. We set the system state to be updated at each decision step, and the corresponding policy is generated according to the state. We use $t+1$ to represent the next step of the current time slot t when the system updates the task allocation policy. For any user u, the current location of its SI can be represented by

$$\mathcal{X}_u(t) = \arg\max_h \mathbf{x}_u(t), h \in \mathcal{H}'. \tag{2}$$

Then, $|\mathcal{X}_u(t+1) - \mathcal{X}_u(t)| > 0$ indicates u is allocated to another server, and its SI is migrated from the source server $\mathcal{X}_u(t)$ to the target server $\mathcal{X}_u(t+1)$. Otherwise, its SI continues to work on the current server to process its tasks without migration.

C. QoS Model and Implementation of DTASM

By comparing the current task allocation policy and the newly generated task allocation policy, the system can determine whether the SI of a user needs to be migrated or not. For a user whose service will be migrated, according to Fig. 2, the service downtime of the user starts from when its SI is suspended on the source server till when its SI gets resumed on the target server, which is determined by four parts. The service suspending time depends on the state context volume and allocated processing capacity by the corresponding SF. Meanwhile, it depends on the processing intensity of corresponding operations, i.e., the processing capacity required by processing per bit state context. The environment preparation (i.e., preparation of dependent SF) time depends on whether the SF has been activated on the target ES, the volume of the stateless context, and the application type (i.e., the required processing capacity and the processing intensity). The state context synchronization time depends on the data volume of the state context and the bandwidth between the source server and the target server. It is worth noting that the environment preparation can be executed in parallel with state context synchronization. Last, the factors determining the service resuming time are similar to those influencing the service suspending time but may require different processing intensity. Therefore, the service downtime ($d_u^{sd}(t)$) of a user u at time t is significantly affected by the selection of the target server, and can be written as

$$d_u^{sd}(t) = d_u^s(t) + \max\left\{d_u^{sy}(t), d_u^{sf}(t)\right\} + d_u^r(t) \tag{3}$$

where $d_u^s(t)$ is the time cost for suspending the SI of u at time t, $d_u^{sy}(t)$ is the time for synchronizing state context from the source server to the target ES, $d_u^{sf}(t)$ is the time for preparing the environment of SF required by the migrated SI of u, and $d_u^r(t)$ represents the time for resuming the SI on the target ES. We assume the task traffic can be switched as soon as the SI is resumed on the target ES.

The functionalities like the checkpoint can be used for suspending and resuming SIs [11], [36], [37]. We assume that the functionality is embedded in every SF and with fixed processing ability. Meanwhile, we employ a generally used task processing model [5], [25], [38] since suspending and resuming the SI of a user are also a kind of task processing. The time for suspending and resuming an SI depends on the data volume of the state context, processing intensity requirements, and the corresponding available processing ability. Thus, the SI suspending time of u can be expressed as

$$d_u^s(t) = \frac{V_u^m(t)\rho_u^s}{\beta_u P_u^{SF}} \tag{4}$$

and the SI resuming time can be written as

$$d_u^r(t) = \frac{V_u^m(t)\rho_u^r}{\beta_u P_u^{SF}} \tag{5}$$

where V_u^m is the volume (i.e., data size in terms of bits) of the state context, P_u^{SF} is the maximum computing capacity of the SF required by u, and β_u is the ratio of computing capacity allocated to the checkpoint functionality embedded in the SF. ρ_u^s and ρ_u^r are the corresponding processing intensities required by suspending and resuming the SI of u (i.e., the computing capacity in terms of CPU cycles required by processing per bit state context). In this article, we set that P_u^{SF}, β_u, ρ_u^s, and ρ_u^r depend on the type of application performed by u, i.e., the values of these parameters are homogeneous for the same kind of applications.

The time for synchronizing state context is the communication latency, including the transmission latency decided by the data volume and the bandwidth between the source server and the target ES, as well as the propagation latency from the source server to the target server [5], [25], [32], [38], i.e.,

$$d_u^{sy}(t) = \frac{V_u^m(t)}{B_{\mathcal{X}_u(t),\mathcal{X}_u(t+1)}} + \delta_{\mathcal{X}_u(t),\mathcal{X}_u(t+1)} \tag{6}$$

where $B_{\mathcal{X}_u(t),\mathcal{X}_u(t+1)}$ and $\delta_{\mathcal{X}_u(t),\mathcal{X}_u(t+1)}$ represent the bandwidth and the propagation latency between the RAN node $\mathcal{X}_u(t)$ and the RAN node $\mathcal{X}_u(t+1)$, respectively.

The time for preparing the environment of SF required by u is considered under two conditions. If the required SF has already been activated on the target server, there is no need to start a new SF, and the environment preparation time is negligible. Otherwise, the target ES activates a new SF and adapts it to support the migrated SIs. For the sake of simplicity, we assume the SF preparation time for each kind of application is constant, and SFs are notified to be activated from the start of the context synchronization by short notification frames.

The edge service controller should consider service downtime when making a new task allocation policy according to the current system state to maintain service continuity. In other words, the controller should try as much as possible to ensure that the service downtime of each user, during a service migration, is less than a precontracted seamless migration constraint

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

value. We set the value as ξ in this article. Then, the service downtime of each user should not be greater than ξ whenever its service is migrated to another server, i.e.,

$$\mathbf{C2}: d_u^{sd}(t) \leq \xi \quad \forall u \in \mathcal{U} \quad \forall t = 1, \ldots, +\infty. \tag{7}$$

To avoid reallocation and ensure QoS for all users, the central cloud server can provide assistance to serve migration-failed users whose service downtime constraints cannot be satisfied while implementing a DTASM process. Besides, the services of users whose task processing breaks any constraint (e.g., latency and resource) are also reforwarded to the cloud server, which will be discussed later in this section. After receiving the state context from the source ES, the target ES can obtain the migration time and resource requirement and determine whether a user's service should be reforwarded. Therefore, there are two synchronization steps for reforwarded users, i.e., from the source ES to the target ES and then to the cloud server. The tasks and SI of these users are reforwarded and migrated to the cloud server via links between each RAN node and the cloud server. We set the cloud server always to activate all kinds of SFs and have enough computing capacity to suspend and resume the SIs of users quickly. The service downtime of a user reforwarded from the target ES to the cloud server can be expressed as

$$d_u^{sd}(t) = d_u^{s}(t) + d_u^{sy1}(t) + d_u^{sy2}(t) \tag{8}$$

where $d_u^{sy1}(t)$ is the state context synchronization time from the source server to the target ES when implementing DTASM, and $d_u^{sy2}(t)$ is the state context synchronization time from the target ES to the cloud server. We assume the suspending and resuming of SI performed on the cloud server consumes negligible time. Supposing that the task of a user is currently being processed on the cloud server, however, the SI of the user needs to be reforwarded from the target ES to the cloud caused by breaking the latency or resource constraint when implementing a new task allocation policy (which will be discussed later). In this case (i.e., a user's task is currently being processed on the cloud server and is reallocated to the cloud server after executing a new DTASM action), we set the service downtime of the user as 0. The reason is that the cloud server possesses sufficient computing capacity and can maintain task processing until starting the service migration process. Then, the user whose task is currently processed on the cloud server and is reallocated to the cloud server after DTASM operation does not need to stop its service.

In addition to the seamless service migration constraint, the QoS of EC services is the primary object pursued by edge-cloud systems. In this article, we consider service latency as the QoS requirement. The total EC service latency of a user includes communication latency and computing latency. We use $y_u(t) \in \mathcal{H}$ to indicate the physical location of u at time t, i.e., the RAN node to which u is currently connected. The communication latency of u can be expressed as [25]

$$d_u^{c}(t) = \frac{V_u^t(t) + V_u^r(t)}{B_{y_u(t),\mathcal{X}_u(t)}} + 2\delta_{y_u(t),\mathcal{X}_u(t)} \tag{9}$$

where $V_u^t(t)$ and $V_u^r(t)$ represent the volume (i.e., data size in terms of bits) of computing tasks and results of u at time t. $\delta_{y_u(t),\mathcal{X}_u(t)}$ represent the propagation between the RAN node $y_u(t)$ and the RAN node $\mathcal{X}_u(t)$. We assume that the RAN nodes can always provide ultrareliable low-latency communications within RAN with advanced communication technologies. Thus, the communication latency from a user to the associated RAN is ignored in this article.

