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 Knowledge-Based Software Engineering

Deep Coalitional Q-Learning for Dynamic Coalition Formation in Edge Computing

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

SUMMARY With the high development of computation requirements in Internet of Things, resource-limited edge servers usually require to cooperate to perform the tasks. Most related studies usually assume a static cooperation approach which might not suit the dynamic environment of edge computing. In this paper, we consider a dynamic cooperation approach by guiding edge servers to form coalitions dynamically. It raises two issues: 1) how to guide them to optimally form coalitions and 2) how to cope with the dynamic feature where server statuses dynamically change as the tasks are performed. The coalitional Markov decision process (CMDP) model proposed in our previous work can handle these issues well. However, its basic solution, coalitional Q-learning, cannot handle the large scale problem when the task number is large in edge computing. Our response is to propose a novel algorithm called deep coalitional Q-learning (DCQL) to solve it. To sum up, we first formulate the dynamic cooperation problem of edge servers as a CMDP: each edge server is regarded as an agent and the dynamic process is modeled as a MDP where the agents observe the current state to formulate several coalitions. Each coalition takes an action to impact the environment which correspondingly transfers to the next state to repeat the above process. Then, we propose DCQL which includes a deep neural network and so can well cope with large scale problem. DCQL can guide the edge servers to form coalitions dynamically with the target of optimizing some goal. Furthermore, we run experiments to verify our proposed algorithm's effectiveness in different settings.

key words: *internet of things, edge computing, multi-agent systems, coalition structure generation, reinforcement learning, MDP, deep Q-network*

1. Introduction

With the high development of Internet of Things (IoT), edge computing has become a fundamental infrastructure of IoT, since it can provide low latency and low energy consumption for performing tasks [1], [2]. However, in IoT environments one resource-limited edge server usually cannot perform the tasks whose workloads are high, and so the cooperation of multiple edge servers is required [3], [4]. However, most related studies [5]–[8] assume a static cooperation approach like assigning a server as a helper server to perform the tasks offloaded from the other servers. However, the static cooperation approach might not suit the dynamic environment of edge computing.

In this paper, we study a new approach that encourages edge servers to dynamically cooperate while avoiding the conventional solution of using fixed cooperation rules. This dynamic cooperation problem raises two issues as follows. 1) If we call the servers that can cooperate to perform the tasks a coalition, then guiding them to optimally form coalitions is difficult. 2) Moreover, edge computing yields a dynamic environment where server statuses like CPU occupancy rate usually change as the tasks are pushed/popped. For instance, allocating more tasks to a server raises its CPU occupancy rate. A higher CPU occupancy rate usually corresponds to a lower computation speed which degrades latency performance.

As for issue 1), coalition structure generation (CSG), as a classical theoretic model for studying cooperation problems, is often used to solve cooperation problems by forming coalitions among agents. Specifically, it guides a set of agents to form several coalitions to obtain more rewards than possible with a single agent [9]–[11]. CSG will be solved once an optimal coalition formation structure is located. However, as stated in issue 2), edge server status such as CPU occupancy rate, dynamically changes with task execution and definitively influences task performance [12]. Thus, while coalition structures also need to change to suit this dynamic, CSG cannot cope with this kind of dynamic. As for issue 2), although dynamic features can be formulated as a Markov decision formation (MDP), a classical model for dynamic decision, it does not include the feature of coalition formation. Therefore, how to realize the dynamic coalition formation of edge servers requires a new theoretical model.

In our previous work [13], we proposed a model called coalitional Markov decision process (CMDP) to formulate the problem of dynamic coalition formation to solve the above two issues. We also proposed a basic algorithm called coalitional Q-learning (CQL) to solve CMDP. Although the problem considered in this paper can be formulated as a CMDP, CQL cannot be directly used to solve it. The reason is stated as follows. If we regard the statuses of the servers and the task set as states, the state space exponentially increases with the number of servers and tasks. CQL is a tabular method that maintains a table to evaluate each decision under each state. Thus, a large state space requires CQL to maintain a table so large that it becomes infeasible.

