IEICE TRANSACTIONS

on Information and Systems

VOL. E105-D NO. 5
MAY 2022

A PUBLICATION OF THE INFORMATION AND SYSTEMS SOCIETY
The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

IEICE TRANS. INF. & SYST., VOL.E105–D, NO.5 MAY 2022

PAPER Special Section on Data Engineering and Information Management

Multi-Agent Reinforcement Learning for Cooperative Task Offloading in Distributed Edge Cloud Computing

Shiyao DING[†a)], ***Student Member and*** **Donghui LIN**[†b)], ***Member***

SUMMARY Distributed edge cloud computing is an important computation infrastructure for Internet of Things (IoT) and its task offloading problem has attracted much attention recently. Most existing work on task offloading in distributed edge cloud computing usually assumes that each self-interested user owns one edge server and chooses whether to execute its tasks locally or to offload the tasks to cloud servers. The goal of each edge server is to maximize its own interest like low delay cost, which corresponds to a non-cooperative setting. However, with the strong development of smart IoT communities such as smart hospital and smart factory, all edge and cloud servers can belong to one organization like a technology company. This corresponds to a cooperative setting where the goal of the organization is to maximize the team interest in the overall edge cloud computing system. In this paper, we consider a new problem called cooperative task offloading where all edge servers try to cooperate to make the entire edge cloud computing system achieve good performance such as low delay cost and low energy cost. However, this problem is hard to solve due to two issues: 1) each edge server status dynamically changes and task arrival is uncertain; 2) each edge server can observe only its own status, which makes it hard to optimize team interest as global information is unavailable. For solving these issues, we formulate the problem as a decentralized partially observable Markov decision process (Dec-POMDP) which can well handle the dynamic features under partial observations. Then, we apply a multi-agent reinforcement learning algorithm called value decomposition network (VDN) and propose a VDN-based task offloading algorithm (VDN-TO) to solve the problem. Specifically, the motivation is that we use a team value function to evaluate the team interest, which is then divided into individual value functions for each edge server. Then, each edge server updates its individual value function in the direction that can maximize the team interest. Finally, we choose a part of a real dataset to evaluate our algorithm and the results show the effectiveness of our algorithm in a comparison with some other existing methods.

key words: *internet of things, edge cloud computing, multi-agent systems, deep reinforcement learning*

1. Introduction

With the rapid development of Internet of Things (IoT), voluminous data is being produced from various IoT devices [1] and such data is usually not useful until it is analyzed well. However, IoT devices usually have poor computational capacities, which requires for computing supplements. Edge cloud computing, as an important computation infrastructure of IoT, has been applied in many IoT scenarios [2], [3]. It includes both cloud servers with high computational resources and edge servers which are closer to the IoT devices and provide cloud-like computation service with low delay [4].

Analyzing the data generated from IoT devices is usually regarded as a task. For instance, analyzing human body data can be regarded as a health monitoring task. Edge servers are usually allocated to various areas and each edge server can offload its tasks to the remote cloud servers, which is called task offloading in distributed edge cloud computing [5]. Specifically, each edge server is accessed by a task queue that holds several tasks at each step. Each edge server observes its own status, such as CPU occupancy rate to decide which tasks are to be performed locally and which tasks are to be offloaded to the cloud servers. Based on those offloading decisions, each edge server's status would be altered and a new task queue would arrive at the next step. Different task offloading decisions usually correspond to different performances such as delay cost and energy cost. Then, how to offload the tasks with the aim of optimizing a certain object like minimizing delay cost is an important research topic in distributed edge cloud computing [5]–[7].

Many studies have examined the task offloading problem in distributed edge cloud computing, however they usually consider the problem at a user level where each user owns an edge server and assume each edge server is self-interested, which corresponds to a non-cooperative setting. Specifically, each user only desires to maximize its own interest, which may cause a conflict between them. For instance, if many edge servers try to offload their tasks to cloud servers with limited channel resources at the same time, the offload rate will decrease due to network congestion. With the high development of smart communities such as smart factory, smart campus and smart hospital, the edge servers are usually owned by an organization like a technology corporation rather than users [8]–[10]. Thus, its goal is to maximize the team reward that emphasizes overall interest of all edge cloud servers rather than each edge server's own interest, which corresponds to a cooperative setting. However, the existing studies are not suitable for this kind of cooperative scenario, making further research essential.

In this paper, we study a new problem called the cooperative task offloading in distributed edge cloud computing. This problem raises two issues: 1) each edge server's status dynamically changes and task arrival is uncertain. 2) each edge server can observe only its own status, which corresponds to a local observation, which makes it hard to achieve cooperation. To cope with these issues, we formulate this problem as a decentralized partially observable Markov de-

Manuscript received June 26, 2021.
Manuscript revised November 16, 2021.
Manuscript publicized December 28, 2021.

[†]The authors are with the Graduate School of Informatics, Kyoto University, Kyoto-shi, 606–8501 Japan.
a) E-mail: dingshiyao0217@gmail.com
b) E-mail: lindh@i.kyoto-u.ac.jp

DOI: 10.1587/transinf.2021DAP0010

cision process (Dec-POMDP) which is a classical model that copes with dynamic decision problems with partial observations. Then, we apply a multi-agent reinforcement learning (MARL) algorithm called value decomposition network (VDN), and propose a VDN-based task offloading algorithm (VDN-TO) to solve it. In VDN-TO, a team value function is used to evaluate team interest and it is then divided into individual value functions for each edge server (regarded as an agent). Each agent updates its individual value function in the direction that can maximize the team reward. Finally, we choose part of a real Google datacenter dataset to conduct evaluations to verify the effectiveness of our proposed algorithm in a comparison with other existing algorithms.

To sum up, our contributions are stated as follows:

- Unlike most existing work on task offloading in distributed edge cloud with a non-cooperative setting, we consider a new problem called cooperative task offloading in distributed edge cloud computing. We formulate it as a Dec-POMDP to handle the features of dynamic changes and partial observations.
- Starting with the VDN algorithm, we propose a novel task offloading algorithm called VDN-TO algorithm to solve this problem. It can make edge servers cooperate with each other to optimize team rewards even under partial observations.
- Evaluations on a Google datacenter dataset verify our VDN-TO algorithm's effectiveness in different settings such as delay-sensitive and energy-sensitive cases.

2. Motivating Scenario

With the strong development of smart communities in IoT [8], [9], edge cloud computing is being used in many scenarios such as smart factories and smart hospitals [10]. In this section, we use the motivating scenario of the smart hospital to illustrate the problem of cooperative task offloading in distributed edge cloud computing. As shown in Fig. 1, there are several inpatient wards in different areas and each ward is equipped with an edge server. Each edge server takes charge of performing the IoT tasks demanded by its inpatient ward. For instance, analyzing patient's body data to monitor his/her health condition can be regarded as a health condition monitoring task. If five patients are in one ward at a moment, a task queue consisting of the five health condition monitoring tasks will arrive at the corresponding edge server. The edge server can decide which tasks are executed locally and which tasks are offloaded to the cloud servers.