We employ a widely used EC model whereby the computing latency depends on the computing capacity requirement and the allocated computing capacity [25], i.e.,

$$d_u^{p}(t) = \frac{\kappa_u(t) V_u^t(t)}{P_u^c(t)} \tag{10}$$

where $\kappa_u(t)$ is the computing intensity required by processing tasks from user u at time t, i.e., the computing capacity in terms of the CPU cycles required for processing (per bit) its computing task. $P_u^c(t)$ is the computing capacity in terms of CPU cycles allocated to u.

To provide guaranteed QoS, the service latency of any u should not exceed its predefined constraint (τ_u), i.e.,

$$\mathbf{C3}: d_u^{c}(t) + d_u^{p}(t) \leq \tau_u \quad \forall u \in \mathcal{U} \quad \forall t = 1, \ldots, +\infty. \tag{11}$$

As the capacity of each ES is limited, we set the computing capacity allocation in this article according to the minimum requirement of satisfying the EC service latency constraint. Therefore, according to (9)–(11), the minimum computing capacity required by u under a given task allocation policy is

$$P_u^{rc}(t) = \frac{\kappa_u(t) V_u^t(t)}{\tau_u - d_u^c(t)}. \tag{12}$$

Under these settings, there are still several cases that the latency constraint cannot be satisfied. First, a user's tasks are allocated to be processed by an ES far away from the RAN that the user is connected to, resulting in the communication latency exceeding the latency constraint. Second, the communication latency does not exceed the service latency constraint but is extremely high, resulting in a huge amount of capacity requirement that exceeds the ES's maximum equipped computing capacity. Last, the total computing capacity required by all users allocated to an ES exceeds the maximum capacity of the ES, which makes some of these users cannot be satisfied. Thus, the computing capacity allocation on an ES h must meet the constraint of its maximum capacity ($\mathcal{P}^h$), i.e.,

$$\mathbf{C4}: \sum_{u \in \mathcal{U}} x_u^h P_u^c(t) \leq \mathcal{P}^h \quad \forall h \in \mathcal{H}. \tag{13}$$

For simplification, we assume that the capacity for running SFs has been prereserved on every ES, and we only consider the capacity allocated to SIs. To maintain QoS, we set the tasks of users whose task processing breaks its latency constraint or required computing capacity cannot be satisfied to be reforwarded to the cloud as the computing capacity of the cloud server is generally sufficient.

D. Problem Formulation

Although the cloud server can process tasks of users that cannot be satisfied by ESs when implementing DTASM, the cloud server usually supports various applications in the whole system. Its capacity also has an upper bound in real systems.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

TABLE I
LIST OF DEFINED NOTATIONS

Symbol	Definition
$\mathcal{U}$	Set of IoT users
$\mathcal{F}$	Set of application kinds
$\mathcal{H}$	Set of RAN and edge servers
$\mathcal{H}'$	Set of servers including cloud server
$B_{i,j}$	Bandwidth between RAN i and RAN j
$\delta_{i,j}$	Propagation delay between RAN i and RAN j
F_u	Kind of application performed by u
$\mathbf{x}_u(t)$	Vector that indicates the task allocation of u
$x_u^h(t)$	Binary indicator represents if u is allocated to h
$\mathcal{X}_u(t)$	The Server on which SI of u is instanced
$y_u(t)$	RAN to which user u is connected at time t
$V_u^m(t)$	Volume of state context of user at time t
$d_u^{sd}(t)$	Service migration downtime of u at time t
$d_u^s(t)$	SI suspending time of u at time t
d_u^{sy}	Context synchronization time of u at time t
d_u^{sf}	Time for preparing SF for u at time t
$d_u^r(t)$	Time for resuming SI of u at time t
P_u^{SF}	Computing capacity of the SF required by u
β_u	Ratio of computing capacity allocated to checkpoint function by the SF required by u
ρ_u^s	Intensity of suspending SI of u (CPU cycles/bit)
ρ_u^r	Intensity of resuming SI of u (CPU cycles/bit)
ξ	Seamless service migration constraint
$V_u^t(t)$	Task volume of user u at time t
$V_u^r(t)$	Result volume of user u at time t
$d_u^c(t)$	Communication latency of u at time t
$d_u^p(t)$	Computing latency of u at time t
$\kappa_u(t)$	Task computing intensity of u at time t
τ_u	Edge computing service latency constraint of u
P_u^c	Computing capacity allocated to u at time t
$P_u^{rc}(t)$	Computing capacity required by u at time t
$\mathcal{P}^h$	Maximum computing capacity of ES h

Besides, the link from edge to cloud always receives enormous amounts of data generated by ubiquitous IoT devices and vertical systems. Thus, forwarding too much load to the cloud server may lead to congestion on the cloud and the links to the cloud, decreasing system performance. In addition, the fundamental objective of the edge service controller is to fully utilize the deployed EC system to provide guaranteed QoS for IoT users and release the load of the cloud.

Therefore, in this article, we set the primary objective of the DTASM problem in a dynamic edge-cloud system as minimizing the long-term expected load forwarded to the cloud server by optimizing task allocation decisions, i.e.,

$$\begin{aligned} \mathbf{P1}: \min_{\mathbf{x}} \lim_{T\to\infty} \frac{1}{T}\sum_{t=0}^{T}\sum_{u\in\mathcal{U}} x_u^{|H'|}(t)\big(V_u^t(t)+V_u^r(t)\big) \\ \text{subject to. } \mathbf{C1}, \mathbf{C2}, \mathbf{C3}, \mathbf{C4}. \end{aligned} \tag{14}$$

The notations defined till now in this article are listed in Table I.

IV. DRL-BASED DTASM APPROACH

This section illustrates our proposed DRL-based DTASM approach, including the design of the training of DTASM model and the user selection on each ES when implementing DTASM.

A. User Selection on Edge Server

It can be seen from problem **P1** that the final task allocation result determines the load forwarded to the cloud server. However, considering the objective of our work and the system settings, each user's task is set not to be actively allocated to the cloud server. In other words, the DTASM algorithm performed by the edge service controller always selects a target ES rather than the cloud server for each IoT user to process their computing tasks. Moreover, users' tasks are reallocated to the cloud passively only when breaking any constraint. Once the task allocation policy is determined, the minimum load to the cloud server can be achieved by minimizing the load from every ES to the cloud server. Thus, in addition to users whose service migration breaks constraints or whose required computing capacity exceeds the maximum computing capacity of the ES, there are still multiple users allocated to one ES. Moreover, the computing capacity required by each of these users is less than the capacity of the ES. Nevertheless, the sum of their capacity requirement exceeds the maximum capacity of the ES. Then, the ES has to decide which users are reallocated to the cloud server.

Based on the above analysis and the objective of minimizing the load forwarded to the cloud server, we can formulate the following subproblem on any ES h when implementing a DTASM operation at time t:

$$\begin{aligned} \mathbf{P2}: \min \sum_{u\in\mathcal{U}_h}\Big(1-\alpha_u^h(t)\Big)\big(V_u^t(t)+V_u^r(t)\big) \\ \text{subject to. } \sum_{u\in\mathcal{U}_h}\alpha_u^h P_u^{rc}(t) \le \mathcal{P}^h \end{aligned} \tag{15}$$

where $\mathcal{U}_h$ represents the set of IoT users whose tasks are allocated to ES h according to the task allocation policy being implemented and the required computing capacity less than the available capacity of h. We use the binary indicator $\alpha_u^h(t)$ to represent the user selection decision that $\alpha_u^h(t)=1$ represents the situation when u is selected by the ES to be served and $\alpha_u^h(t)=0$ means u is rejected by h and its tasks are reallocated to the cloud server for processing. The constraint indicates that the sum of computing capacity required by those selected users must not exceed the maximum computing capacity of the ES. We can find that **P2** can be reformulated as a knapsack problem that maximizes the sum load of selected users, i.e.,

$$\begin{aligned} \mathbf{P3}: \max \sum_{u\in\mathcal{U}_h}\alpha_u^h(t)\big(V_u^t(t)+V_u^r(t)\big) \\ \text{subject to. } \sum_{u\in\mathcal{U}_h}\alpha_u^h P_u^{rc}(t) \le \mathcal{P}^h. \end{aligned} \tag{16}$$

Then, the optimal user selection result based on problem **P3** can be obtained by methods such as dynamic programming [39].