To resolve the above issues, we first recast the problem into CMDP as it can well reflect the features of dynamic coalition formation. At each step of CMDP, the current edge server statues can be observed, and a coalition structure consisting of several coalitions is determined. Then, each coalition of servers chooses a task (action) to perform cooperatively and both statuses of server and task set change accord-

Manuscript received May 10, 2021.
Manuscript revised October 22, 2021.
Manuscript publicized December 14, 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.2021KBP0007

ingly. In the next step, a new coalition structure is decided to repeat the above process. Unlike ordinary CSG, where the goal is to maximize the reward in one step, the goal of CMDP is to optimize the sum of rewards over all steps in a trajectory. To ensure that CMDP can be applied to large state spaces, we propose a novel method called deep coalitional Q-learning (DCQL). It can well cope with large state spaces, since it includes a deep neural network, in which several layers of neural units are progressively constructed to map the input (state) to an output.

To sum up, our contributions are stated as follows:

- Unlike the static cooperation approaches used in most studies, we consider a dynamic cooperation approach for edge computing and formulate it as a CMDP where the server coalitions can change to suit the dynamic characteristics of environment.
- Since the basic algorithm to solve CMDP, CQL, cannot cope with large state spaces, we propose a novel algorithm called DCQL to handle the scale issue.
- We run experiments that show edge servers can dynamically form coalitions to perform tasks cooperatively. Experiment results verify that our proposed DCQL algorithm is more effective than CQL, especially in large state spaces.

2. Motivating Scenario

As shown in Fig. 1, there is an edge computing system consisting of one centralized coordinator, seven resource-limited edge servers and a task set consisting of several tasks. Each task has a different workload and a corresponding number of servers is required to perform the tasks to satisfy some requirement such as maximum latency. We assume each box represents one unit of workload (e.g., 1000 million CPU cycles) and one edge server can perform just one unit of workload. For instance, task 2 requires at least three edge servers to form a coalition to perform cooperatively. Moreover, each edge server's status (defined as edge server i's occupancy rate $or_i \in \{1\%, 2\%, \ldots, 100\%\}$ which is inverse proportion to the number of tasks allocated on it) is dynamically altered by performing tasks and influences the task performances. When tasks are executed, the corresponding energy would be consumed. We assume that the energy consumed by each server is directly related to the occupancy rate or_i and so raising the occupancy rate increases the energy consumed.

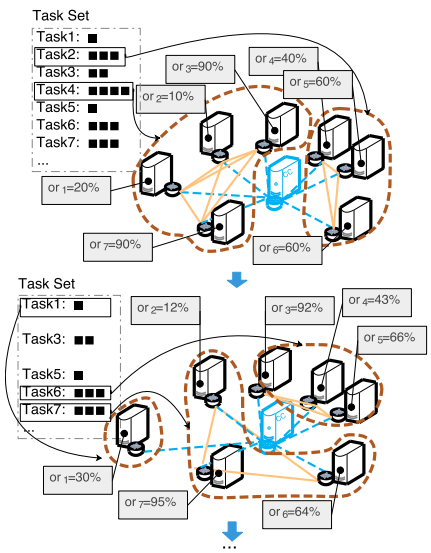


Fig. 1 Dynamic coalition formation in an edge computing system.

In order to describe this dynamic, the statuses of all edge servers and the task set are regarded as state s. In each state, a coalition structure is determined and each coalition chooses a task to perform. Then, each or_i and the task set would be altered after performing the tasks (corresponds to a new state). Correspondingly, the coalition structure needs to be redetermined according to the new state, as is shown in Fig. 1. Besides energy cost, changing the coalition structure of servers in edge computing at each step incurs a cost. That is because, forming a coalition of servers usually requires the participating servers to construct signal channels. Thus, formation/release of coalitions would need the construction/release signal channels which would incur some costs. Thus, the goal of the problem is to make the servers form coalitions to perform all the tasks as soon as possible, while minimizing both the costs of energy consumed and changes of coalition structures.

This scenario well reflects the difficulties in dynamic coalition formation in edge computing: it includes both features of dynamic and coalition formation where the coalitions should change to suit the state at each step. Moreover, the state space is huge: the state size exponentially increases with the number of either edge servers or tasks. For instance, each server has 100 possible statuses (or_i: $0.01 \sim 1$ with scale interval of 0.01), thus three servers has already corresponded to 10^6 joint statuses.