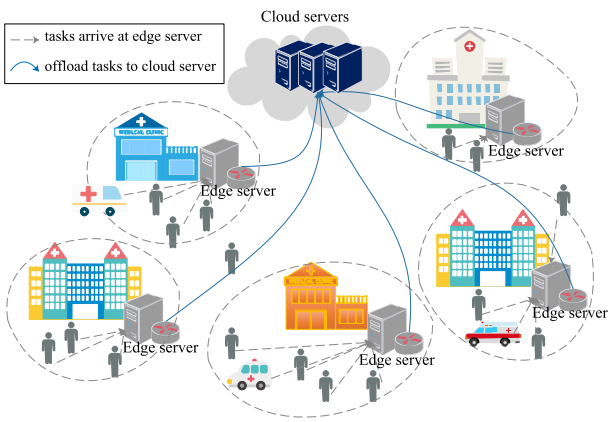

Fig. 1 A distributed edge cloud computing system in a smart hospital.

Since cloud servers usually have more powerful computation capacity than edge servers, it will cost less time if the tasks are performed by cloud servers. However, offloading time would become significant if many edge servers choose to offload their tasks at the same time. This is because the offloading rate usually has an inverse relationship with the number of offloaded tasks. Since all edge servers are owned by one organization, all the edge servers are in cooperative relationships and try to optimize the team interest like minimizing the sum of all IoT task delay costs rather than its own interest. If such cooperation is not achieved well, delay in performing tasks may cause a bad user experience for patients. However, it is difficult to maximize the team interest due to the issues stated in Sect. 1, which requires further study.

3. Related Work

Cloud computing can provide sufficient computational resources, yet it usually incurs huge bandwidth costs and long delays, and so may not satisfy some latency-sensitive IoT applications [11]. Edge computing, as a supplement of cloud computing, can provide cloud-like services and it is usually closer to the IoT devices with controllable latency and low energy consumption [2]. However, edge servers are not as rich in capacities such as processing speed and memory size as cloud servers. Therefore, edge cloud computing has been proposed as a solution applied in many IoT scenarios [3], [12]. In addition, edge servers usually spread across various areas, yielding which is called distributed edge cloud computing.

In distributed edge cloud computing, an important problem is how to offload the tasks arriving at edge servers to cloud servers with the aim of optimizing some objective. Many studies have examined task offloading in distributed edge cloud computing. However, most of them considered the problem in a non-cooperative setting. Chen et al. [5] consider a multi-user computation offloading problem for mobile-edge cloud computing. Each mobile device chooses a channel to offload their computation tasks. However, the uplink data rate is slow, which degrades performance, if many devices choose the same channel. They assume that the mobile devices are non-cooperative and compete for the limited channel resources. They formulate the problem as a theoretic game model and propose a distributed computation offloading algorithm. Liu et al. [6] consider an edge cloud computing network with a three-layer hierarchical architecture consisting of a cloud platform, multiple gateways, and a lot of IoT users. The gateway allows a limited number

of devices to be accessed at the same time, thus each device has a non-cooperative relationship with other devices. They utilize a centralized user clustering method to group the IoT users into different clusters according to user priorities, and offloading proceeds in accordance with user priorities. Chen et al. [7] also consider a non-cooperative environment where each end user observes its local environment to learn optimal decisions for either local computing or edge computing with the goal of minimizing long-term system cost. They formulate it as a stochastic game and propose a fully-decentralized learning method where each agent independently learns its own policy. The above studies do not consider the cooperative setting, where each edge server is assumed to be self-interested, and simply learns its own policy independently. This approach can hardly satisfy the setting wherein all edge servers belong to one organization and the relationship of edge servers is cooperative. Thus, in this paper, we study the new problem of cooperative task offloading in distributed edge cloud computing.

4. Cooperative Task Offloading Problem

We consider the distributed edge cloud computing system shown in Fig. 2, which includes several edge servers distributed among various areas and a cloud server cluster. We regard multiple cloud servers as one single server entity since this paper does not focus on how the tasks are performed in the cloud servers. At each step, each edge server would be accessed by a task queue consisting of several tasks and the edge server decides which tasks to execute locally and which tasks to offload to the cloud servers. Server status like CPU occupancy rate would be altered by allocating the tasks and also influences task performance attributes such as latency and energy consumption. We use cost to evaluate the performance numerically, and consider two types of costs: delay cost and energy cost in this paper. The delay cost consists of computation delay and transmission delay, and the energy cost is incurred by performing/uploading the tasks. We consider the goal of how to minimize team cost in order to ensure all tasks in all edge

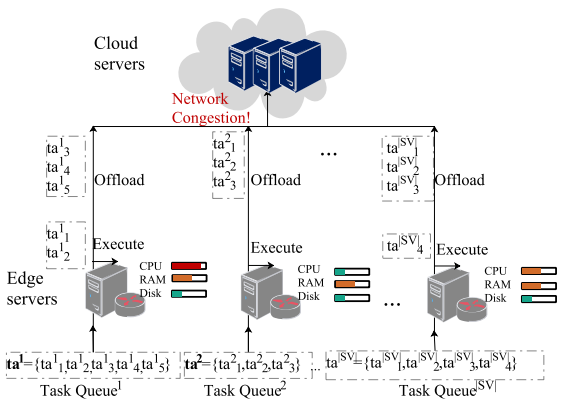

Fig. 2 In a distributed edge cloud computing system, task offloading can trigger network congestion if many edge servers choose to upload tasks at the same time.

servers have good performance. This means that all edge servers should cooperate in deciding task offloading rather than considering just their own interests.

While it seems that each edge can easily acquire status information of the other servers by sending Hello packets, we assume each edge server has only a partial observation in which each edge server cannot observe other server status information. The reasons for this setting are stated as follows: 1) Since sending a packet takes a certain time, collecting the status information of all edge servers will cost more time that exponentially increases with the number of edge servers, which incurs an additional delay cost. This delay cost would become significant, especially if network congestion happens [13]. 2) As each edge server must acquire global information for all edge servers to do decision-making, the size of observation also exponentially increases with the number of edge servers. This would make learning problematic when the number of edge servers is large [7]. 3) Moreover, system robustness is degraded since each edge server cannot make a decision until acquiring all edge server information. If one server fails to send its own information, all other edge servers are unable to make their decisions, which paralyzes the whole edge cloud computing system. Thus, we consider the problem in a decentralized way with partial observations.