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

B. DRL-Based Framework

Although the optimal user selection policy can be obtained above, the difficulty of problem **P1** is still dramatically increased when considering the system dynamics and heterogeneity. Therefore, we leverage DRL in our approach as it has the advantage of adapting to an unknown system without requiring prior global knowledge that may be inaccessible in some actual systems [40], such as the attributes of underlay communication networks, the heterogeneous resource capacity constraints, and independent user selection methods of ESs introduced above. Besides, DRL agent can update its policy from real-time interactions with the environment, enabling it to adapt to the ever-changing services in IoT systems [3].

For an edge-cloud IoT system with dynamic system states, the objective of the edge service controller is to maintain a long-term QoS guarantee and performance optimization by providing dynamic management, i.e., task allocation and service migration in this article. Meanwhile, the system state transition at each time t depends only on the real-time state and the task allocation policy employed in the current state. Thus, the DTASM process is an MDP. Then, DRL can be employed to train an intelligent DTASM agent, which can be installed on the edge service controller to generate task allocation action for any observed system state. The IoT system executes the task allocation action generated by the DRL agent following the DTASM process, and a reward is fed back to the agent after implementation. Meanwhile, the system moves on to the next state. The agent updates its policy by utilizing these experienced states, actions, and rewards. The state, action, and reward used in this article are defined as follows.

State: The system state at any time t is represented by an array constructed by the real-time state of IoT users, including the volume of computing task and result, the size of state context, computing intensity, real-time location, the application kind, and current SI location of each user, i.e.,

$$s_t = \left[V_u^t(t), V_u^r(t), V_u^m(t), \kappa_u(t), y_u(t), F_u, \mathcal{X}_u(t)\right]_{|\mathcal{U}|\times 7}. \tag{17}$$

Action: An array is used to indicate the generated task allocation action for a given state s_t, i.e.,

$$a_t = \left[a_{i,j}\right]_{|\mathcal{U}|\times|\mathcal{H}|}. \tag{18}$$

Each row of a_t represents the probability distribution of allocating the task processing of the corresponding user to ESs in the system, and the sum of elements in each row is 1. The system deterministically selects the ES with the highest probability in each row to allocate the user's task processing. Then, the system implements DTASM according to a_t based on the current real-time system state, and a final task allocation policy $\mathbf{x}_t = [x_u^h(t)]_{|\mathcal{U}|\times|\mathcal{H}'|}$ is obtained after the implementation of DTASM operations.

Reward: We design the total load forwarded to the cloud under final policy $\mathbf{x}_t$ as the reward r_t of implementing the action a_t. Since we aimed to minimize the load forwarded to the cloud server, a negative reward value is employed, i.e.,

$$r_t = -\sum_{u\in\mathcal{U}} x_u^{|H'|}(t)\left(V_u^t(t) + V_u^r(t)\right). \tag{19}$$

The candidate task allocation action of an IoT user includes all ESs in the system. Thus, the action space size of a task allocation policy is immense (e.g., 8^{200} candidate actions for an IoT system with 8 ESs and 200 users). Thus, value-based reinforcement learning (RL) algorithms (e.g., Q-learning) are not usable in such a problem with a vast discrete action space. The reason is that value-based algorithms generally select the optimal action by comparing state–action values of all candidate actions, which is challenging, even impossible. Policy-based RL algorithms are more efficient in problems with continuous or large state and action space.

C. DTASM Algorithm Based on Twin-Delayed DDPG

The policy-based DRL methods update the agent based on the policy gradient method, which mainly includes stochastic policy gradient and deterministic policy gradient [41]. The stochastic policy gradient methods select actions based on probability distributions of actions and update policy parameters based on the probability of selected actions [42]. However, the probability of selecting any action is likely to be extremely small in problems with vast discrete action space unless the policy tends to be deterministic since the action probability is usually estimated by chain rule, which may result in an unstable training process. Deterministic policy gradient algorithms can directly generate actions and update parameters based on estimated state-action values [43]. Moreover, a deterministic policy generates determined action for a given state, making it more reliable and more suitable for systems with strict requirements on reliability and QoS. Thus, this article employs the DDPG method.

In DRL, a policy π can generate an action for any given state, i.e., $\pi : \mathcal{S} \to \mathcal{A}$. Given any random initial state s_t, the expected return of taking an action a_t following the policy π is defined as the state–action value (Q-value), which is usually estimated by deep neural networks in complex problems [40]:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\{\mathcal{S},\pi\}}[R_t|s_t, a_t] \tag{20}$$

where R_t is the expected sum of rewards and usually employs a discounted format with a discount factor $\gamma \in [0, 1)$, i.e.,

$$R_t = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i). \tag{21}$$

Then, according to the Bellman equation, (20) can be rewritten as

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s\sim\mathcal{S}}\left[V(s_{t+1}, \pi)\right] \tag{22}$$

where $V(s_{t+1}, \pi)$, known as the state value function, is the expectation of state–action values following policy π, i.e.,

$$V(s_{t+1}, \pi) = \mathbb{E}_{a_{t+1}\sim\pi}\left[Q^\pi(s_{t+1}, a_{t+1})\right]. \tag{23}$$

Thus, the objective of DRL is to learn an optimal policy π^* that can maximize the expected sum of long-term rewards from any random initial system state, i.e.,

$$\pi^* = \arg\max_\pi \sum_t \mathbb{E}_{(s_t,a_t)}[r(s_t, a_t)]. \tag{24}$$

Under a deterministic policy, the state–action value only depends on the state transition feature of the environment. For

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

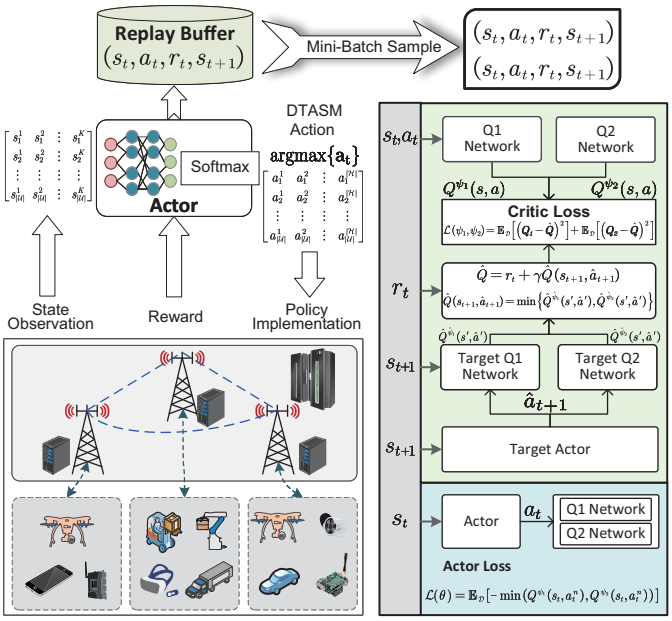


Fig. 3. Training of DRL-based DTASM agent based on twin-delayed DDPG ($s' \equiv s_{t+1}, a' \equiv a_{t+1}, \hat{a}' \equiv \hat{a}_{t+1}$).

a considered edge-cloud system that maintains relative stability during the implementation of DTASM, the state transition after executing an action is also determined. Thus, we can use Q-value to represent the state value of a given state under a deterministic policy. Assuming we use a policy network π parameterized by θ, a Q-function with parameters ψ to estimate the Q-value of any given state–action pair (s_t, a_t), the parameters of the Q-network during training can be then updated by minimizing the Bellman residual loss [42], i.e.,

$$\mathcal{L}_Q(\psi) = E_{(s_t,a_t)\sim\mathcal{D}}\Big[\big(Q^{\psi}(s_t,a_t) - (r(s_t,a_t) + \gamma Q^{\psi}(s_{t+1},\pi(s_{t+1})))\big)^2\Big]. \quad (25)$$

For any observed system state, the task allocation action is generated by an actor policy network, whose parameters can be updated by minimizing the negative expected state–action value using the gradient descent method [41]

$$\mathcal{L}(\theta) = E_{(s_t,a_t)\sim\mathcal{D}}[Q(s_t,\pi_\theta(s_t))]. \quad (26)$$

To improve the training stability, we employ the twin-delayed DDPG architecture [44], i.e., twin Q-networks, target networks, and delayed update of the actor policy network and target networks. Fig. 3 presents the architecture of training DTASM policy based on twin-delayed DDPG. To update parameters of actor policy network using Q-values of sampled state–action pairs, the actions inputted to the Q-networks should maintain the gradient information of the actor policy network so that the backward optimizer can update the parameters θ. Therefore, we employ a softmax activation function at the output layer of the actor policy network to get the probabilities of selecting every ES for each user. Then, the output of the softmax function is recorded as the differentiable action (a_t) for training. After that, we can obtain an executable action $(\widetilde{a}_t)$ by implementing an argmax function on a_t to select the server with maximum probability as the task allocation result (i.e., $\widetilde{a}_t = \arg\max(a_t)$). $\widetilde{a}_t$ is executed following the DTASM work procedures described in Section III. The final task allocation policy $\mathbf{x}_t$ and a reward r_t are obtained when completing the execution of $\widetilde{a}_t$. Meanwhile, the system evolves into a new state s_{t+1}. Finally, the experience, including the state, the differentiable action, the reward, and the state transition (i.e., (s_t, a_t, r_t, s_{t+1})), is recorded to a replay buffer and sampled later to update parameters. The symbols of s_t, a_t, r_t, and s_{t+1} shown on the right-hand side of Fig. 3 represent the experiences sampled from the replay buffer, and those in the gray box indicate the input of each training procedure.

