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

Akito Suzuki
NTT Network Service Systems Laboratories
NTT Corporation
Tokyo, Japan
akito.suzuki.tw@hco.ntt.co.jp

Masahiro Kobayashi
NTT Network Service Systems Laboratories
NTT Corporation
Tokyo, Japan
masahiro.kobayashi.pk@hco.ntt.co.jp

ICC 2022 - IEEE International Conference on Communications | 978-1-5386-8347-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICC45855.2022.9838540

***Abstract*—Edge computing is a new paradigm to provide computing capability at the edges close to end devices. A significant research challenge in edge computing is finding an efficient task offloading to edge and cloud servers, considering various task characteristics and limited network and server resources. Several studies have proposed the reinforcement learning (RL) based task offloading method, because RL can immediately output the efficient offloading by pre-learning. However, due to the performance problem of RL, these previous studies do not consider clouds or focus only on a single cloud. They also do not consider the bandwidth and topology of the backbone network. Such shortcomings could lead to degrading the performance of task offloading. Therefore, we formulated a task offloading problem for multi-cloud and multi-edge networks, considering network topology and bandwidth constraints. Moreover, we proposed a task offloading method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL) to solve the performance problem of RL. This method introduces a cooperative multi-agent technique through centralized training and decentralized execution, improving the efficiency of task offloading. Simulations revealed that the proposed method drastically reduces the average latency while satisfying all constraints, compared with the greedy approach. It also revealed that the proposed cooperative learning method improves the efficiency of task offloading.**

***Index Terms*—Edge Computing, Deep Reinforcement Learning, Task Offloading**

I. Introduction

With the development of communication technologies, diverse applications have emerged in various domains, e.g., healthcare, smart cities, and manufacturing. These applications are generally offloaded and processed in the cloud servers because of the limitation of the computation resources of end devices (EDs), e.g., personal computers, smartphones, Internet of Things (IoT) devices, and cars, whose system is called cloud computing (CC). These offloaded applications' tasks consist of the demand for computing and communication with various characteristics, e.g., traffic heavy, computing heavy, or latency sensitive. Since cloud servers are generally located far from EDs, offloading tasks to the cloud inevitably generates additional transmission latency. Therefore, CC degrades the performance of latency-sensitive tasks.

To cope with the issue, edge computing (EC) has been proposed to deploy computing resources at the edge servers close to EDs. Combining CC and EC provides multiple offloading choices, improving the efficiency of offloading. For example, offloading computing-heavy tasks to the cloud is effective because the cloud usually has sufficient computing resources. Similarly, offloading traffic-heavy and latency-sensitive tasks to the edge is effective to shorten the transmission latency. Thus, combining CC and EC is essential to improve the efficiency of offloading tasks with various characteristics.

Several studies have addressed task offloading problems for CC and EC [1]–[7]. Specifically, reinforcement learning (RL) [8] has been focused on as a solution [4]–[7]. RL can immediately output the preferred task offloading by learning the relationship between input network patterns and output task offloading in advance. However, existing methods based on mathematical optimization still have problems with computational cost. For example, Wang et al. [1] accelerates the computation time more than 100 times using their parallel optimization framework. However, due to the computational cost, they could only evaluate it in a small network with one cloud and a few base stations (BSs) (i.e., a few edges).

Although RL can solve the problem of computation time, two issues remain in the above studies. One is that existing studies [4]–[6] do not consider CC or target only the network with a single cloud. As mentioned above, combining CC and EC is inevitable to improve task offloading efficiency. Moreover, a typical network has multiple clouds. The other is that existing studies [4]–[7] do not consider the bandwidth capacity and the backbone network topology. Many studies try to minimize task latency by shortening the path that offloaded tasks take. When a control system does not consider the bandwidth capacity, some links may become congested due to the concentration of tasks. Therefore, the system should obtain the task offloading that minimizes the latency while satisfying all link bandwidth constraints. In addition, some of the above studies assume a typology that is not practical as a backbone network, e.g., a full-mesh network.

Multi-agent RL is an effective way to deal with more complex problems. It is a system of multiple agents interacting within a common environment. Each agent works together with the other agent(s) to achieve a team reward. The learning cost of each agent can be reduced by assigning each agent to each task. However, when training decentralized and independent agents to optimize the team reward, each agent faces a non-stationary learning problem [9]. An example

of the problem is that when each agent independently acts simultaneously, all tasks will be allocated on the light-load cloud, resulting in server overload. To avoid the non-stationary problem, Zhan et al. [4] proposed the algorithm combining multi-agent RL and game theory. However, the actions of decentralized agents based on game theory fall into a sub-optimal solution of Nash equilibrium.

In this paper, we formulate an optimal task offloading problem for multi-cloud and multi-edge networks, considering network topology and bandwidth constraints. Here, optimal offloading is defined as a solution that maximizes server and link resource efficiency and minimizes task latency while satisfying the constraints of server and link capacity and task latency. The decision variables are the computing resource allocation of tasks and the routing between the ED and the allocated server. We also propose a task offloading method based on cooperative multi-agent deep reinforcement learning (**Coop-MADRL**). We assign an agent at each edge that has learned the optimal task offloading. Moreover, we introduce a cooperative training technique in which several agents jointly optimize a single reward in a *centralized training and decentralized execution* manner, which can prevent the non-stationary problem and improve the task offloading efficiency.

The rest of the paper is organized as follows. Section II describes our task offloading problem. Section III briefly reviews RL. Section IV describes the Coop-MADRL-based task offloading method. Section V evaluates its performance, and Section VI concludes the paper.