Server: An edge cloud computing system consists of several servers $SV = \{sv^0, sv^1, sv^2, \ldots, sv^{|SV|}\}$ where we regard many cloud servers as a single entity with infinite computational resources denoted as sv^0, and sv^i $(i > 0)$ represents the i-th edge server. Each server's parameters are denoted by a vector

$$\mathbf{sv^i} = [f^i, ec^i, CPU^i, RAM^i, Disk^i], \tag{1}$$

where f^i is server i's *computation rate* denoting the executed CPU cycles per second, ec^i is server i's *unit energy cost* denoting the energy consumed per CPU cycle, its unit of measure is J/cyc (J is Joule and cyc is CPU cycle), and it can be calculated via the measurement method stated in [14], CPU^i is server i's available *CPU resource*, RAM^i is server i's available *RAM resource*, $Disk^i$ is server i's available *Disk resource*. Moreover, these three types of resources are altered by pushing/popping tasks. We denote CPU^i_{max}, RAM^i_{max}, $Disk^i_{max}$ as the maximum of these available resources.

Task: At each step, a task queue consisting of several tasks might arrive at edge server sv^i, which is denoted as $\mathbf{ta}^i = \{ta^i_1, ta^i_2, \ldots, ta^i_{|\mathbf{ta}^i|}\}$ with

$$ta^i_j = [work^i_j, b^i_j, Req^{i,CPU}_j, Req^{i,RAM}_j, Req^{i,Disk}_j], \tag{2}$$

where $work^i_j$ is task j's *workload* denoting the CPU cycles required to be performed, b^i_j is task j's *data size* whose unit of measure is *bit*. Although b^i_j can be independent with $work^i_j$, we assume b^i_j has a linear relationship with $work^i_j$ in this paper, i.e., $b^i_j = c * work^i_j$, where c is a constant whose unit of measure is bit/cyc. $Req^{i,CPU}_j$, $Req^{i,RAM}_j$ and

$Req_j^{i,Disk}$ are task j's requirements for CPU, RAM and Disk resources.

Each task would be performed either on edge server or offloaded to the cloud servers. Thus, we refer to some related work to set the costs corresponding to the parts on which they are executed.

Computing at edge server: First, we consider the case when the task is performed on edge servers and refer to [5] to set the following equations. The computation time of the task ta_j^i is given by

$$t_j^i = \frac{work_j^i}{f^i}, \tag{3}$$

where the bigger the f^i is, the less time t_j^i is. The corresponding computation energy is given by

$$e_j^i = ec^i work_j^i, \tag{4}$$

where the bigger the $work_j^i$ is, the more energy it costs. Thus, the sum cost of computation time and energy at the edge servers can be calculated by

$$C_{j,edge}^i = \omega_t t_j^i + \omega_e e_j^i, \tag{5}$$

where $\omega_t, \omega_e \in [0, 1]$ denote the weighting parameters of delay and energy ($\omega_t + \omega_e = 1$).

Computing at cloud server: Similarly, we can calculate the cost while the task is offloaded to the cloud servers. We consider that a base-station is used to establish wireless communication channel between the edge servers and the cloud servers [15]. Since all edge servers belong to one organization, we assume they share one wireless channel. Then, from [5]–[7] offload rate $ur(Up)$ is calculated by

$$\begin{aligned} ur(Up) &= w \log_2\Big(1 + \frac{qg}{\omega + qg \sum_{i\neq 0} |Up^i|}\Big) \\ &= w \log_2\Big(1 + \frac{1}{\frac{\omega}{qg} + \sum_{i\neq 0} |Up^i|}\Big) \\ &= w \log_2\Big(1 + \frac{1}{l + \sum_{i\neq 0} |Up^i|}\Big), \end{aligned} \tag{6}$$

where w is the channel bandwidth whose unit of measure is bit/s, $\log_2\Big(1 + \frac{qg}{\omega + qg \sum_{i\neq 0} |Up^i|}\Big)$ denotes the partition of the channel bandwidth that can be used (dimensionaless) where the more tasks are uploaded, the smaller this value is, q is the edge server's transmission power which is determined by the wireless base-station according to some power control algorithms such as [16], [17], g denotes the channel gain between the edge server and the base-station, $\sum_{i\neq 0} |Up^i|$ is the number of tasks being offloaded from all edge servers, Up^i is the set of tasks being offloaded from server i, Up is the set including all tasks offloaded, ω is the background noise power. Since ω, q and g are constants, we denote $l = \frac{\omega}{qg}$ as the channel coefficient. Thus, the unit of measure of upload rate $ur(Up)$ is bit/s. For instance, let us consider the case where $w = 10bit/s$ and $l = 0$. Then, at a step, if only one task is uploaded, the upload rate will be $10 \log_2(2) = 10bit/s$. If two tasks are uploaded at the same time, the upload rate will be $10 \log_2(3/2) = 5.8bit/s$.

Correspondingly, the offloading time of task $t_{j,off}^i \in Up^i$ is given by

$$t_{j,off}^i(Up) = \frac{b_j^i}{ur(Up)}. \tag{7}$$

After the tasks are offloaded to cloud server sv^0, its corresponding calculation time is given by

$$t_{j,exe}^i = \frac{work_j^i}{f^0}, \tag{8}$$

where f^0 is the computation rate of cloud server sv^0. Since in most IoT environments task arrival might be sparse, this paper assumes that the cloud servers are accessed using in on-demand manner where the user only pays when the cloud servers are in running. Then, the more task workloads (CPU workloads) are uploaded to the cloud servers, the longer usage-time and a higher cost (e.g., fee) would be. Since the energy cost increases in proportion to CPU workload, the evaluations in this paper use energy cost. Specially, the computation energy of executing task t_j^i at cloud server is defined by

$$e_{j,exe}^i = ec^0 work_j^i. \tag{9}$$

From [5], the energy of uploading task t_j^i to the cloud servers is

$$e_{j,up}^i = \frac{qb_j^i}{ur(Up)}. \tag{10}$$

The sum cost of computation time and energy for performing task ta_j^i on cloud server sv^0 can be calculated by

$$C_{j,cloud}^i(Up) = \omega_t(t_{j,off}^i(Up) + t_{j,exe}^i) + \omega_e(e_{j,exe}^i + e_{j,up}^i). \tag{11}$$

Then, at one step the total cost of performing the tasks from all servers can be calculated and is denoted as $Cost$. In this paper, the following three cases are considered respectively.

Latency-sensitive case: only delay cost is desired to be minimized. Then, the total delay cost of both edge and cloud servers can be calculated by

$$C_{time} = \sum_{i=1}^{|SV|}\Big[\sum_{j\notin Up^i} t_j^i + \sum_{j\in Up^i}(t_{j,off}^i(Up) + t_{j,exe}^i)\Big]. \tag{12}$$

Thus, we have $Cost = C_{time}$ in latency-sensitive case.