Algorithm 1: Training of DRL-Based DTASM Agent

Input: Actor network (θ), twin Q-networks (ψ_1, ψ_2), target actor network ($\hat{\theta}$), target twin Q-networks ($\hat{\psi}_1, \hat{\psi}_2$), IoT system
Output: Trained DTASM policy with parameter θ

Initialize the IoT system
Initialize a replay buffer
for $t = 1, 2, 3, \cdots,$ **do**
  if $t \le T_c$ **then** // Experience collection
    Randomly update the system state of IoT users
    $s_t \leftarrow$ Observed system state
    /* Generate and map action based on system state */
    $a_t = \pi_\theta(s_t)$, $\widetilde{a}_t = \arg\max(a_t)$
    $(r_t, s_{t+1}) = \text{DTASM}(\widetilde{a}_t)$
    Add (s_t, a_t, r_t, s_{t+1}) to the replay buffer
  else
    Randomly update the system state of IoT users
    $s_t \leftarrow$ Observed system state
    $a_t = \pi_\theta(s_t)$, $\widetilde{a}_t = \arg\max(a_t)$
    $(r_t, s_{t+1}) = \text{DTASM}(\widetilde{a}_t)$
    Add (s_t, a_t, r_t, s_{t+1}) to the replay buffer
    update Q-networks:
    $\mathcal{D} = [(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s})]$ // The sampled experience
    /* Estimate the Q and target Q value for all samples: */
    $\hat{a}_{i+1} = \hat{\pi}_{\hat{\theta}}(s_{i+1})$
    $\hat{Q}_{(s_{i+1},\hat{a}_{i+1})} = \min\{\hat{Q}^{\hat{\psi}_1}(s_{i+1},\hat{a}_{i+1}), \hat{Q}^{\hat{\psi}_2}(s_{i+1},\hat{a}_{i+1})\}$
    $\hat{Q} = r(s_i, a_i) + \gamma \hat{Q}_{(s_{i+1},\hat{a}_{i+1})}$
    $Q_1 = Q^{\psi_1}(s_i, a_i)$, $Q_2 = Q^{\psi_2}(s_i, a_i)$
    $\mathcal{L}_Q(\psi_1, \psi_2) = \mathbb{E}_{\mathcal{D}}[(\boldsymbol{Q_1} - \hat{\boldsymbol{Q}})^2] + \mathbb{E}_{\mathcal{D}}[(\boldsymbol{Q_2} - \hat{\boldsymbol{Q}})^2]$
    $(\psi_1, \psi_2) = \text{Adam}(\nabla_{\{\psi_1,\psi_2\}}(\mathcal{L}_Q(\psi_1, \psi_2)))$
    if $t\%\lambda = 0$ **then** // Delayed update
      update actor:
      $\mathcal{D} = [(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s})]$
      $a_i^n = \pi_\theta(s_i)$ // Generate new actions
      $\mathcal{L}(\theta) = \mathbb{E}_{\mathcal{D}}[\min(Q^{\psi_1}(s_i, a_i^n), Q^{\psi_2}(s_i, a_i^n))]$
      $\theta = \text{Adam}(\nabla_\theta(-\mathcal{L}(\theta)))$
      update target networks:
      $\hat{\theta} = \mu\hat{\theta} + (1-\mu)\theta$
      $\hat{\psi}_1 = \mu\hat{\psi}_1 + (1-\mu)\psi_1$
      $\hat{\psi}_2 = \mu\hat{\psi}_2 + (1-\mu)\psi_2$
    end
  end
end

Algorithm 1 details the training of DRL-based DTASM policy. Before the training starts, in addition to the randomly initialized actor policy network parameterized by θ, twin Q-networks (ψ_1, ψ_2), a target actor policy network ($\hat{\theta}$), and target twin Q-networks ($\hat{\psi}_1, \hat{\psi}_2$) are also created to improve

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

the training efficiency and stability. The parameters of target networks are duplicated from the corresponding policy network, i.e., $\hat{\theta} \leftarrow \theta$, $\hat{\psi}_1 \leftarrow \psi_1$, and $\hat{\psi}_2 \leftarrow \psi_2$. Then, the IoT system is initialized, and a replay buffer is initialized to record experiences (i.e., state, action, reward, and state transition) during the training process. Besides, the system will run T_c episodes for experience collection before updating parameters. An untrained actor policy network generates task allocation policies for observed system states in these episodes. These collected experiences will be utilized to train the agent later. After launching the policy training, a batch of experience tuples is sampled from the replay buffer. For distinguishing purposes, the sample experiences are indicated by symbol *i*, which equals to the index symbol *t* shown on the right of Fig. 3. Then, the target actor generates a new action for the next state in each experience tuple (line 18). The target *Q*-value of each next state and new generated target action pair is estimated by the target twin *Q*-networks. The minimum value between them is selected as the target *Q*-value of the next state–action value to tackle the overestimation of the *Q*-value (lines 19 and 20). Then, the target *Q*-value of each sampled state and action pair is calculated in line 21. Similarly, the current updated twin *Q*-networks estimate *Q*-values of sampled state and action pairs (line 22). Finally, the loss values of the twin *Q*-networks are calculated in line 23, and the corresponding parameters (i.e., ψ_1 and ψ_2) are updated by the Adam optimizer to minimize the loss values. The delayed updates of the actor network and target networks are employed to improve the training stability, which means that the actor and all target networks are updated one time after every λ episodes. λ is called the delayed update frequency. The actor parameters are updated by minimizing the expected negative *Q*-values of the sampled states and newly generated actions (lines 27–30), in which the new action of each sampled experience is generated by the real-time updated actor policy network (line 28). After updating the actor network, the target networks are updated (lines 32–34), where μ is the soft update coefficient that makes the training process more stable. Finally, the performance of actions generated by the agent will converge to an optimal value, which indicates the successful training of the DRL-based DTASM agent. The edge service controller can perform the trained agent policy for DTASM management in the edge-cloud system. The replay buffer can be updated with the latest experiences during the runtime of the system. If the system state changes dramatically or a significant performance degradation is observed, the controller can update its policy by recalling the training process with stored experiences.

V. Performance Evaluation

A. Simulation Setup

To evaluate our proposed DRL-based DTASM algorithm, we conduct extensive simulations in the Pytorch environment. We randomly generate an IoT system, where the computing capacity of each ES and links' bandwidth are randomly generated from the corresponding parameter value interval. Meanwhile, we set that there are ten kinds of applications, and the parameters related to SFs are also randomly generated. Each IoT user is randomly installed with one kind of these applications. At every training episode, the user states are randomly updated with randomly selected values. Besides, to simulate the unbalance load among RAN nodes, we use a randomly generated gamma distribution to imitate the number of IoT users connecting to each RAN node. During training, we employ neural networks with four hidden layers, and the hidden size of every hidden layer is 256. The maximum capacity of the replay buffer is 2048, and the minibatch size is 128. The delayed update frequency (λ) is 20, and $(1-\mu)=0.005$. The learning rate is 1×10^{-4}. More simulation parameters are detailed in Table II. We compare the proposed DRL-based approach with several benchmarks.