II. Problem Definition

A. Overview

We describe an overview of the task offloading problem. We assume a network consisting of the EDs, edges, and clouds. EDs generate various tasks with diverse applications. Each task consists of required computing demand, traffic demand, and maximum permissible latency to accomplish the task. Each ED can compute its tasks locally or offload tasks to the neighboring edge or cloud. We collectively refer to edges and clouds as nodes. All nodes have computing resources to execute the tasks instead of EDs, called edge or cloud servers. All nodes also have a function of traffic forwarding to another node as a router. Only each edge has a function that determines the optimal node to offload each task. Note that we assume that the ED decides whether or not to offload its tasks, and optimizing the ED's decision is outside the scope of this paper.

We describe the procedure of our task offloading system. We consider a discrete time-step t. We assume that each ED has one or more tasks and consider $\mathcal{K}$ tasks during $t \in [0, T]$. At the beginning of each t, each task arrives at the nearest edge of each ED. Each edge observes the information of tasks and network usage. On the basis of the observation, our method deployed in each edge calculates the optimal node to offload the task (see Section IV for details). When multiple tasks arrive simultaneously in each edge, our method repeats determining the offloading node in a first-in first-out (FIFO) manner. After that, our system aggregates the traffic demand information between nodes and then calculates and updates the optimal route between nodes. Next, each edge forwards tasks to the optimal nodes through the optimal route, and the node executes the task and returns the result to the ED. After a certain amount of time, it proceeds to the next $t + 1$. The executing tasks continue to consume the resources of the offloaded node and the link(s) it traverses until it returns a result to the ED. The tasks accepted at t do not need to be completed before $t + 1$.

Note that it is also adequate to calculate and update the route once every few steps. In this case, our system calculates the optimal route on the basis of the average traffic volumes between nodes for several steps.

TABLE I
Symbol descriptions for physical network

Symbol	Definition
$G(\boldsymbol{N}, \boldsymbol{L})$	Network graph
$n \in \boldsymbol{N}$	Node
$\text{link}\,(i,j) \in \boldsymbol{L}$	Link from node i to node j
$e \in \boldsymbol{E} \subset \boldsymbol{N}$	Edge
$c \in \boldsymbol{C} \subset \boldsymbol{N}$	Cloud
v_i^N	i^{th} node processing capability
w_i^N	i^{th} node capacity
w_{ij}^L	(i,j) link capacity
α_{ij}^L	Distance coefficient of link (i,j)

TABLE II
Symbol descriptions for tasks

Symbol	Definition
$t \in T$	Time-step (T: Total time steps)
$\mathcal{D} := \{\boldsymbol{D}_k\}$	Task set ($k \in \mathcal{K}$, $\mathcal{K}$: Total tasks)
t_k	Acceptance time of k^{th} task
β_k	Task type of k^{th} task
C_k	Computing demand of k^{th} task
$B_k^{\text{up}}, B_k^{\text{down}}$	Upload/Download traffic demand of k^{th} task
$\tau_k^{\max}$	Maximum permissible latency of k^{th} task

B. Network Model

Table I summarizes the definitions of the network model variables. We consider the physical-network graph $G(\boldsymbol{N}, \boldsymbol{L})$ consisting of a physical node set $\boldsymbol{N}$ and a physical link set $\boldsymbol{L}$. We assume that each physical node has a role as an edge or cloud. We denote the edge as $e \in \boldsymbol{E} \subset \boldsymbol{N}$ and the cloud as $c \in \boldsymbol{C} \subset \boldsymbol{N}$. We also denote the numbers of nodes, edges, and clouds as $|\boldsymbol{N}|$, $|\boldsymbol{E}|$, and $|\boldsymbol{C}|$. We assume that EDs connect to the nearest edge through the access network, which is not included in $G(\boldsymbol{N}, \boldsymbol{L})$ in this paper. We also assume that the routes in the access network take the shortest path, and route optimization in the access network is outside the scope of this paper. We denote the node processing capability of i^{th} node as $v_i^N \in \mathbb{R}^+$, which indicates the limit of computing resources, e.g., the central processing unit (CPU) capability per second in i^{th} node ([G cycles/s]). Here, $\mathbb{R}^+$ indicates the set of positive real number. We also denote the node capacity of i^{th} node as $w_i^N \in \mathbb{N}$, which indicates the upper limit of the number of allocated tasks. For example, when we assign one CPU core to each task, w_i^N is equal to the number of CPU cores in

i^{th} node. We denote the bandwidth capacity of link (i,j) as $w_{ij}^L \in \mathbb{R}^+$, which indicates the limit of bandwidth resources ([Mbps]). All links also have transmission latency depending on the distance between each node. We define the distance coefficient of link (i,j) as $\alpha_{ij}^L \in \mathbb{R}^+$, which determines the latency of each link. A concrete example is shown in Fig. 1.

C. Task Model

Table II summarizes the definitions of the task model variables. We describe a task model for uniformly representing various demands of EDs. We define the task set as $\mathcal{D} = \{\boldsymbol{D}_k\}$ and the k^{th} task as $\boldsymbol{D}_k := [t_k, \beta_k, C_k, B_k^{\text{up}}, B_k^{\text{down}}, \tau_k^{\max}]$. Here, $t_k \in T$ is the accepted time of k^{th} task, $\beta_k \in \mathbb{N}$ is the task type uniquely given for each application, C_k is the required computing demand ([G cycles]), $B_k^{\text{up}} \in \mathbb{R}^+$ and $B_k^{\text{down}} \in \mathbb{R}^+$ are the required upload and download traffic demand ([Mbits]), and $\tau_k^{\max} \in \mathbb{R}^+$ is the maximum permissible latency to accomplish the k^{th} task ([ms]). The computing demand refers to the total amount of CPU cycles needed to accomplish the task. The traffic demand refers to the amount of traffic generated by the task. The reason for distinguishing between upload and download traffic is to accommodate tasks with varying traffic depending on the computing process. When $\boldsymbol{D}_k$ is accepted, it consumes the computing and bandwidth resources on $G(\boldsymbol{N}, \boldsymbol{L})$ depending on the amount of computing and traffic demand of $\boldsymbol{D}_k$. If the task is assigned to the edge closest to the ED, the bandwidth resources consumed on $G(\boldsymbol{N}, \boldsymbol{L})$ are regarded as 0. Note that if the task parameters are unknown, it is necessary to use different systems to estimate the parameters, which is out of the scope of this paper.