Energy-sensitive case: only energy cost is desired to be minimized. Then, the total energy cost of both edge and cloud servers can be calculated by

$$C_{energy} = \sum_{i=1}^{|SV|}\Big[\sum_{j\notin Up^i} e_j^i + \sum_{j\in Up^i}(e_{j,exe}^i + e_{j,up}^i)\Big]. \tag{13}$$

Thus, we have $Cost = C_{energy}$ in energy-sensitive case.

Balance case: the sum of delay and energy cost is desired to be minimized, where $Cost$ is defined based on Eqs. (12) and (13).

$$Cost = \omega_t \frac{C_{time} - C_{time}^{min}}{C_{time}^{max} - C_{time}^{min}} + \omega_e \frac{C_{energy} - C_{energy}^{min}}{C_{energy}^{max} - C_{energy}^{min}}, \tag{14}$$

where C_{time}^{max}, C_{time}^{min}, C_{energy}^{max} and C_{energy}^{min} represent the maximum/minimum values of time and energy, separately.

Moreover, the tasks must be accomplished by providing enough computation resources to the edge server. Obviously the total computation resource requirements of all tasks executed at edge server sv^i cannot exceed the maximum computation resource of server sv^i. That is,

$$\begin{aligned} &\sum_{j \notin Up^i} Req_j^{i,CPU} + CPU^i \le CPU_{max}^i, \\ &\sum_{j \notin Up^i} Req_j^{i,RAM} + RAM^i \le RAM_{max}^i, \\ &\sum_{j \notin Up^i} Req_j^{i,Disk} + Disk^i \le Disk_{max}^i. \end{aligned} \tag{15}$$

Thus, the goal is to minimize the discounted cost summation under a period with certain length T, i.e., $\sum_{t=1}^{T} \gamma^t Cost[t]$, where γ is a discount factor denoting the importance of future cost. In conclusion, a cooperative task offloading problem in distributed edge cloud computing is defined as follows.

Definition 1: Given a distributed edge cloud computing system with $|SV|$ servers and a certain period consisting of T steps, the goal is to minimize the discounted cost summation of all servers in an episode with satisfying the constraints of computation resources, i.e.,

$$\begin{aligned} &\min \sum_{t=1}^{T} \gamma^t Cost[t] \\ &\text{s.t. Eq. (15)} \end{aligned} \tag{16}$$

5. Formulate Problem as Dec-POMDP

The problem of cooperative task offloading in distributed edge cloud computing has two distinctive features: partial observations and dynamic changes. Thus, we formulate this problem as a Dec-POMDP which is a classical model for formulating discrete time decision processes with partial observations [18].

In Dec-POMDP, each agent has a local observation and obtains a joint immediate reward depending on the results from all agents. Dec-POMDP can be described as the tuple $<\mathcal{N}, \mathcal{S}, \mathcal{A}^i, O^i, \mathcal{T}, r^i>$, where $\mathcal{N}$ is the set of agents, $\mathcal{S}$ is the set of states, $\mathcal{A}^i$ is the set of agent i's actions and collecting each agent action $a^i \in \mathcal{A}^i$ can form a joint action $a = a^1 \times \cdots \times a^{|\mathcal{N}|} \in \mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^{|\mathcal{N}|}$, O^i is the set of agent i's observation, $\mathcal{T}$ is the state transition function and $\mathcal{T}(s, a, s')$ is the probability of the environment transitioning to state s' after taking joint action a under state s, i.e., $\mathcal{T}(s, a, s') = Prob(s'|s, a)$, r^i is the reward function for agent i and $r^i(s, a, s')$ is the reward obtained by agent i after taking joint action a under state s. How to formulate the cooperative task offloading problem as Dec-POMDP is stated as follows.

Observation: We regard each edge server as an agent, each edge server has only a partial observation consisting of two parts: 1) its own current status, 2) its arriving task queue. Thus, agent i's observation is defined as

$$o^i[t] = \left[\mathbf{sv}^i[t],\ \mathbf{ta}^i[t] \right], \tag{17}$$

where $\mathbf{sv}^i[t]$ is server i's status at step t, and $\mathbf{ta}^i[t]$ is edge server i's task queue arriving at step t.

State: By collecting each edge server's observation, we can form a state defined as

$$s[t] = \left[o^1[t], \ldots, o^{|SV|}[t] \right]. \tag{18}$$

Action: As for observation $o^i[t]$, each edge server would decide which tasks to execute locally or to offload to the cloud servers. Each task offloading set Up^i can be regarded as action $a^i \in \{0, 1\}^{|\mathbf{ta}^i[t]|}$ where 0 represents local computing and 1 represents offloading. For instance, as for task queue $\mathbf{ta}^i = \{ta_1^i, ta_2^i, ta_3^i, ta_4^i\}$, $a^i = [0, 0, 1, 0]$ means that task ta_3^i is offloaded to the cloud servers and the other three tasks are performed locally. Then, we define a joint action at step t which includes the actions from all edge servers as follows.

$$\mathbf{a}[t] = (a^1[t], \ldots, a^{|SV|}[t]). \tag{19}$$

Policy: We define policy function $\pi^i : O^i \times \mathcal{A}^i \to [0, 1]$ for each agent, which means each agent takes an action a^i under its observation o^i based on policy π^i probabilistically. Specially, a ε-greedy policy is used where the agent chooses a current optimal action or randomly chooses action with a certain probability.

Transition function: The server statuses of the next step are deterministically decided after making the task offloading decisions from all servers. However, the task queues arriving at next step are uncertain, which corresponds to a stochastic environment. Thus, the environment transfers to the next state $s[t + 1]$ upon completion of joint action $\mathbf{a}[t]$ based on transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$; this means the environment probabilistically transfers to the next state s', depending on current state s and joint action $\mathbf{a}$, i.e., $\mathcal{T}(s, \mathbf{a}, s') = Prob(s'|s, \mathbf{a})$.

Reward function: As for the transition $(s[t], \mathbf{a}[t], s[t + 1])$, we can obtain cost $Cost[t]$ at step t. In Dec-POMDP, the objective is always to maximize the reward $r(s[t], \mathbf{a}[t], s[t+1])$, thus we take the inverse of $Cost[t]$ as the immediate reward at step t, i.e., $r(s[t], \mathbf{a}[t], s[t + 1]) = -Cost[t]$. Thus, minimizing the delay and energy costs corresponds to maximizing the reward.

Objective function: Let us consider a certain period h with T steps and h can be represented as follows,

$$h = [s[1], \mathbf{a}[1], s[2], \mathbf{a}[2], \ldots, s[T], \mathbf{a}[T], s[T + 1]]. \tag{20}$$

Then, the discounted sum $R(h)$ of the immediate rewards in the period is given by

$$R(h) = \sum_{t=1}^{T} \gamma^{t-1} r(s[t], \mathbf{a}[t], s[t+1]), \tag{21}$$

which is what we want to maximize in one period. Thus, the objective function is given as follows.

$$J(\pi^1, \ldots, \pi^{|SV|}) = \mathbb{E}_{\pi^1,\ldots,\pi^{|SV|},\mathcal{T}}[R(h)], \tag{22}$$

which means we want to identify a couple of policies $(\pi^1, \ldots, \pi^{|SV|})$ for all agents that can maximize the expectation of team rewards during total period h.