TABLE II
Simulation Parameters (Param)

Param	Value	Param	Value
$\mathcal{P}^h$	[60, 100] GHz	β	[20, 30] %
$B_{i,j}$	[20, 80] Mbps	$\delta_{i,j}$	[1, 3] ms
$B_{i,\lvert\mathcal{H}'\rvert}$	[50, 120] Mbps	$\delta_{i,\lvert\mathcal{H}'\rvert}$	[10, 20] ms
V_u^t	[512, 1024] KByte	V_u^r	[10, 20] KByte
V_u^m	[1, 8] MByte	d_u^{sf}	[100, 200]ms
τ_u	[800, 1000] ms	κ_u	[228, 538] cycles / bit
P_u^{SF}	$[200, 300]\times10^6$Hz	ρ^s, ρ^r	[350, 450] cycles / kbits

WoMig-Stable: The SIs of IoT users are continuously providing service without active migration (WoMig). Users' tasks are always allocated to the ESs on which their SIs are currently instanced and are only reforwarded to the cloud when they cannot be satisfied due to ESs' capacity constraints.

WoMig-Dynamic: To test the performance of WoMig under different conditions, we set the system updating the task allocation policy randomly after every 500 episodes and subsequently runs without service migration for 500 episodes.

Random: The system randomly selects an ES for each user to process their computing tasks. Meanwhile, the SIs are migrated by comparing the new policy and the current policy.

Nearest: Allocating tasks of a user to its nearest ES can minimize the computing capacity requirement due to minimized communication delay. Thus, the Nearest algorithm generates new task allocation policies that select the nearest ES for users to process their tasks.

Service Downtime-Aware Nearest (DANearest): Users whose service migration does not break the seamless migration constraint is reassigned to their nearest ES to reduce the load forwarded to the cloud caused by breaking the service migration constraint.

B. Performance Metrics

We evaluate the proposed DRL-based DTASM approach and compare it with benchmarks under the following metrics.

1) *Load Forwarded to Cloud:* This performance is calculated based on (14), which can reflect the accuracy and effectiveness of task allocation policies. Because the load offloaded to the cloud is caused by breaking constraints of capacity, service latency, or seamless

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

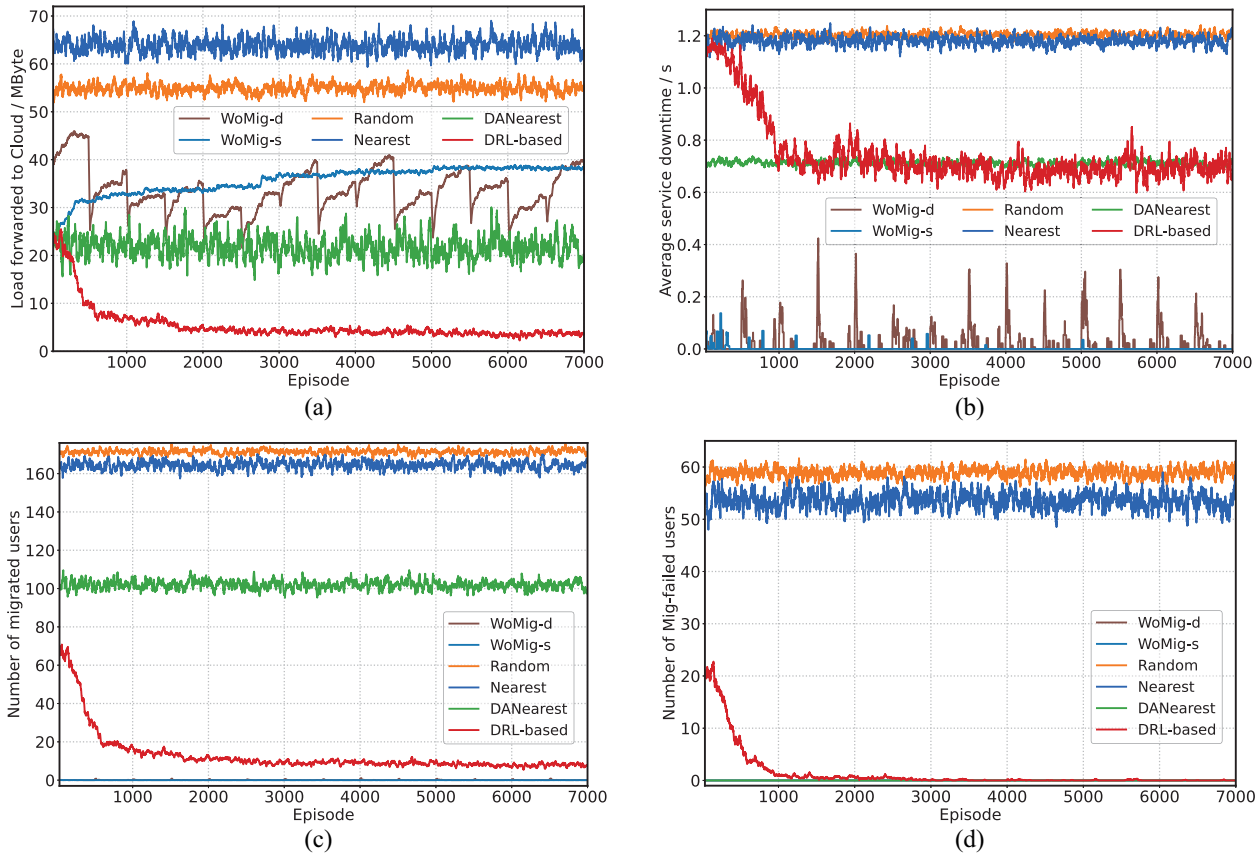


Fig. 4. Performances over training episodes. $|\mathcal{U}| = 200$, $|\mathcal{H}| = 8$, $\xi = 1200$ ms, and $\gamma = 0.1$. (a) Load forwarded to the cloud server. (b) Average service downtime. (c) Number of migrated users. (d) Number of migration-failed users.

migration. These conditions can all be attributed to unreliable task allocation policies.

2) *Average Service Downtime:* The service downtime is calculated by (3) and (8), which can reflect the cost introduced by implementing DTASM. As the service downtime of every user is hard to visualize and with high variability, we employ the average value of the users whose SI are migrated to another server after implementing a DTASM action.
3) *Number of Migrated Users:* The total number of users whose SI is migrated to another server after implementing the new task allocation policy. This performance can also reflect the cost of implementing a DTASM operation as migrating more service generally introduce a higher cost in real IoT systems.
4) *Number of Migration-Failed (Mig-Failed) Users:* The number of users whose service migration breaks the seamless migration constraint during the implementation of a DTASM action. This performance can reflect the reliability of the new task allocation policy generated by the EC service controller and the effectiveness of the performed algorithm.

C. Simulation Results

Fig. 4 shows the performances during the training process in a considered IoT system, where 8 ESs and 200 IoT users are randomly generated. The parameters of ESs, SFs, and bandwidths are randomly initialized based on parameters in Table II. Then, we fix these settings for the simulations of all evaluated algorithms. Then, at each time, we update the system state of users (i.e., connected RAN, task volume, result volume, computing intensity, and volume of state context) with values randomly selected from configured parameters. We employ the moving average method to process the results of every 20 episodes.

Fig. 4(a) shows that the load forwarded to the cloud server changes over the training phase. We can observe that the Random and the Nearest task allocation policies result in much more load being forwarded to the cloud server compared with other algorithms. The reason is that Nearest and Random methods can result in a high probability of breaking the seamless service migration constraint. Meanwhile, compared with the Random algorithm, the Nearest method may allocate too many users to one ES, resulting in more users being reassigned to the cloud server due to insufficient computing capacity. Without service migration, the load forwarded to the cloud server gradually increases, and the value maintains stability at the later stage. The reason is that the number of users reallocated to the cloud server because of unsatisfied computing capacity requirements gradually increases as the system runs dynamically. Eventually, the system enters a steady state where users remaining in the edge system can always be satisfied. Furthermore, this is also apparent from the subsequent performance of the WoMig-dynamic approach after each random update. Moreover, the seamless migration