3. Preliminaries

3.1 Coalition Structure Generation (CSG)

Let us consider a set $\mathcal{N} = \{1, \ldots, n\}$ of agents. A coalition is formed by several agents joining which is denoted as c, i.e., $c \subseteq \mathcal{N}$. Then $v_{cha} : 2^{\mathcal{N}} \to \mathbb{R}$ is the characteristic function used to evaluate the value of a coalition c ($2^{\mathcal{N}}$ is the set of all possible coalitions). Since each agent can only choose one coalition to join, several disjoint coalitions can be formed where each way of forming coalitions is called a coalition structure denoted as cs, i.e., $cs = \{c_1, \ldots, c_{|cs|}\}$ that satisfies $(\forall i \neq j, c_i \cap c_j = \emptyset) \wedge (\bigcup_{i=1}^{|cs|} c_i = \mathcal{N})$. The goal of CSG is to identify an optimal coalition structure $cs*$ that has a maximized sum of all coalition values, i.e., $cs^* = argmax_{cs \in \mathcal{P}^{\mathcal{N}}} \sum_{c_i \in cs} v_{cha}(c_i)$ where $\mathcal{P}^{\mathcal{N}}$ is the set of all possible coalition structures. Unlike our proposed CMDP model, CSG would be finished once the optimal coalition structure has been found.

3.2 Markov Decision Process (MDP)

MDP, a traditional model for discrete time decision-making process, is often used to model agent's dynamic behavior [14]. In MDP, a subject that makes a decision is called an agent and the other subject that is impacted by the agent is called the environment. The agent observes state $s \in \mathcal{S}$ of the environment ($\mathcal{S}$ is the set of states can be observed), and takes action $a \in \mathcal{A}$ from the action set. Then, the environment probabilistically transfers to the next state s' based on a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$. Correspondingly, based on the transition of (s, a, s'), the agent obtains a reward $r(s, a, s')$ determined by reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The agent's goal is to maximize its accumulated rewards $\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t, s_{t+1})$ during a period by learning an optimal policy.

4. Coalitional Markov Decision Process (CMDP)

Since coalitional Markov decision process (CMDP) can well reflect the issues of dynamic coalition formation, we first introduce CMDP in this section. Then, we state how to formulate the dynamic coalition formation problem in edge computing as a CMDP in Sect. 6.

4.1 The Model

Agent and coalition: We consider a set $\mathcal{N} = \{1, \ldots, n\}$ of agents. Each agent can form a coalition c_i with other agents and each agent can only join one coalition. Thus, a way of coalition formation is called a coalition structure denoted as cs, i.e., $cs = \{c_i | (\forall i \neq j, c_i \cap c_j = \emptyset) \wedge (\bigcup_{i=1}^{|cs|} c_i = \mathcal{N})\}$. Then, we use $\mathcal{P}^{\mathcal{N}} = \{cs_1, \ldots, cs_{|\mathcal{P}^{\mathcal{N}}|}\}$ to denote the set of all possible coalition structures.

State: We use state s to denote the status of environment and define a set $\mathcal{S} = \{s_1, s_2, \ldots, s_{|\mathcal{S}|}\}$ including all possible states.

Action: We denote $\mathcal{A}^{c_i}$ as the action set of coalition c_i where each coalition c_i chooses an action from $\mathcal{A}^{c_i}$. We denote joint action α^{cs} from all coalition actions under a given coalition structure cs, i.e., $\alpha^{cs} = a^{c_1} \times \cdots \times a^{c_{|cs|}}$. Correspondingly, the set of joint action α^{cs} is denoted by $\mathcal{A}^{cs} = \mathcal{A}^{c_1} \times \cdots \times \mathcal{A}^{c_{|cs|}}$.

Transition function: Based on current state s, the environment probabilistically transfers to the next state s', after taking joint action α^{cs}. This probability function, called the transition function, is denoted as $\mathcal{T} : \mathcal{S} \times \mathcal{A}^{cs} \times \mathcal{S} \to [0, 1]$, i.e., $\mathcal{T}(s, \alpha^{cs}, s')$=$Prob(s'|s, \alpha^{cs})$.

Reward function: $r : \mathcal{S} \times \mathcal{A}^{cs} \times \mathcal{S} \to \mathbb{R}$ is the reward function. In this case, what we want to maximize is the sum rewards from all agents rather than an individual agent's reward. The reward function is similar to the characteristic function in CSG which is used to evaluate the values of different coalitions. However, there are two differences: 1) CSG reflects a static coalition formation process, where it is solved once an optimal coalition structure is found. Thus, the characteristic function involves only coalitions. However, in CMDP, there is a dynamic state transition process during coalition formation; this means an optimal coalition structure must be identified in each state. Thus, the reward function should involve both coalition and state. 2) Moreover, in CSG there is no concept of action where the agents desire only to form coalitions. However, after forming a coalition from several agents, the coalition must choose an optimal action from an action set. Thus, the reward function is related to the state-coalition-action pair rather than a single coalition. Moreover, the characteristic function can be regarded as a special case of reward function when CMDP consists of only one state and one action.