6. Algorithm

There are usually two main types of Dec-POMDP: non-cooperative setting and cooperative setting. The non-cooperative setting can be solved by using fully decentralized MARL algorithms such IDQL [19] and IDRQL [20]. Their effectiveness in tackling the task offloading problem in edge clouds has been confirmed in some studies [21]. The cooperative setting considered in this paper can be solved by the value decomposition network (VDN) [22] and QMIX [23], classical cooperative MARL algorithms. In this section, we propose a novel VDN-based task offloading algorithm (VDN-TO) for distributed edge cloud computing. Unlike the traditional VDN which focuses on the general case, VDN-TO is intended to solve the server cooperation problem in edge cloud computing. Specifically, VDN-TO can be divided into two parts: 1) centralized learning and 2) decentralized execution, as shown in Fig. 3.

As for 1) centralized learning, although each agent can independently learn its individual policy by its own individual reward like in [21], it cannot achieve a cooperative behavior since the team reward is not considered. Thus, we consider to train the agents in a centralized way.

As for 2) decentralized execution, the joint action space $|\mathcal{A}|^{|SV|}$ will exponentially increase with the number of agents if we directly choose a joint action from the joint action space $\mathcal{A} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^{|SV|}$. This is called centralized execution which makes the problem suffer the curse of dimensionality. Thus, we consider to make each agent independently choose action a^i from its own action space $\mathcal{A}^i$.

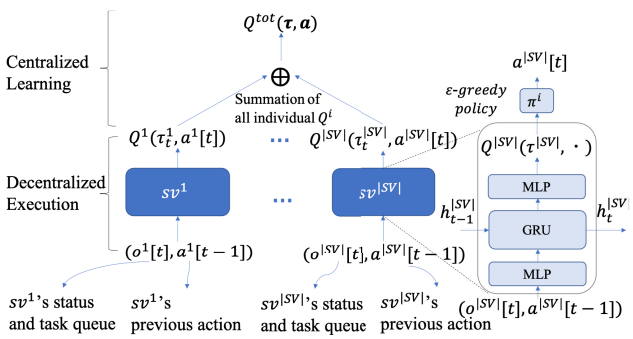

Fig. 3 The framework of value decomposition network based task offloading (VDN-TO) algorithm in distributed edge cloud computing.

6.1 Centralized Learning

In reinforcement learning, state-action value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is normally used to learn the optimal policy for agents; it evaluates the quality that results from taking particular actions in certain states [24], [25]. After learning an optimal state-action value function, an optimal policy can be obtained by taking the action with maximum state-action value (Q-value). In this paper, we consider two types of Q-value: total Q-value Q^{tot} and individual Q-value Q^i. Total Q-value Q^{tot} is used to evaluate joint action $\mathbf{a}$ under the team reward. Then, individual Q-value Q^i is updated in the direction that can maximize Q^{tot} and is used to choose individual action a^i. We elucidate them separately as follows.

Total Q: In MARL, the Q-value for state and joint action is used to denote the discounted sum of rewards that can be obtained in the future after taking joint action $\mathbf{a}$ under state s. When a couple of optimal policies $(\pi^{1*}, \ldots, \pi^{|SV|*})$ are given, it is called optimal Q-value and is defined as

$$Q^*(s, \mathbf{a}) = \mathbb{E}_{\pi^{1*},\ldots,\pi^{|SV|*}}[R(h) \mid s[1] = s, \mathbf{a}[1] = \mathbf{a}],$$

where "$|s[1] = o, \mathbf{a}[1] = \mathbf{a}$" means the initial state and joint action are fixed on state s and $\mathbf{a}$, respectively. $Q^*(s, \mathbf{a})$ is the conditional expectation of $R(h)$ given $s[1] = s, \mathbf{a}[1] = \mathbf{a}$. By recursion, it can be rewritten as follows.

$$Q^*(s, \mathbf{a}) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \mathbf{a}, s')[r(s, \mathbf{a}, s') + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q^*(s', \mathbf{a}')]. \tag{23}$$

However, we cannot obtain the state s in Dec-POMDP as each agent has only its own partial observation o^i. But, agents can benefit from conditioning on their entire action-observation history τ_t^i until step t [23], which is defined by

$$\tau_t^i = [o^i[1], a^i[1], o^i[2], a^i[2], \ldots, o^i[t-1], a^i[t-1], o^i[t]]. \tag{24}$$

We collect all agents' action-observation histories at step t as joint trajectory $\tau_t = [\tau_t^1, \ldots, \tau_t^{|SV|}]$. Then, we replace the global state s with τ_t to define total Q-value Q^{tot} as $Q^{tot} : \Gamma \times \mathcal{A} \to \mathbb{R}$ where Γ is the set of all possible joint trajectories τ. Specially, Q^{tot} is defined as the sum of the individual Q-values of all agents, i.e.,

$$Q^{tot}(\tau, \mathbf{a}; \theta) = \sum_{i=1}^{|SV|} Q^i(\tau^i, a^i; \theta).$$

Then, our goal is to learn an optimal Q^{tot*} that can maximize the sum $R(h)$ of team rewards over period h. Specially, a reply memory D is used to store the tuple of $(\tau_t, r[t])$ where the joint trajectory τ_t stores the observations and actions of all agents upto step t and $r[t]$ is the team reward obtained at step t. Then, we randomly sample the tuples from D to

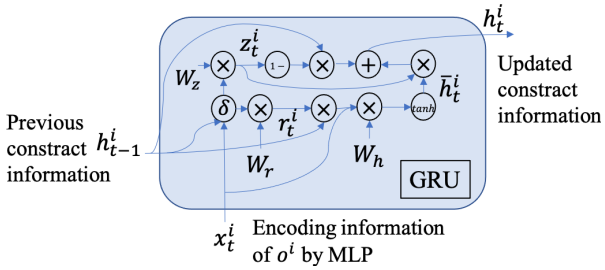

Fig. 4 The detail of GRU module.

train $Q^{tot}(\tau, a)$ with the target of minimizing the following loss function:

$$\mathcal{L}(\theta) = \sum_{k \in D} \Big[\Big(y_k - Q^{tot}(\tau_k, \mathbf{a}_k; \theta)\Big)\Big], \tag{25}$$

where k is the sample index and $y_k = r_k + \gamma \max_{\mathbf{a}'} Q(\tau_k, \mathbf{a}'; \theta)$ is the target, θ is the set of all agent parameters.