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

constraint is not violated in the system without migration. Thus, loads reforwarded to the cloud server caused by WoMig-d and WoMig-s approaches are much less than that introduced by Random and Nearest algorithms. Similarly, compared with the Nearest algorithm, the DANearest algorithm can significantly reduce the load forwarded to the cloud server because the migration constraint can always be satisfied. The proposed DRL-based approach can minimize the load forwarded to the cloud server compared with other algorithms. After 2000 episodes of training, the performance converged to the minimum value of about 4 Mbyte. Fig. 4(b) shows the comparison of average service downtime. We can find that the average service downtime caused by the Random approach and the Nearest approach is about 1200 ms, which is much higher than that introduced by other algorithms because much more users break the seamless migration constraint [Fig. 4(d)]. Service downtime is also observed in the system without service migration (i.e., WoMig-s and WoMig-d), especially in the episodes immediately following a random task allocation (e.g., WoMig-d). The service downtime is introduced by breaking the capacity constraints of ESs and migrating SIs from ESs to the cloud. Due to dynamic changes in computing requirements, in some episodes, the total capacity required by users allocated to one ES may exceed the maximum capacity of the server. Thus, SI of some users need to be migrated to the cloud server, and service downtime is introduced. The proposed DRL-based DTASM algorithm results in a similar service downtime performance as the DANearest algorithm, and the value is the lowest in algorithms with active service migration. Fig. 4(c) shows the number of migrated users after the implementation of DTASM procedures. Random and the Nearest approaches result in much higher numbers of migrated users than the other algorithms. They need to migrate about 170 and 165 users, which are close to the 200 IoT users. Since the DANearest algorithm considers the seamless migration constraint, it migrates far fewer users than the Nearest algorithm. In most cases, after each random task allocation in systems without active service migration, there are few users are migrated from the ES to the cloud due to breaking capacity constraints of ESs. Our DRL-based approach performs significantly superior to other algorithms that migrate users actively. The average number of migrated users caused by the DRL-based algorithm is about 8, which indicates that it can provide optimal performance with minimum migration cost since higher costs are introduced when migrating more services in real systems. Fig. 4(d) shows the number of migration-failed users when implementing DTASM operations. We can observe that almost no user breaks the migration constraint when employing the DRL-based algorithm.

Fig. 5 presents the convergence performance of our DRL-based algorithm under different discount factors [45]. The first 50 episodes are the data collection episodes when the agent has not started the training process. We can find that the proposed DRL agent performs with slight performance variation over different discount factors, but all converge to an optimal value. Besides, we can observe that the number of migration-failed users converges to 0 even with different discount factors. These results prove the effectiveness of the proposed DRL-based algorithm.

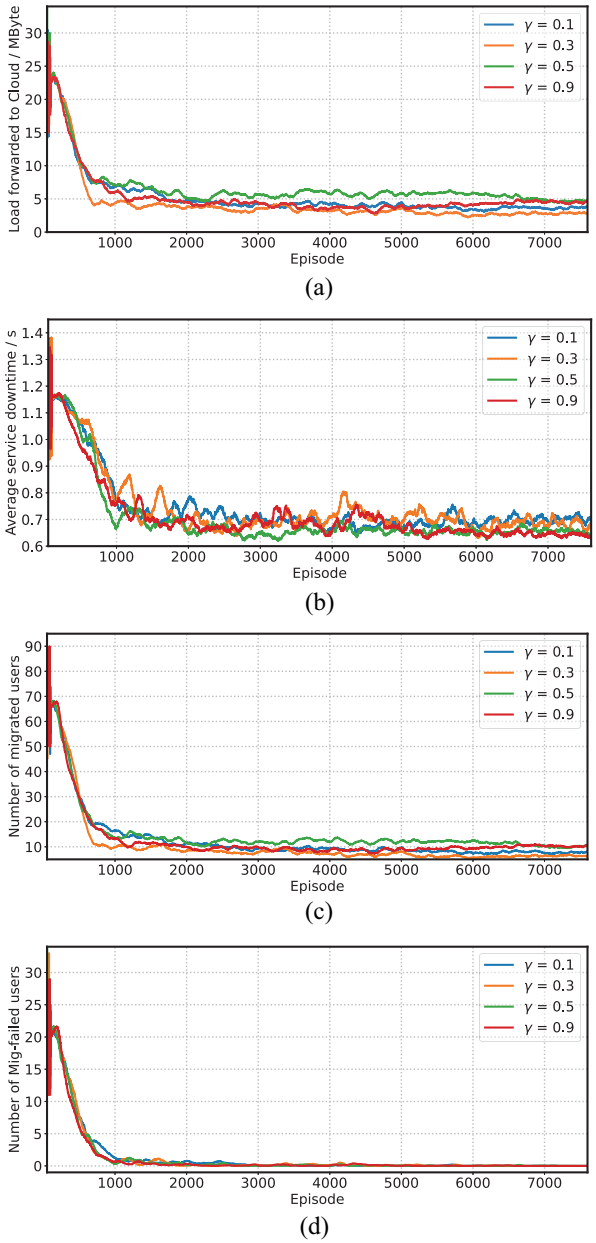


Fig. 5. Impact of discount factors on convergence. $|\mathcal{U}| = 200$, $|\mathcal{H}| = 8$, and $\xi = 1200$ ms. (a) Load forwarded to the cloud server. (b) Average service downtime. (c) Number of migrated users. (d) Number of migration-failed users.

In addition, we investigated the impact of the number of IoT users. As shown in Fig. 6, we change the number of IoT users and then average the results of the last 500 episodes of each simulation. We can find from Fig. 6(a) that all algorithms increase the load forwarded to the cloud server as the number of users increases. Nevertheless, the proposed DRL-based algorithm can always obtain optimal performance compared with others. Besides, the performance obtained in the system without service migration is even inferior to that obtained by the Random algorithm when there are more than 240 IoT users, which reveals the necessity of DTASM. Fig. 6(b) shows

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

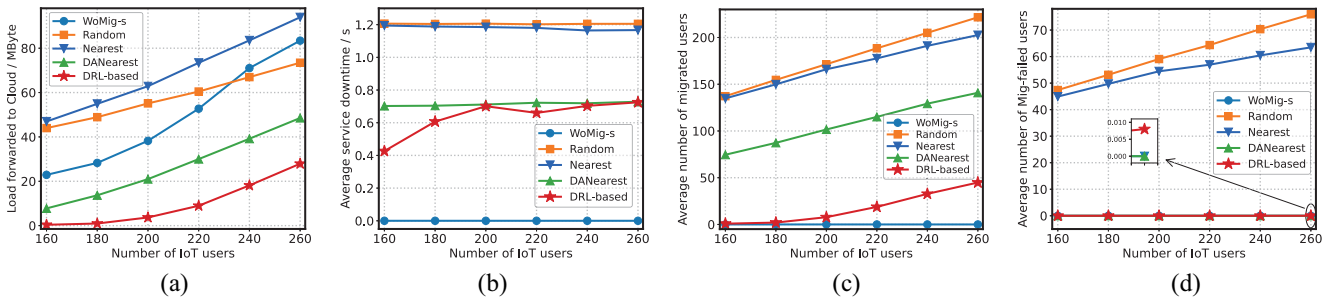


Fig. 6. Impact of the number of IoT users. $|\mathcal{H}| = 8$ and $\xi = 1200$ ms. (a) Average load forwarded to Cloud. (b) Average service downtime. (c) Average number of migrated users. (d) Average number of migration-failed users.

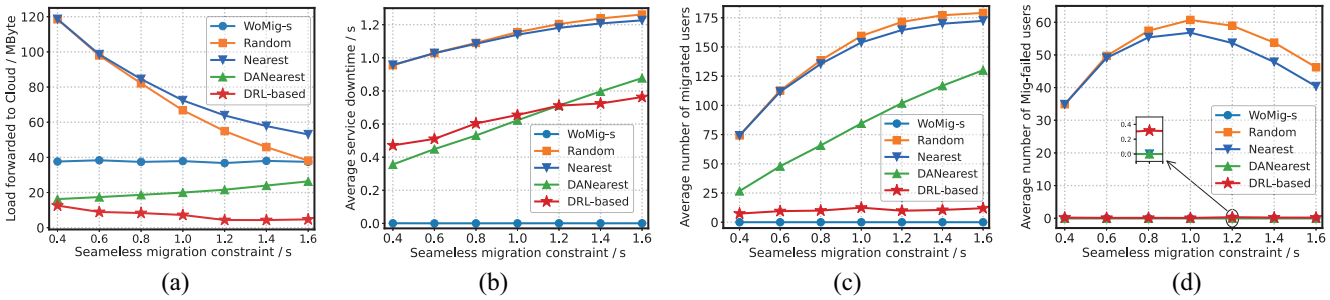


Fig. 7. Impact of the seamless migration constraint (ξ). $|\mathcal{H}| = 8$, $|\mathcal{U}| = 200$, and $\gamma = 0.1$. (a) Average load forwarded to Cloud. (b) Average service downtime. (c) Average number of migrated users. (d) Average number of migration-failed users.

that the DRL-based DTASM algorithm can minimize the average service downtime as compared to other algorithms with active service migration. Besides, the DRL-based algorithm introduces higher average service downtime as the number of users increases and reaches a stable value that almost equals that caused by the DANearest algorithm. Fig. 6(c) reveals the average number of migrated users. We can find that all algorithms with active service migration migrate more users as the number of IoT users increases, but the proposed DRL-based algorithm can obtain optimal performance with minimal user migration. Fig. 6(d) shows the number of migration-failed users. We can find that our DRL-based approach almost always enables all migrated users to satisfy the seamless migration constraint. However, as the number of users increases, both Random and Nearest methods will cause more users to break the constraint. We can conclude from Fig. 6 that our DRL-based algorithm can achieve optimum performance with minimal cost and almost always be able to satisfy the seamless migration constraint.