D. Optimization Problem

We formulate the task offloading problem, which minimize Eq. (1) while satisfying constraints Eqs. (2)–(17). Table III summarizes the definitions of the variables of the task offloading problem. We aim to find optimal task allocation variables $\boldsymbol{Y}$ and routing variables $\boldsymbol{X}_t$. Here, $\boldsymbol{Y} := \{y_{kn}\}$ shows the task allocation in which y_{kn} is 1 if the computing demand of the k^{th} task is assigned to the n^{th} node; otherwise, 0. $\boldsymbol{X}_t := \left\{x_{ij,t}^{pq}\right\}$ shows the proportion of traffic demand m_t^{pq} from origin node p to destination node q passing through link (i,j) at t. Here, $\boldsymbol{M}_t := \{m_t^{pq}\}$ shows the traffic demand matrix between node p and node q at t. Since we assume multi-path routing, $x_{ij,t}^{pq}$ can take a continuous value between 0 and 1. We also define the ED placement as $\boldsymbol{Z} := \{z_{ke}\}$, in which z_{ke} is 1 if the nearest edge of the ED requested k^{th} task is the e^{th} edge; otherwise, 0. We assume that $\boldsymbol{Z}$ is constant during $t \in [0, T]$.

We introduce an objective function:

$$\min : \sum_{t \in T} \left(U_t^N + U_t^L\right) + \lambda \sum_{k \in \mathcal{K}} \left(\tau_k^N + \tau_k^L\right), \tag{1}$$

where U_t^N and U_t^L show the maximum node and link utilization at t, which are defined as $U_t^N := \max_i(\boldsymbol{U}_t^N)$ and $U_t^L := \max_{ij}(\boldsymbol{U}_t^L)$. Here, we denote the i^{th} node utilization as $u_{i,t}^N \in \boldsymbol{U}_t^N$ and the link (i,j) utilization as $u_{ij,t}^L \in \boldsymbol{U}_t^L$. The τ_k^N and τ_k^L show the node and link latency of k^{th} task, whose formulations are described below. The λ indicates the weighting parameter determining the importance ratio of resource efficiency and task latency.

TABLE III
SYMBOL DESCRIPTIONS FOR TASK OFFLOADING

Symbol	Definition
$\boldsymbol{X}_t := \left\{x_{ij,t}^{pq}\right\}$	Proportion of passed m_t^{pq} on link (i,j)
$\boldsymbol{Y} := \{y_{kn}\}$	Task allocation (task k, node n)
$\boldsymbol{M}_t := \{m_t^{pq}\}$	Traffic matrix from node p to node q
$\boldsymbol{Z} := \{z_{ke}\}$	Device placement (task k, edge e)
$\boldsymbol{U}_t^N := \left\{u_{i,t}^N\right\}$	i^{th} node utilization at step t
$\boldsymbol{U}_t^L := \left\{u_{ij,t}^L\right\}$	(i,j) link utilization at step t
$U_t^N = \max_i\left(\boldsymbol{U}_t^N\right)$	Maximum node utilization at step t
$U_t^L = \max_{ij}\left(\boldsymbol{U}_t^L\right)$	Maximum link utilization at step t
τ_k^N, τ_k^L	Node latency and link latency of k^{th} task
λ	Weighting parameter of objective function
$\mathbb{I}_{k,t}$	Binary variable indicating executing tasks
$\mathcal{P}_k^{\text{up}}$, $\mathcal{P}_k^{\text{down}}$	Upload and download path set of k^{th} task
τ_p	Latency coefficient of path p
$\mathcal{L}_p$	Link set along with path p
r_p	Traffic splitting ratio of path p

We impose three kinds of constraints: node capacity, link capacity, and task latency. We first define the binary variable $\mathbb{I}_{k,t}$ as follows.

$$\mathbb{I}_{k,t} := \begin{cases} 1 & (t_k \le t \le t_k + \tau_t^N + \tau_t^L) \\ 0 & (\text{otherwise}), \end{cases} \tag{2}$$

where $\mathbb{I}_{k,t}$ is 1 if the k^{th} task is executing at t; otherwise, 0. Here, t_k is the acceptance time of k^{th} task and $t_k + \tau_t^N + \tau_t^L$ shows the completion time of k^{th} task.

1) Node Capacity Constraints: A task allocation variable y_{kn} is formulated to minimize the maximum node utilization U_t^N while satisfying the node capacity constraints as follows.

$$\text{s.t.} \quad : \quad \sum_{n \in \boldsymbol{N}} y_{kn} = 1 \qquad (\forall k \in \mathcal{K}) \tag{3}$$

$$\sum_{k \in \mathcal{K}} C_k \, \mathbb{I}_{k,t} \, y_{kn} \le w_n^N U_t^N \qquad (\forall n \in \boldsymbol{N}) \tag{4}$$

$$y_{kn} \in \{0,1\} \qquad (\forall k \in \mathcal{K}, \forall n \in \boldsymbol{N}) \tag{5}$$

$$0 \le U_t^N \le 1 \tag{6}$$

Equation (3) shows that the computing demand of each task must be allocated to any nodes. Equation (4) shows the constraint of node capacity. The term $\mathbb{I}_{k,t}\, y_{kn}$ in Eq. (4) shows the task allocation of executing tasks at t.