Individual Q: We note above that Q^{tot} is the sum of individual Q^i which will be updated with the aim of maximizing Q^{tot} by backpropagation gradients. Specially, the value of Q^i is based on a deep neural network called deep recurrent Q-network (DRQN) [20]. The input of DRQN is each agent's observation, previous action and its history information based on τ^i and the output is the values $Q^i(\tau^i, a^i)$ for each action $a^i \in \mathcal{A}^i$.

In DRQN, a classical recurrent neural network called gated recurrent unit (GRU) is used to process observation-action history. Specially, τ_t^i can be handled by the GRU unit to obtain abstract information h_t^i. As shown in Fig. 4, the current information $(o^i[t], a^i[t-1])$ and the previous abstract information h_{t-1}^i are input to GRU and then the updated abstract information h_t^i is output and used as part of the input in the next step; x_t^i is the result of encoding $(o^i[t], a^i[t-1])$ by a multilayer perceptron (MLP) layer. Specially, h_t^i can be calculated as follows.

$$\begin{aligned} r_t^i &= \delta(W_r \cdot [h_{t-1}^i, x_t^i]), \\ z_t^i &= \delta(W_z \cdot [h_{t-1}^i, x_t^i]), \\ \tilde{h}_t^i &= \tanh(W_h \cdot [r_t * h_{t-1}^i, x_t^i]), \\ h_t^i &= (1 - z_t^i) * h_{t-1}^i + z_t^i * \tilde{h}_t^i, \end{aligned} \tag{26}$$

where $[\,\cdot\,,\,\cdot\,]$ means to connect two vectors and $*$ means the product of two matrices. In this case, since each edge server has the same structure, we assume all agents share one neural network.

6.2 Decentralized Execution

The above sub-setion described how Q^i is trained in a centralized way. However, choosing actions in a centralized way will incurs a huge joint action space, as it increases exponentially with agent number. Thus, we consider the decentralized execution approach where each agent independently chooses action a^i based on its individual Q^i, i.e.,

Algorithm 1 Value Decomposition Network based Task offloading Algorithm (VDN-TO)

```
Initialize replay memory D to capacity N
Initialize DRQL network Q^i with random weights
for episode m=1, M do
    for step t =1, T do
        For each edge server i, the task queue ta^i[t] arrives and its own
        status sv^i[t] is observed to form the observation
        o^i[t] = [sv^i[t], ta^i[t]]
        Add o^i[t] and a^i[t − 1] to the trajectory τ_t^i for each agent i
        Decentralized execution:
        for edge server i =1, |SV| do
            Based on the trajectory τ_{t−1}^i, edge server i randomly selects
            an action a^i from A^i
            otherwise selects a^i[t] = argmax_{a^i} Q^i(τ_t^i, a^i; θ)
        end for
        Collect the actions a^i[t] from all edge servers to form a joint
        action a[t] = [a^1[t], ..., a^{|SV|}[t]]
        Execute joint action a[t] in distributed edge cloud, then transfer
        to the next state s[t + 1]
        Obtain reward r[s[t], a[t], s[t + 1]] and set s[t + 1] = s[t]
        Store transition (τ[t], r[t]) in D
        Centralized learning:
        Sample random mini-batch of transitions from D
        Calculate y_k and perform a gradient descent step on (y_k −
        Q^tot(τ_k, a_k; θ))^2 to update θ
    end for
end for
```

$$\operatorname*{argmax}_{\mathbf{a}} Q^{tot}(\tau, \mathbf{a}) = \begin{pmatrix} \operatorname*{argmax}_{a^1} Q^1(\tau^1, a^1) \\ \operatorname*{argmax}_{a^2} Q^2(\tau^2, a^2) \\ \cdots \\ \operatorname*{argmax}_{a^{|SV|}} Q^{|SV|}(\tau^{|SV|}, a^{|SV|}) \end{pmatrix}.$$

This means that the result of collecting each optimal action by operation $\operatorname*{argmax}_a Q^i(\tau^i, a^i)$ is equivalent to that of directly searching for a joint action by operation $\operatorname*{argmax}_{\mathbf{a}} Q^{tot}(\tau, \mathbf{a})$. To ensure that a global argmax performed on Q^{tot} yields the same result as a set of individual argmax operations performed on each Q^i, monotonicity can be enforced through a constraint on the relationship between Q^{tot} and each Q^i [23], i.e, $\frac{\partial Q^{tot}}{\partial Q^i} \geq 0$. This means that all Q^i functions have the same monotonicity with regard to Q^{tot}. It is easy to prove that VDN-TO satisfies this equation since we have $\frac{\partial Q^{tot}}{\partial Q^i} = 1 > 0$ for each agent.

Finally, the specific process of decentralized execution is shown in the left part of Fig. 3 where each edge server captures its observation o^i, and takes action a^i based on its own policy $\pi^i(\tau^i, a^i)$. Then, all observations o^i are collected to form state $s = (o^1, \ldots, o^{|SV|})$ and all actions are collected to form a joint action $\mathbf{a} = (a^1, \ldots, a^{|SV|})$. The environment transfers to the next state s' based on the transition function $\mathcal{T}(s, \mathbf{a}, s')$ and reward $r(s, \mathbf{a}, s')$ can be obtained. Specifically, our proposed VDN-TO algorithm is performed by Algorithm 1 in the distributed edge cloud computing.

7. Evaluation

This section uses task data from a real dataset to evaluate the performance of our proposed method in performing task offloading in distributed edge cloud computing.

7.1 Evaluation Settings

Task setting: The real data chosen is called google-cluster data; it represents 29 day's worth of Borg cell information from May 2011, on a cluster of about 12.5k servers [26]. The task information includes the computation resources of RAM, CPU and Disk, all of which are used directly as task parameters. Since it does not include the information about task workload, we randomly generate the workload of each task following a uniform probability distribution. Specifically, in this paper we use a uniform distribution among $(0, work_{max})$ where $work_{max}$ is the maximum workload of the tasks, i.e., $work \sim unif(0, work_{max})$.

Since the scenario we considered in this paper is a smart hospital which usually hosts no more than ten servers, we randomly choose five servers' data from two days [26]. The chosen server numbers are {3938719206, 351618647, 329150663, 1303745, 431052910}. Then, each episode is deemed to be finished when all tasks in the task set are been completed.

Neural network setting: The detail of neural network structure in VDN-TO is shown in Fig. 5 (Since each agent has the same neural network framework, we only draw one of them). As for the first MLP, its input is agent i's current observation o^i and its previous action a^i, then the layer size is equal to $|o^i| + |a^i|$. Specifically, each server has five features based on Eq. (1) and each action has dimension of 5 (we assume the maximum number of arriving task at each step is 5, i.e., $|a^i| = ta_{max} = 5$). Thus, the input MLP layer size is 10 and none activation function is used. Then, GRU with 64 neurons takes the results of the first MLP layer and its hidden variable h^i_{t-1} at the last step as the inputs. The activation function of the layer is tanh. Thus, it outputs hidden variable h^i_t with dimension of 64. As for the second MLP, it takes h^i_t as its input and outputs the Q-values for each action. The output layer size $2^{ta_{max}}$ is equal to $2^5 = 32$, since we assume $ta_{max} = 5$. We implemented it using Tensorflow 2.0.