Fig. 7 exhibits the impact of the seamless migration constraint. We can find that the seamless migration constraint does not influence the performances in systems without service migration. Therefore, the following part will not discuss this approach. From Fig. 7(a), we find that all algorithms, except for the DANearest algorithm, decrease the load offloaded to the cloud as the seamless migration constraint reduces. Random and Nearest algorithms result in the most significant decrease in load offloaded to the cloud. The reason is that these two algorithms do not consider migration constraints, which makes their performance more sensitive to the constraint. Moreover, a higher migration constraint enables more service migrations to satisfy the constraint, significantly reducing the load reforwarded to the cloud server. Meanwhile, when employing our DRL-based algorithm, the load offloaded to the cloud decreases slightly because the controller can migrate more users to appropriate ESs in cases with higher migration constraints. Then, the probability that users' computing capacity requirements exceed the maximum capacity of ESs after implementing a DTASM action is reduced. The load forwarded to the cloud caused by the DANearest algorithm slightly increased, mainly caused by migrating more users to their nearest ES. The probability that the capacity of one ES cannot satisfy users' requirements increases when too many users are allocated to it. In such conditions, limiting the number of migrated users can utilize other servers to release the load from overloaded ESs and reduce the load reforwarded to the cloud due to breaking the computing capacity constraints. Fig. 7(b) shows the comparison of average service downtime, and we can see that the average service downtime increases as the constraint increases. The DANearest and the DRL-based algorithms consider the migration constraint when designing task allocation policies. Thus, more SIs that may cause higher downtime are migrated under higher allowed constraints, increasing the average service downtime. Meanwhile, the DRL-based algorithm only migrates necessary users rather than all users that satisfy the constraint because the agent is trained to minimize the load forwarded to the cloud. Thus, the DRL-based algorithm introduces lower service downtime than the DANearest algorithm when the migration constraint is higher than 1.2 s. The average service downtime decreases when employing algorithms without considering the migration constraint (i.e., the Random and Nearest methods) as the

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

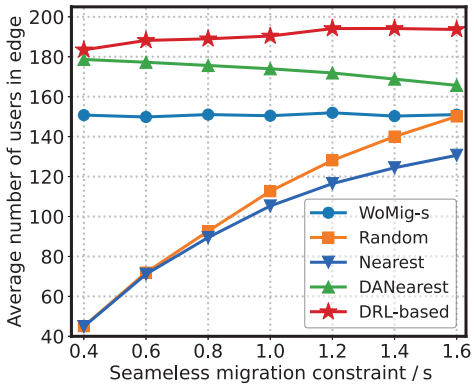


Fig. 8. Average number of users allocated to ESs under different migration constraints. $|\mathcal{H}| = 8$, $|\mathcal{U}| = 200$, and $\gamma = 0.1$.

migration constraint decreases. The main reason is that fewer users are allocated to ESs due to breaking the migration constraint under lower migration constraint conditions. To prove this view, we plot the average number of users allocated to ESs of these simulations in Fig. 8, from which we can see that most users are reallocated to the cloud server when employing the Random and Nearest approaches when setting lower seamless migration constraints. Compared with ESs, the cloud server can dramatically reduce the SI suspending time and SI resuming time, resulting in low service downtime of users allocated to or migrated from the cloud server. So is the state context synchronization latency. However, the controller can migrate more users with higher service downtime without breaking the constraint when setting higher migration constraints, which increases the average service downtime. Most users are allocated to ESs. Fig. 7(c) presents the average number of service-migrated users. We can observe that the DRL-based algorithm migrates much fewer users than others and maintains stability, which reveals that the DRL-based algorithm takes only meaningful service migration actions rather than voracious ones. The DANearest algorithm migrates more users when the migration constraint increases since it always migrates every user satisfying the constraints. When adopting Random and Nearest algorithms, fewer users are migrated in scenarios with lower migration constraints. The reason is that most users are allocated to the cloud in these conditions, and most of them will still be reallocated to the cloud server due to breaking constraints when implementing DTASM, making them not be classified as migrated users. Fig. 7(d) shows the average number of migration-failed users. The DRL-based algorithm can ensure almost all service migrations satisfy the constraint. The Random algorithm and the Nearest approach both present an upward and then a downward trend. Few users are allocated to ESs when setting lower migration constraints, and only some of these users are counted into migration-failed users when their migration failed. The users on the cloud are not counted as migration-failed users, although their service migration breaks the constraint since they are still reallocated to the cloud. More users are allocated to ESs when setting higher migration constraints because service migration related to the cloud server result in low service downtime, but most of these users still cannot satisfy the migration constraint when being migrated among ESs. Thus, the number of migrated users and migration-failed users both increased at first. When the migration constraint is high enough, the controller can migrate more users without breaking the constraint, and more users can be allocated to ESs. Thus, the average service downtime still increases [i.e., Fig. 7(b)], while the number of migration-failed users decreases.

VI. Conclusion

This article investigated the DTASM problem in edge-cloud IoT systems, in which the dynamic computing requirements, user mobility, and seamless service migration constraint are comprehensively considered. The investigated problem was decomposed into task allocation and user selection subproblems. The user selection problem is reformulated as a knapsack problem and addressed by dynamic programming. Then, to cope with the enormous action space challenge of task allocation in the system with a mass of IoT users, an approach based on twin-delayed DDPG was proposed. The objective is to generate a DTASM action for any observed system state to minimize the total computing load forwarded to the cloud server while satisfying the seamless service migration constraint. Simulation results demonstrated that the proposed DRL-based approach can minimize the computing load forwarded to the cloud server and can ensure that almost all service migrations satisfy the migration constraint compared with benchmarks. Besides, the DRL-based approach can minimize the number of migrated users, thereby reducing the overall service migration cost in the IoT system.

References

[1] D. Wu, R. Bao, Z. Li, H. Wang, H. Zhang, and R. Wang, "Edge-cloud collaboration enabled video service enhancement: A hybrid human-artificial intelligence scheme," *IEEE Trans. Multimedia*, vol. 23, pp. 2208–2221, 2021, doi: 10.1109/TMM.2021.3066050.

[2] D. Wu, Z. Yang, B. Yang, R. Wang, and P. Zhang, "From centralized management to edge collaboration: A privacy-preserving task assignment framework for mobile crowdsensing," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4579–4589, Mar. 2021.

[3] Y. Zhou *et al.*, "Service-aware 6G: An intelligent and open network based on the convergence of communication, computing and caching," *Digit. Commun. Netw.*, vol. 6, no. 3, pp. 253–260, 2020.

[4] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for Internet of Things realization," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2961–2991, 4th Quart., 2018.

[5] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 377–390, Feb. 2021.

[6] D. Wu, X. Han, Z. Yang, and R. Wang, "Exploiting transfer learning for emotion recognition under cloud-edge-client collaborations," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 2, pp. 479–490, Feb. 2021.

[7] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.

[8] G. Yang, L. Hou, X. He, D. He, S. Chan, and M. Guizani, "Offloading time optimization via Markov decision process in mobile-edge computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2483–2493, Feb. 2021.

[9] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, and Y. Yang, "Deep reinforcement learning based service migration strategy for edge computing," in *Proc. IEEE Int. Conf. Serv. Orient. Syst. Eng. (SOSE)*, 2019, pp. 116–1165.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.

[10] I. Farris, T. Taleb, H. Flinck, and A. Iera, "Providing ultra-short latency to user-centric 5G applications at the mobile network edge," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 4, p. e3169, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3169

[11] M. Horii, Y. Kojima, and K. Fukuda, "Stateful process migration for edge computing applications," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Barcelona, Spain, 2018, pp. 1–6.

[12] R. Bruschi, F. Davoli, P. Lago, C. Lombardo, and J. F. Pajo, "Personal services placement and low-latency migration in edge computing environments," in *Proc. IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*, 2018, pp. 1–6.