2) Link Capacity Constraints: A routing variable $x_{ij,t}^{pq}$ is formulated to minimize the maximum link utilization U_t^L while satisfying the link capacity constraints as follows.

$$\text{s.t.} \quad : \quad \sum_{j:(i,j) \in L} x_{ij,t}^{pq} - \sum_{j:(j,i) \in L} x_{ji,t}^{pq} = 0 \qquad (\forall p, q \in \boldsymbol{N}, i \ne p, i \ne q) \tag{7}$$

$$\sum_{j:(i,j) \in L} x_{ij,t}^{pq} - \sum_{j:(j,i) \in L} x_{ji,t}^{pq} = 1 \tag{8}$$

$$(\forall p, q \in \boldsymbol{N}, i = p)$$

$$\sum_{p,q \in \boldsymbol{N}} m_t^{pq} x_{ij,t}^{pq} \leq w_{ij}^L U_t^L \tag{9}$$

$$(\forall (i,j) \in L, \forall p, q \in \boldsymbol{N})$$

$$0 \leq x_{ij,t}^{pq} \leq 1 \quad (\forall (i,j) \in L, \forall p, q \in \boldsymbol{N}) \tag{10}$$

$$0 \leq U_t^L \leq 1 \tag{11}$$

Equations (7)–(8) show the traffic flow conservation law. Equation (9) shows the constraint of link capacity. The m_t^{pq} in Eq. (9) can be formulated as follows.

$$m_t^{pq} = \sum_{k \in \mathcal{K}} B_k^{\mathrm{up}} \, \mathbb{I}_{k,t} \, z_{kp} \, y_{kq} \tag{12}$$

$$m_t^{qp} = \sum_{k \in \mathcal{K}} B_k^{\mathrm{down}} \, \mathbb{I}_{k,t} \, z_{kp} \, y_{kq} \tag{13}$$

$$(\forall p \in \boldsymbol{E}, \forall q \in \boldsymbol{N}, p \neq q)$$

Equation (12) shows the upload traffic demands from origin node p to destination node q. Here, z_{kp} and y_{kq} decide the node p and node q, and $\mathbb{I}_{k,t}$ extracts the executing tasks. Equation (13) shows the download traffic demands from node q to node p, which is the opposite of the upload.

3) Latency Constraints: We first formulate the node latency τ_k^N and link latency τ_k^L of k^{th} task as follows.

$$\tau_k^N := \sum_{n \in \boldsymbol{N}} y_{kn} \frac{C_k}{v_n^N} \tag{14}$$

$$\tau_k^L := \max_{p \in \mathcal{P}_k^{\mathrm{up}}} \left(\tau_p B_k^{\mathrm{up}}\right) + \max_{p \in \mathcal{P}_k^{\mathrm{down}}} \left(\tau_p B_k^{\mathrm{down}}\right) \tag{15}$$

Equation (15) shows that τ_k^L is determined by the bottleneck path with the maximum latency when the task goes through multiple paths. Here $\mathcal{P}_k^{\mathrm{up}}$ and $\mathcal{P}_k^{\mathrm{down}}$ are the set of upload and download paths of k^{th} task, which is calculated on the basis of $\boldsymbol{X}_t$, $\boldsymbol{Y}$ and $\boldsymbol{Z}$. The $\tau_p \in \mathbb{R}^+$ is the latency coefficient of path p considering the link distance coefficient α_{ij}^L and link capacity w_{ij}^L. The τ_p is formulated as follows.

$$\tau_p := r_p \cdot \frac{1}{\min_{(i,j) \in \mathcal{L}_p} (w_{ij}^L)} \cdot \sum_{(i,j) \in \mathcal{L}_p} \alpha_{ij}^L, \tag{16}$$

where r_p is the traffic splitting ratio of path p calculated by $\boldsymbol{X}_t$. The $\mathcal{L}_p$ is the link set along with path p. The term $\min_{(i,j) \in \mathcal{L}_p} (w_{ij}^L)$ indicates the bottleneck bandwidth on the path p. Finally, the latency constraint is formulated as follows.

$$\text{s.t.} \quad : \quad \tau_k^N + \tau_k^L \leq \tau_k^{\max} \qquad (\forall k \in \mathcal{K}) \tag{17}$$

III. Reinforcement Learning

A. Single-Agent Reinforcement Learning

Single-agent RL solves the decision problem in which an agent interacts with an environment. The agent observes state $s \in \mathcal{S}$, takes action $a \in \mathcal{A}$, receives reward r, and transfers to the new state $s' \in \mathcal{S}$. The goal of the agent is to maximize the action value $Q(s, a)$, which is defined as the expectation of the sum of rewards obtained in the future when action a is selected in state s. DRL [10], [11] can handle continuous and high-dimensional state space by approximating the $Q(s, a)$ with a deep neural network (DNN). It was reported that the performance of RL algorithms drastically worsens as the number of candidate actions increases [11]. Therefore, stable DRL becomes difficult as the number of actions increases.

B. Multi-Agent Reinforcement Learning

A multi-agent environment can be described consisting of $\langle N, \mathcal{S}, \mathcal{A}, R, P, \mathcal{O}, \gamma \rangle$, in which N is the number of agents, $\mathcal{S}$ is state space, $\mathcal{A} = \{\mathcal{A}^1, \ldots, \mathcal{A}^N\}$ is the set of actions for all agents, P is the transition probability among the states, R is the reward function, and $\mathcal{O} = \{\mathcal{O}^1, \ldots, \mathcal{O}^N\}$ is the set of observations for all agents. The i^{th} agent at t observes the own local observation o_t^i, takes action a_t^i ($\boldsymbol{a}_t = \{a_t^i\}$), and receives reward r_t. Each agent has an observation-action history $h^i \in \boldsymbol{h}$, where the history indicates a series of past observations and $\boldsymbol{h}$ is the observation-action history of all agents.