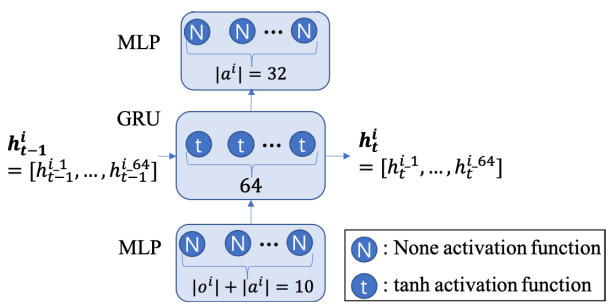

Fig. 5 The detail of neural network structure.

7.2 Evaluation Results

In this paper, the following task offloading (TO) baseline algorithms are adopted: 1) IDQL-TO [21] which is a deep RL based TO approach; 2) QL-TO [27] which is a tabular RL based TO approach; 3) Joint-TO [28] which is a rule-based TO approach where the edge servers are divided to two groups according to the task data size that they need to offload, and only one group of servers can jointly offload tasks; and 4) random policy. Their performance is compared to that of our VDN-TO algorithm. Since the tasks include some random elements which make them different at each episode, we take 5 episodes as one *round* and use the average of $R(h)$ in one round to compare. Specially, each experiment consisted of 500 episodes which corresponds to 100 rounds. We used the same hyperparameters for all RL/DRL based algorithms with $\alpha = 0.01$ and $\gamma = 0.9$.

First, we consider the latency-sensitive case whose objective cost is defined in Eq. (12). The results are shown in Fig. 6. QL-TO approaches the performance of random policy. That is because QL-TO is a tabular method which prevents it from learning an optimal policy in large state space. Since IDQL-TO can well cope with the large state space, it can attain better performance than QL-TO. However, the result of learning is unstable: the range of its oscillation is very large, and sometimes its performance worse than that of random policy even in the final learning phase. This is because each edge server just considers its own interest and they will conflict if they choose to offload many tasks at the same time. Joint-TO can attain better performance than QL-TO and is more stable than IDQL-TO. However, its performance does not improve since it does not learn over the iterations. On the other hand, our proposed VDN-TO algorithm uses a total Q and each edge server tries to optimize it in a cooperation way, which yields good and stable performance.

Second, we consider the energy-sensitive case whose objective cost is defined in Eq. (13). The results are shown in Fig. 7. Since cloud servers have much higher unit energy

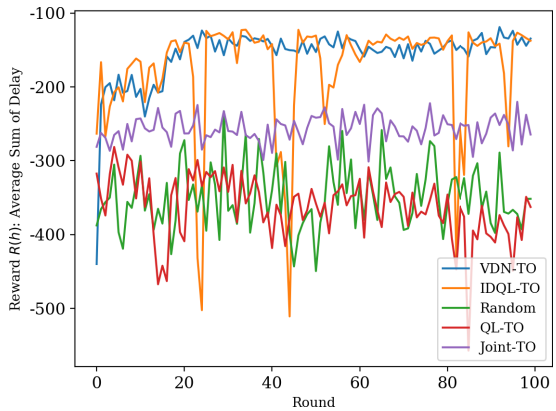

Fig. 6 Comparing the performances of VDN-TO with IDQL-TO, QL-TO, Joint-TO and random policy in latency-sensitive case.

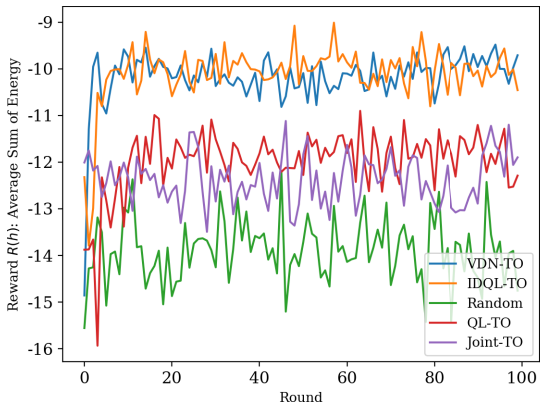

Fig. 7 Comparing the performances of VDN-TO with IDQL-TO, QL-TO, Joint-TO and random policy in energy-sensitive case.

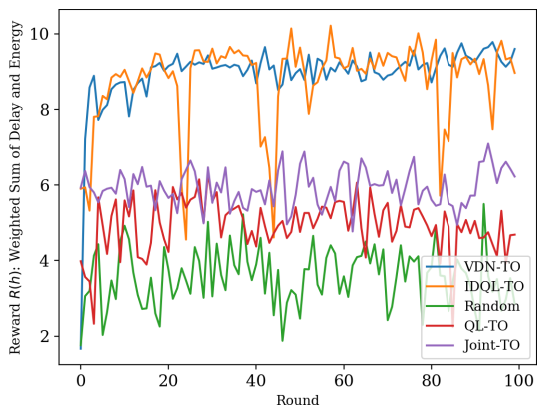

Fig. 8 Comparing the performances of VDN-TO with IDQL-TO, QL-TO, Joint-TO and random policy in balance case.

cost than edge servers, the optimal policy is to execute all tasks at edge servers in this energy-sensitive case. Thus, each edge server's maximized interest can result in maximizing the team interest. Moreover, each agent's optimal policy does not influence other agents' optimal policies. Although QL-TO still suffers the large state space problem, its learning environment becomes more stable (non-interest-conflict) than in the latency-sensitive case so its performance is superior to that of random policy. Joint-TO can get the same performance as QL-TO in a stable way. However, its performance does not improve since it does not learn over the iterations. In this non-interest-conflict situation, each edge server's maximized interest is consistent with team maximized interest, thus the self-interested IDQL-TO can also achieve a performance as good as VDN-TO. Our proposed VDN-TO algorithm can still learn optimal policies in a stable manner.

Third, we consider a balance case whose objective cost is defined in Eq. (14). Without losing generality, we set $\omega_t = 0.5$, $\omega_e = 0.5$ in Eq. (14). The results are shown in Fig. 8. Although QL-TO still suffers the large state space problem, its learning environment is more stable than in the latency-sensitive since the energy-sensitive part is included. Thus, its performance is better than that of random policy. Joint-TO still maintains a stable performance due to its characteristic of rule-based. Although IDQL-TO has similar performance with VDN-TO, it exhibits oscillation. Since the latency part requires a cooperative setting under non-interest-conflict situation, which can be well handled by VDN-TO, the performance of VDN-TO is better than that of IDQL-TO.