[13] "3rd Generation Partnership Project; technical specification group services and system aspects; study on application architecture for enabling edge applications; (release 17), version 17.0.0," 3rd Gener. Partnership Project (3GPP), Sophia Antipolis, France, 3GPP Rep. (TR) 23.758, Dec. 2019. [Online]. Available: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3614

[14] T. Taleb, A. Ksentini, and P. A. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 369–382, Apr.–Jun. 2019.

[15] H. Guo, J. Liu, and J. Zhang, "Computation offloading for multi-access mobile edge computing in ultra-dense networks," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 14–19, Aug. 2018.

[16] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[17] H. Li, H. Xu, C. Zhou, X. Lü, and Z. Han, "Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 10214–10226, Sep. 2020.

[18] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2651–2664, Dec. 2018.

[19] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018.

[20] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[21] Y. Hao, M. Chen, H. Gharavi, Y. Zhang, and K. Hwang, "Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5552–5561, Aug. 2021.

[22] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, early access, Nov. 10, 2020, doi: 10.1109/TMC.2020.3036871.

[23] W. Zhang, Y. Hu, Y. Zhang, and D. Raychaudhuri, "SEGUE: Quality of service aware edge cloud service migration," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, 2016, pp. 344–351.

[24] C. Zhang and Z. Zheng, "Task migration for mobile edge computing using deep reinforcement learning," *Future Gener. Comput. Syst.*, vol. 96, pp. 111–118, Jul. 2019.

[25] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[26] Z. Liang, Y. Liu, T.-M. Lok, and K. Huang, "Multi-cell mobile edge computing: Joint service migration and resource allocation," *IEEE Trans. Wireless Commun.*, vol. 20, no. 9, pp. 5898–5912, Sep. 2021.

[27] B. Gao, Z. Zhou, F. Liu, F. Xu, and B. Li, "An online framework for joint network selection and service placement in mobile edge computing," *IEEE Trans. Mobile Comput.*, early access, Mar. 9, 2021, doi: 10.1109/TMC.2021.3064847.

[28] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on Markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.

[29] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. A. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, 2015, pp. 1–9.

[30] C. Li, L. Zhu, W. Li, and Y. Luo, "Joint edge caching and dynamic service migration in SDN based mobile edge computing," *J. Netw. Comput. Appl.*, vol. 177, Mar. 2021, Art. no. 102966.

[31] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, and I. Humar, "A dynamic service migration mechanism in edge cognitive computing," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–15, May 2019. [Online]. Available: https://doi.org/10.1145/3239565

[32] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.

[33] "Network functions virtualisation (NFV); virtual network functions architecture," Eur. Telecommun. Stand. Inst (ETSI), Sophia Antipolis, France, ETSI document GS NFV-SWA 001, 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_nfv-swa001v010101p.pdf

[34] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 140–147, Feb. 2018.

[35] W. Bao *et al.*, "Follow me fog: Toward seamless handover timing schemes in a fog computing environment," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 72–78, Nov. 2017.

[36] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proc. Linux Symp.*, vol. 2, 2008, pp. 85–90.

[37] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Fast service migration in 5G trends and scenarios," *IEEE Netw.*, vol. 34, no. 2, pp. 92–98, Mar./Apr. 2020.

[38] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[39] M. Hristakeva and D. Shrestha, "Different approaches to solve the 0/1 knapsack problem," in *Proc. Midwest Instruction Comput. Symp.*, 2005, pp. 1–15.

[40] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

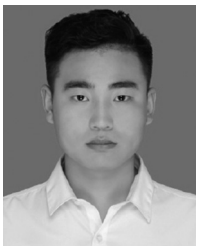
[41] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[42] T. Haarnoja *et al.*, "Soft actor-critic algorithms and applications," 2018, *arXiv:1812.05905*.

[43] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2019, *arXiv:1509.02971*.

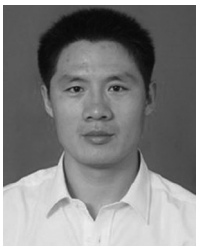
[44] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.

[45] T. M. Ho and K.-K. Nguyen, "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach," *IEEE Trans. Mobile Comput.*, early access, Dec. 10, 2020, doi: 10.1109/TMC.2020.3043736.



Yan Chen received the B.S. degree in information engineering from China University of Mining and Technology, Xuzhou, China, in 2016, where he is currently pursuing the Ph.D. degree in information and communication engineering with the School of Information and Control Engineering.

From December 2020 to December 2021, he was a visiting Ph.D. student with the School of Electrical Engineering, Aalto University, Espoo, Finland, under the support of the Chinese Government Scholarship awarded by the China Scholarship Council. His research interests include edge computing, Internet of Things, and wireless networks.

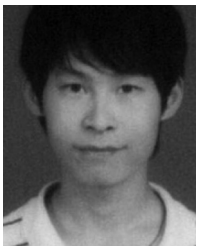


Yanjing Sun (Member, IEEE) received the Ph.D. degree in information and communication engineering from China University of Mining and Technology, Xuzhou, China, in 2008.

He has been a Professor with the School of Information and Control Engineering, China University of Mining and Technology since July 2012. His current research interests include wireless communication, embedded real-time system, wireless sensor networks, and cyber–physical system.

Prof. Sun is also a Council Member of the Jiangsu Institute of Electronics and a member of the Information Technology Working Committee of the China Safety Production Association.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.



Chenyang Wang (Member, IEEE) received the B.S. and M.S. degrees in computer science and technology from Henan Normal University, Xinxiang, China, in 2013 and 2017, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, College of Intelligence and Computing, Tianjin University, Tianjin, China.

He has been a visiting Ph.D. student under the support of China Scholarship Council with the School of Electrical Engineering, Aalto University, Espoo, Finland, since May 15, 2021. His current research interests include edge computing, big data analytic, reinforcement learning, and deep learning.

Mr. Wang received the Best Student Paper Award of the 24th International Conference on Parallel and Distributed Systems by IEEE Computer Society in 2018. He also received the Best Paper Award of IEEE International Conference on Communications in 2021.

Tarik Taleb (Senior Member, IEEE) received the B.E. degree (with Distinction) in information engineering and the M.Sc. and Ph.D. degrees in information sciences from Tohoku University, Sendai, Japan, in 2001, 2003, and 2005, respectively.

He is the Founder and the Director of the MOSA!C Lab, Espoo, Finland. Since October 2018, he has been a Full Professor with the Center of Wireless Communications, University of Oulu, Oulu, Finland. From October 2014 to December 2021, he was a Professor with the School of Electrical Engineering, Aalto University, Espoo. Prior to that, he was a Senior Researcher and a 3GPP Standards Expert with NEC Europe Ltd., Heidelberg, Germany. He was then leading NEC Europe Labs Team, involved with research and development projects on carrier cloud platforms, an important vision of 5G systems. From 2006 to 2009, he was an Assistant Professor with the Graduate School of Information Sciences, Tohoku University, in a laboratory fully funded by KDDI. From 2005 to 2006, he was a Research Fellow with the Intelligent Cosmos Research Institute, Sendai. He has also been directly engaged in the development and standardization of the Evolved Packet System as a member of the 3GPP System Architecture Working Group. His current research interests include architectural enhancements to mobile core networks (particularly 3GPP's), network softwarization and slicing, mobile cloud networking, network function virtualization, software-defined networking, mobile multimedia streaming, and unmanned vehicular communications.

Prof. Taleb was a recipient of the 2021 IEEE ComSoc Wireless Communications Technical Committee Recognition Award, the 2017 IEEE ComSoc Communications Software Technical Achievement Award, and several best paper awards at prestigious IEEE-flagged conferences for some of his research work. He was a co-recipient of the 2017 IEEE Communications Society Fred W. Ellersick Prize in 2017, the 2009 IEEE ComSoc Asia–Pacific Best Young Researcher Award in 2009, the 2008 TELECOM System Technology Award from the Telecommunications Advancement Foundation in 2008, the 2007 Funai Foundation Science Promotion Award in 2007, the 2006 IEEE Computer Society Japan Chapter Young Author Award in 2006, the Niwa Yasujirou Memorial Award in 2005, and the Young Researcher's Encouragement Award from the Japan Chapter of the IEEE Vehicular Technology Society in 2003.

Authorized licensed use limited to: Beijing Information Science & Tech Univ. Downloaded on March 01,2023 at 12:47:16 UTC from IEEE Xplore. Restrictions apply.