1) Independent Q-Learning: The most naive approach to solve the multi-agent RL problem is to treat each agent independently. This idea is formalized in an independent Q-learning (IQL) algorithm [12], [13], which decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment.

2) Value Decomposition Networks: Value decomposition networks (VDNs) [14] aim to learn a joint action-value function $Q_{tot}(\boldsymbol{h}, \boldsymbol{a})$. It is assumed that the joint action-value function Q_{tot} can be additively decomposed into N Q-value functions for N agents, in which each Q-value function Q_i only relies on the local state-action history:

$$Q_{tot}(\boldsymbol{h}, \boldsymbol{a}) = \sum_{i=1}^{N} Q_i(h^i, a^i). \tag{18}$$

Each agent observes its local state, obtains the Q-values for its action, and selects an action, and then the sum of Q-values for the selected action of all agents provides the total Q-value of the problem. Because each agent's DNN is updated on the basis of the total Q-value, each agent learns the optimal behavior for all agents, i.e., they learn cooperative behavior.

IV. Proposed Method

A. Modeling

We first define the variables that represent a subset of tasks as follows.

$$\mathcal{K}_t := \{k \in \mathcal{K} \mid \mathbb{I}_{k,t} = 1\} \tag{19}$$

$$\mathcal{K}_e := \{k \in \mathcal{K} \mid z_{ke} = 1\} \tag{20}$$

$$\mathcal{D}_t := \{\boldsymbol{D}_k \in \mathcal{D} \mid t_k = t\} \tag{21}$$

$$\mathcal{D}_{e,t} := \{\boldsymbol{D}_k \in \mathcal{D} \mid t_k = t, k \in \mathcal{K}_e\} \tag{22}$$

The $\mathcal{K}_t$ indicates the index subset of tasks executed at time-step t. The $\mathcal{K}_e$ indicates the index subset of tasks accepted at edge e. The $\mathcal{D}_t$ indicates the subset of tasks accepted at time-step t. The $\mathcal{D}_{e,t}$ indicates the subset of tasks accepted on edge e at time-step t, i.e., $\mathcal{D}_{e,t} \subset \mathcal{D}_t \subset \mathcal{D}$.

Table IV summarizes the definitions of the variables of Coop-MADRL. We introduce $|\boldsymbol{E}|$ agents equal to the number

TABLE IV
SYMBOL DESCRIPTIONS FOR COOP-MADRL

Symbols	Definitions
$t^{\mathrm{sim}} \in T^{\mathrm{sim}}$	Time-step for simulator (T^{sim}: Total time-steps)
$\mathcal{G} := \{g_e\}$	Agent set ($1 \leq e \leq \lvert\boldsymbol{E}\rvert$)
$s_t \in \mathcal{S}$	State at step t ($\mathcal{S}$: State space)
$\mathcal{O} := \{\mathcal{O}^e\}$	Observation sets for all agents
$o_t^e \in \mathcal{O}^e$	Observation for agent g_e at step t
$\boldsymbol{o}_t := \{o_t^e\}$	All observation at step t
$\mathcal{A} := \{\mathcal{A}^e\}$	Action sets for all agents ($\mathcal{A}^e$: Action space)
$a_t^e \in \mathcal{A}^e$	Action for agent g_e at step t
$\boldsymbol{a}_t := \{a_t^e\}$	All action at step t
r_t	Reward for agent g_e at step t
$Q_e\left(o_t^e, a_t^n\right)$	Action-value function for agent g_e
$Q_{tot}(\boldsymbol{o}_t, \boldsymbol{a}_t)$	Joint action-value function for all agent
$\mathcal{M}$	Replay memory
$h^i \in \boldsymbol{h}$	observation-action history
$\boldsymbol{h}$	observation-action history of all agents

of edges and assign each agent for each edge offloading control. The e^{th} agent $g_e \in \mathcal{G}$ learns how to optimize task offloading for the e^{th} edge. A state is defined as $s_t = [\mathcal{D}_t, \boldsymbol{U}_t^L, \boldsymbol{U}_t^S]$. An observation for agent g_e is defined as $o_t^e = [\mathcal{D}_{e,t}, \boldsymbol{U}_t^L, \boldsymbol{U}_t^S]$. The candidate action set $\mathcal{A}^e$ is defined as the set of nodes that offload tasks. When edge e does not receive any tasks at t, agent g_e chooses the action "do nothing." We excluded nodes from $\mathcal{A}^e$ that have no remaining resources and edges that were more than two hops away from the edge e. We designed the reward function to return a negative value if the constraints are not satisfied; otherwise, a positive value depends on the objective function value (see Section IV-D for details).

B. Formulation

Coop-MADRL-based method consists of centralized training and decentralized execution. The decentralized agents continually execute the task offloading after centralized training.

Algorithm 1 shows the centralized training using Coop-MADRL. Line 1 shows the initialization of agent parameters. A series of procedures (lines 2–18) is repeatedly executed until learning is complete. Lines 3–4 show the generation of tasks and the initialization of environment parameters. A series of actions is called an episode, and each episode (lines 5–16) is repeatedly executed. In each episode, agents collect learning samples that are combinations of $\langle \boldsymbol{o}_t, \boldsymbol{a}_t, r_t \rangle$. We denote a time-step for the network simulator as $t^{\mathrm{sim}} \in T^{\mathrm{sim}}$, which is reset at the beginning of each episode. In line 7, when edge e accepts multiple tasks at t^{sim}, agent g_e selects one task in a FIFO manner. Here, the two task subsets can be written as following relationship: $\mathcal{D}_{e,t} \subseteq \mathcal{D}_{e,t^{\mathrm{sim}}} (\subset \mathcal{D})$. In line 9, a random action is selected with probability ε; otherwise, an action that maximizes $Q_e\left(o_t^e, a'\right)$ is selected with probability $1 - \varepsilon$. This is to avoid convergence to a local optimum solution. Each agent executes lines 7–9 in parallel. In line 10, task offloading is updated in accordance with $\boldsymbol{a}_t$ by Alg. 3. Line 11 shows the reward calculation. Lines 12–13 mean the termination condition of agent learning. In this algorithm, $r_t \leq -1$ is the terminate condition, i.e., the state that does not satisfy at least one constraint. In lines 14–15, if all tasks accepted at t^{sim} are allocated, it proceeds to the next $t^{\mathrm{sim}} + 1$. Line 17 shows stores in replay memory $\mathcal{M}$. In line 18, all agents $\mathcal{G}$ are trained by the history of episodic transition, which is randomly taken from $\mathcal{M}$.