To summarize the above experiments, we can conclude that our proposed VDN-TO algorithm can solve the cooperative task offloading problem in distributed edge cloud computing. It exceeds the performance of other baseline algorithms in three classical settings of edge cloud computing: latency-sensitive case, energy-sensitive case, and a balance between latency and energy. Moreover, we consider a decentralized setting and the centralized manner is not considered in this paper. Although the centralized manner can evaluate the results of joint action more accurately (it treats the problem as a single agent problem), it might yield better performance than cooperation under partial observations. However, this advantage might be effective only if the number of edge servers is small, which is seldom the case in real-world scenarios.

8. Conclusion

In this paper, we studied the new problem of cooperative task offloading in edge cloud computing, with the target of maximizing team reward. We proposed a value decomposition network based task offloading algorithm called VDN-TO to guide the edge servers towards cooperating with each other. We validated our approach by using a real dataset to compare it with baseline algorithms. The results showed our approach achieved significantly bigger rewards than the baseline algorithms. We plan to conduct a larger scale experiment to examine the effectiveness of our proposed algorithm in more depth.

Acknowledgments

This research was partially supported by a Grant-in-Aid for Scientific Research (B) (21H03556, 2021–2024), and a Grant-in-Aid for Challenging Exploratory Research (20K21833, 2020–2023) from the Japan Society for the Promotion of Science (JSPS).

References

[1] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I.A.T. Hashem, A. Siddiqa, and I. Yaqoob, "Big IoT data analytics: Architecture, opportunities, and open research challenges," IEEE Access, vol.5, pp.5247–5261, 2017.

[2] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," IEEE Wireless Commun. Lett., vol.6, no.6, pp.774–777, 2017.

[3] H. Chang, A. Hari, S. Mukherjee, and T. Lakshman, "Bringing the cloud to the edge," IEEE Conference on Computer Communications Workshops, pp.346–351, 2014.

[4] S. Li and J. Huang, "Energy efficient resource management and task

scheduling for IoT services in edge computing paradigm," IEEE International Symposium on Parallel and Distributed Processing with Applications, pp.846–851, 2017.
[5] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," IEEE/ACM Trans. Netw., vol.24, no.5, pp.2795–2808, 2015.
[6] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource offloading with edge computing in IoT networks via machine learning," IEEE Internet Things J., vol.7, no.4, pp.3415–3426, 2020.
[7] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," IEEE Internet Things J., vol.6, no.3, pp.4005–4018, 2018.
[8] H. Nishi, "Information and communication platform for providing smart community services: System implementation and use case in Saitama city," IEEE International Conference on Industrial Technology, pp.1375–1380, 2018.
[9] S. Donovan, J. Chung, M. Sanders, and R. Clark, "MetroSDX: A resilient edge network for the smart community," IEEE International Conference on Pervasive Computing and Communications Workshops, pp.575–580, 2017.
[10] A.A. Abdellatif, A. Mohamed, C.F. Chiasserini, M. Tlili, and A. Erbad, "Edge computing for smart health: Context-aware approaches, opportunities, and challenges," IEEE Netw., vol.33, no.3, pp.196–203, 2019.
[11] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," Proc. First Edition of the MCC Workshop on Mobile Cloud Computing, pp.13–16, 2012.
[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet Things J., vol.3, no.5, pp.637–646, 2016.
[13] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," EURASIP Journal on Wireless Communications and Networking, pp.1–21, 2020.
[14] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," IEEE International Conference on Computer Communications, pp.2716–2720, 2012.
[15] D. López-Pérez, X. Chu, A.V. Vasilakos, and H. Claussen, "On distributed and coordinated resource allocation for interference mitigation in self-organizing LTE networks," IEEE/ACM Trans. Netw., vol.21, no.4, pp.1145–1158, 2013.
[16] M. Xiao, N.B. Shroff, and E.K.P. Chong, "A utility-based power-control scheme in wireless cellular systems," IEEE/ACM Trans. Netw., vol.11, no.2, pp.210–221, 2003.
[17] M. Chiang, P. Hande, T. Lan, and C.W. Tan, Power control in wireless cellular networks, Now Foundations and Trends, 2008.
[18] F.A. Oliehoek and C. Amato, A concise introduction to decentralized POMDPs, Springer Briefs in Intelligent Systems, Springer, 2016.
[19] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," PloS One, vol.12, no.4, e0172395, 2017.
[20] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents, pp.29–37, 2015.
[21] X. Liu, J. Yu, Z. Feng, and Y. Gao, "Multi-agent reinforcement learning for resource allocation in IoT networks with edge computing," China Communications, vol.17, no.9, pp.220–236, 2020.
[22] P. Sunehag, G. Lever, A. Gruslys, W.M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J.Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multiagent learning based on team reward," The 17th International Conference on Autonomous Agents and Multiagent Systems, no.3, pp.2085–2087, 2017.
[23] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," International Conference on Machine Learning, pp.4295–4304, 2018.
[24] C.J.C.H Watkins and P. Dayan, "Q-learning," Machine Learning, vol.8, no.3-4, pp.279–292, 1992.
[25] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," Nature, vol.518, no.7540, pp.529–533, 2015.
[26] C. Reiss, J. Wilkes, and J.L. Hellerstein, "Google cluster-usage traces: Format + schema. Technical report," Google Inc., Mountain View, CA, USA, 2011.
[27] F. Jiang, W. Liu, J. Wang, and X. Liu, "Q-learning based task offloading and resource allocation scheme for internet of vehicles," IEEE/CIC International Conference on Communications in China, pp.460–465, 2020.
[28] S. Barbarossa, S. Sardellitti, and P.D. Lorenzo, "Joint allocation of computation and communication resources in multiuser mobile cloud computing," IEEE 14th Workshop Signal Process. Adv. Wireless Commun., pp.26–30, 2013.

Shiyao Ding is a Ph.D. student in informatics from Kyoto University, Japan since October 2019. He received the Master degree in engineering from Osaka University, Japan in September 2019. His current research interests include reinforcement learning, multiagent systems, services computing, Internet of Things, edge computing and cloud computing.

Donghui Lin received the Ph.D. degree in informatics from Kyoto University, Japan in 2008. He was a researcher at National Institute of Information and Communications Technology (NICT), Japan in 2008–2011. He then joined, in 2012, the Department of Social Informatics of Kyoto University, where he is an associate professor since 2018. His current research interests include services computing, Internet of Things, multiagent systems, and intercultural collaboration. He was a recipient of the 2012 Achievement Award of the Institute of Electronics, Information and Communication Engineers (IEICE), Japan. He has been serving as a program committee member for major international conferences in the areas of services computing and multiagent systems, including IEEE SCC, IEEE ICWS and AAMAS.