Algorithm 2 shows the proposed task offloading method using the Coop-MADRL algorithm. Line 1 shows the pre-training of $\mathcal{G}$ by using Alg. 1. Next, this algorithm continually repeats lines 2–9 as long as the system accepts new tasks. In line 6, each agent selects an a_t^e that maximizes $Q_e\left(o^e, a^e\right)$.

Algorithm 1 Centralized training of Coop-MADRL

1: **initialize:** agent parameters, $t \leftarrow 0$
2: **while** $t < T$ **do**
3: generate tasks $\mathcal{D}$ for training
4: **initialize:** environment parameters
5: **for** $t^{\mathrm{sim}} = 0, T^{\mathrm{sim}}$ **do**
6: **for each** $g_e \in \mathcal{G}$ **do**
7: $\mathcal{D}_{e,t} \leftarrow$ select one task $\left(\mathcal{D}_{e,t^{\mathrm{sim}}}\right)$
8: $o_t^e \leftarrow$ observation $\left(\mathcal{D}_{e,t}, \boldsymbol{U}_t^L, \boldsymbol{U}_t^S\right)$
9: $a_t^e \leftarrow$ select epsilon greedy action $\left(o_t^e\right)$
10: update environment $\left(\boldsymbol{o}_t, \boldsymbol{a}_t\right)$ by Alg. 3
11: $r_t \leftarrow$ calculate reward by Alg. 4
12: **if** $r_t \leq -1$ **then**
13: terminate episode: $t^{\mathrm{sim}} \leftarrow T^{\mathrm{sim}}$
14: **if** all $\mathcal{D}_{t^{\mathrm{sim}}}$ are allocated **then**
15: $t^{\mathrm{sim}} \leftarrow t^{\mathrm{sim}} + 1$
16: $t \leftarrow t + 1$
17: store episodic transition $\left(\boldsymbol{o}_j, \boldsymbol{a}_j, r_j\right), \forall j \in$ episode steps
18: train all agents $\mathcal{G}$ by random episodic transition

Algorithm 2 Decentralized execution of Coop-MADRL

1: $t^{\mathrm{sim}} \leftarrow 0$, $Q_e\left(o^e, a^e\right) \leftarrow$ train all agents $\mathcal{G}$ by Alg. 1
2: **while** True **do**
3: **for each** $g_e \in \mathcal{G}$ **do**
4: $\mathcal{D}_{e,t} \leftarrow$ select one task $\left(\mathcal{D}_{e,t^{\mathrm{sim}}}\right)$
5: $o_t^e \leftarrow$ observation $\left(\mathcal{D}_{e,t}, \boldsymbol{U}_t^L, \boldsymbol{U}_t^S\right)$
6: $a_t^e \leftarrow \arg\max_{a' \in \mathcal{A}^e} Q_e\left(o_t^e, a'\right)$
7: update environment $\left(\boldsymbol{o}_t, \boldsymbol{a}_t\right)$ by Alg. 3
8: **if** all $\mathcal{D}_{t^{\mathrm{sim}}}$ are allocated **then**
9: $t^{\mathrm{sim}} \leftarrow t^{\mathrm{sim}} + 1$

Algorithm 3 Update environment

$\boldsymbol{Y} \leftarrow$ allocate tasks $\left(\boldsymbol{a}_t, \mathcal{D}_t\right)$
$\boldsymbol{U}_t^N \leftarrow$ calculate node utilization $\left(\mathcal{D}_{\mathrm{t}}, \boldsymbol{Y}\right)$
$\boldsymbol{M}_t \leftarrow$ calculate traffic matrix $\left(\mathcal{D}_t, \boldsymbol{Y}, \boldsymbol{Z}\right)$
$\boldsymbol{U}_t^L, \boldsymbol{X}_t \leftarrow$ calculate link utilization $\left(\boldsymbol{M}_{\mathrm{t}}\right)$
$\tau_k^N, \tau_k^L \leftarrow$ calculate latency $\left(\boldsymbol{X}_{\mathrm{t}}, \boldsymbol{Y}, \boldsymbol{Z}\right)$
return $\boldsymbol{U}_t^N, \boldsymbol{U}_t^L, \tau_k^N, \tau_k^L$

Algorithm 4 Reward calculation

$r_t \leftarrow \mathrm{Eff}\left(U_{t+1}^L\right) + \mathrm{Eff}\left(U_{t+1}^N\right) + \mathrm{Eff}\left(\tau_t^{\mathrm{ave}}\right)$
if $U_{t+1}^L > 1$ or $U_{t+1}^L > 1$ **then**
return -2
return $\max(-2, \min(2, r_t))$

C. Update Environment

Algorithm 3 shows the procedure of the update environment. The task allocation variables $\boldsymbol{Y}$ and the routing variables $\boldsymbol{X}_t$ are updated in Alg. 3, which concept is based on the proposed methods presented in [15], [16]. Line 1 shows the calculation of $\boldsymbol{Y}$. Line 2 shows the calculation of $\boldsymbol{U}_t^N$. Line 3 shows the calculation of $\boldsymbol{M}_t$ using Eqs. (12)–(13). Line 4 shows the calculation of $\boldsymbol{U}_t^N$. In line 4, we solve the route-optimization problem, which calculates $\boldsymbol{X}_t$ using Eqs. (7)–(11). Line 5 shows the calculation of latency. Finally, Alg. 3 returns variables for the reward calculation.

D. Reward Calculation

We design the reward function on the basis of the objective function Eq. (1). Algorithm 4 shows the procedure of the reward calculation for $\mathcal{G}$. The term $\mathrm{Eff}(x)$ shows the efficiency function and is defined as follows:

$$\mathrm{Eff}(x) = \begin{cases} -x + 0.8 & (x \leq 0.8) \\ -2x + 1.6 & (0.8 < x \leq 1) \\ -2x & (1 < x). \end{cases} \quad (23)$$

We designed this function so that the efficiency decreases as x increases. It returns a positive value depending on x if $x < 0.8$; otherwise it returns a negative value. The decrease of efficiency doubles when x is more than 0.8. The term τ_t^{ave} shows the average satisfaction of latency, which is defined as follows:

$$\tau_t^{\mathrm{ave}} = \sum_{k \in \mathcal{K}_t} \left(\frac{\tau_k^N + \tau_k^L}{\tau_k^{\max}} \right) / |\mathcal{K}_t|. \quad (24)$$

The τ_k^N and τ_k^L is calculated by Eqs. (14)–(15). If it does not satisfy $U_{t+1}^L > 1$ or $U_{t+1}^L > 1$, it return -2. Finally, it returns a clipped reward within $-2 \leq r_t \leq 2$.

V. Evaluation

We evaluated the performance of the proposed method through simulations. For each agent's DNN layer, we introduced deep recurrent Q-networks (DRQN) [17]. We used a three-layer DNN consisting of two fully connected layers and the gated recurrent unit (GRU) layer [18]. We adopted Double-DQN [11] as the DRL algorithm. We implemented the DRL algorithm based PyTorch [19] and PyMARL [20]. We solved the route optimization using the GNU Linear Programming Kit (GLPK) [21].

A. Evaluation conditions

We set the number of tasks to $\mathcal{K} = 1000$, the total time steps to $T = 2.0 \times 10^5$, the total time steps of each episode to $T^{\mathrm{sim}} = 100$ and the weighting parameter to $\lambda = 1$. We assume that each ED has one task during $t^{\mathrm{sim}} \in [1, T^{\mathrm{sim}}]$, the acceptance time t_k is randomly generated within 1–100, and the ED placement $\boldsymbol{Z}$ is randomly generated. We prepared four types of tasks assuming various use cases. Table V summarizes the parameters for task generation. The task parameters are reset at the beginning of each episode. For the physical-network conditions, we prepared the 12-node topology consisting of 9 edges and 3 clouds as shown in Fig. 1.

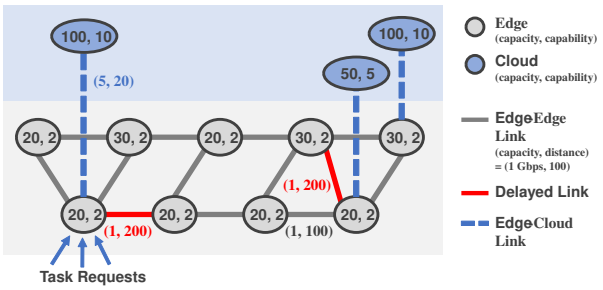

Fig. 1. Network Topology. The values in nodes and links show (w_i^N, v_i^N) and $(w_{ij}^L, \alpha_{ij}^L)$. We increased the latency of two links (red lines).

We describe our methods (VDN) and two comparison methods (IQL and greedy algorithm). VDN indicates the Coop-MADRL-based method shown in Alg. 2. IQL indicates the MADRL-based method without cooperation, i.e., each agent learns on the basis of their reward. Note that single-agent DRL was not evaluated because learning is clearly unsuccessful due to requirements for huge training iterations until the Q-values for all actions are sufficiently close to the optimal. The greedy algorithm (GA) has a policy that allocates tasks to the nearest edge until the node utilization of edge exceeds a threshold and allocates tasks to the nearest cloud after exceeding a threshold. In this evaluation, we set the threshold to 0.6. Note that GA also optimizes routes calculated with the route-optimization algorithm at each t.

B. Performance Evaluation

Figure 2 shows the average performance of each method. We carried out 20 calculations with random initial conditions and set the same random seeds for all evaluations. The width of each bar indicates the standard deviation ($\pm\sigma$). The performance of each method can roughly be compared with the average reward. We also investigate the details of performance as shown in the 2–5$^{\mathrm{th}}$ metrics in Fig. 2. The performance metrics are the average U_t^N, average U_t^L, average constraint violation, and average total latency $\tau_k^N + \tau_k^L$ ([ms]).

Figure 2 shows that VDN performed better than IQL for all metrics. This result indicates that the coordination among agents by VDN improves the performance of task offloading and can prevent constraint violations. On the other hand, since each agent of IQL acts independently, task offloading is concentrated on lightly loaded resources in some cases, causing constraint violations as described in Introduction. The result also shows that VDN performed better than GA for average reward and latency. However, VDN has a higher maximum node and link utilization than GA. The reason is that agents chose the action that maximized the average reward even if the node and link utilization were increased. Since our model considers network constraints and topology, the agent chose the action that reduces the latency as much as possible within the constraints. We suppose that VDN can reduce latency by using neighbor lightly loaded edges and prioritizing allocating computing-heavy tasks in the cloud. We also suppose that VDN can calculate the solution that

TABLE V
TASK GENERATION PARAMETERS

Definitions	Type 1: Basic Task	Type 2: Download-heavy Task	Type 3: Computing-heavy Task	Type 4: Latency-sensitive Task
B_k^{up} [Mbits]	0–50	0–1	90–100	0–50
B_k^{down} [Mbits]	0–50	90–100	0–1	0–50
C_k [G cycles]	0.1, 0.2, 1, 2	0.1, 0.2	10, 20	0.1, 0.2, 1, 2
$\tau_k^{\max}$ [ms]	$10(B_k^{\mathrm{up}} + B_k^{\mathrm{down}}) + 500C_k$	$10(B_k^{\mathrm{up}} + B_k^{\mathrm{down}}) + 500C_k$	$10(B_k^{\mathrm{up}} + B_k^{\mathrm{down}}) + 250C_k$	$5(B_k^{\mathrm{up}} + B_k^{\mathrm{down}}) + 500C_k$
Ratio	30%	20%	20%	20%

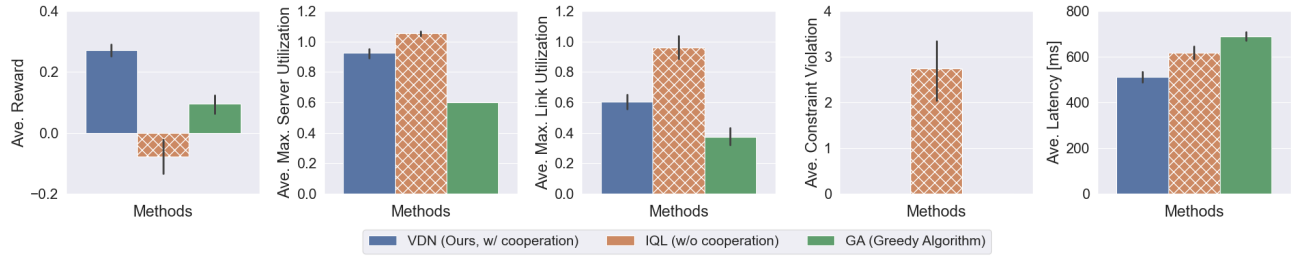

Fig. 2. Performance evaluation for each method

minimizes the average U_t^N and U_t^L when not considering the latency in the objective function. We conclude that VDN learns more superior policies through cooperative learning.

We mention the computation time of training and execution of agents. We used Intel core i9-9980HK for the evaluation. VDN and IQL took about 1 day to finish the training. The execution time per t of all methods was less than 1 s. This result shows that the RL-based method performs sufficiently in terms of computation time.

VI. CONCLUSION

We formulated of an optimal task offloading problem for multi-cloud and multi-edge networks, considering network topology and bandwidth constraints. We proposed a cooperative task offloading method based on cooperative multi-agent deep reinforcement learning (Coop-MADRL). This method can quickly obtain efficient task offloading by learning the relationship between network demand patterns and optimal task offloading by using deep reinforcement learning in advance. Moreover, this method introduces a cooperative multi-agent technique, improving the efficiency of task offloading. Simulations revealed that the proposed method drastically reduces the average latency while satisfying all constraints. It also revealed that the proposed cooperative learning method improves the efficiency of task offloading.

For future work, we plan to evaluate the performance of the proposed method in terms of computation time and scalability and compare it with existing methods. We also plan to evaluate the performance of the proposed method in more complicated use cases or real-world applications.

REFERENCES

[1] Y. Wang *et al.*, "Cooperative task offloading in three-tier mobile computing networks: An ADMM framework," *IEEE Trans. Veh. Technol.*, vol. 68, no. 3, pp. 2763–2776, 2019.

[2] H. Yuan and M. Zhou, "Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems," *IEEE Trans. Autom. Sci. Eng.*, 2020.

[3] C. Kai *et al.*, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 7, no. 2, pp. 624–634, 2020.

[4] Y. Zhan *et al.*, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, 2020.

[5] D. C. Nguyen *et al.*, "Deep reinforcement learning for collaborative offloading in heterogeneous edge networks," in *Proc. IEEE/ACM CCGrid*, 2021, pp. 297–303.

[6] W. Hou *et al.*, "Multi-agent deep reinforcement learning for task offloading and resource allocation in cybertwin based networks," *IEEE Internet Things J.*, 2021.

[7] Y. Zhang *et al.*, "Distributed multi-cloud multi-access edge computing by multi-agent reinforcement learning," *IEEE Trans. Wireless Commun.*, vol. 20, no. 4, pp. 2565–2578, 2020.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[9] G. J. Laurent *et al.*, "The world of independent learners is not markovian," *International Journal of Knowledge-based and Intelligent Engineering Systems*, vol. 15, no. 1, pp. 55–64, 2011.

[10] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[11] H. Van Hasselt *et al.*, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI*, 2016, pp. 2094–2100.

[12] M. TAN, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. ICML*, 1993.

[13] A. Tampuu *et al.*, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[14] P. Sunehag *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward." in *Proc. AAMAS*, 2018, pp. 2085–2087.

[15] A. Suzuki *et al.*, "Extendable NFV-integrated control method using reinforcement learning," *IEICE Trans. Commun.*, vol. E103.B, no. 8, pp. 826–841, 2020.

[16] A. Suzuki *et al.*, "Cooperative multi-agent deep reinforcement learning for dynamic virtual network allocation," in *Proc. ICCCN*, 2021, pp. 1–11.

[17] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc. AAAI Fall Symposium Series*, 2015.

[18] J. Chung *et al.*, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. NIPS Workshop*, 2014.

[19] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. NIPS*, 2019, pp. 8026–8037.

[20] M. Samvelyan *et al.*, "The StarCraft Multi-Agent Challenge," *CoRR*, vol. abs/1902.04043, 2019.

[21] "GLPK," https://www.gnu.org/software/glpk/.