Cost function: In the process of dynamic coalition formation, altering the coalition structure would bring corresponding cost, since coalition formation/dissolution in real-world scenarios usually corresponds to physical activities which incur some costs like energy which cannot be ignored. Thus, we define $cost : \mathcal{S} \times \mathcal{P}^{\mathcal{N}} \times \mathcal{S} \times \mathcal{P}^{\mathcal{N}} \to \mathbb{R}$ as the cost function to calculate the cost incurred by altering a coalition structure.

Policy: In CMDP, a coalition's decision-making for choosing an action proceeds in two phases: coalition formation for all agents and each coalition taking an action. Thus, it corresponds to two types of policies, separately. As the coalition formation, we define policy $\pi^{cs} : \mathcal{S} \times \mathcal{P}^{\mathcal{N}} \to [0, 1]$ to denote the probability of forming coalition structure cs under state s, i.e., $\pi^{cs}(s, cs) = Prob(cs|s)$. Then, each coalition c can choose an action from its action set. We collect the actions from all coalitions which is defined as α^{cs}. Correspondingly, we define a policy $\pi^{\alpha} : \mathcal{S} \times \mathcal{P}^{\mathcal{N}} \times \mathcal{A}^{cs} \to [0, 1]$ to denote the probability of choosing a joint action under state s and coalition structure cs.

Objective function: To sum up, we use the tuple $<\mathcal{N}, \mathcal{S}, \mathcal{A}^{c_i}, \mathcal{T}, r, cost>$ to denote a CMDP. Figure 2 shows how CMDP formulates a dynamic coalition formation. Under current state s, coalition structure cs is determined based on policy π^{cs}. Then, based on policy π^{α}, a joint action α^{cs} is determined. By implementing α^{cs}, the environment transfers to the next state based on transition function $\mathcal{T}$ and the above process is repeated. Finally, we can obtain a trajectory $h = [s[1], cs[1], \alpha^{cs}[1], s[2], \ldots, s[T], cs[T], \alpha^{cs}[T], s[T +$

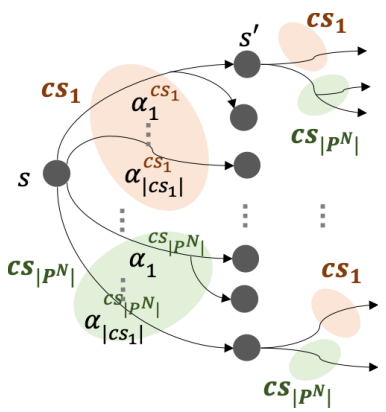


Fig. 2 The dynamic transition process in CMDP.

1]] based on that dynamic coalition formation process. Correspondingly, we can obtain the discounted sum *return* $R(h)$ of immediate rewards along trajectory h. It is defined as

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

where $\gamma \in [0, 1)$ is the discount factor to denote how important the rewards obtained in the future are. Unlike ordinary MDP, the cost of altering coalition structures also needs to be considered in CMDP which is a discounted sum $C(h)$ of immediate costs along trajectory h defined as:

$$C(h) = \sum_{t=1}^{T} \gamma^{t-1} cost(s[t], cs[t], s[t+1], cs[t+1]). \tag{2}$$

The goal of CMDP is to learn optimal policies of π^{cs} and π^{α} that can maximize $R(h)$ and the cost $C(h)$; $C(h)$ is always a negative number (the bigger the cost of changing cs is, the smaller the value of $cost$ is). Thus, we use weighted sum $J(h)$ as the objective function of CMDP which is defined as

$$J(h) = R(h) + \omega C(h), \tag{3}$$

where ω is the weight used to evaluate how important the cost is. Then, the goal changes to identify a couple of policies π^{cs} and π^{α} that can maximize the expectation of weighted sum $J(h)$ along trajectory h as follows.

$$\pi^{cs*}, \pi^{\alpha*} = \arg\max_{\pi^{cs}, \pi^{\alpha}} \mathbb{E}_{p^{\pi^{cs}, \pi^{\alpha}}(h)}[J(h)], \tag{4}$$

where $\mathbb{E}_{p^{\pi^{cs}, \pi^{\alpha}}(h)}$ denotes the expectation over trajectory h drawn from $p^{\pi^{cs}, \pi^{\alpha}}(h)$ which denotes the probability density of observing trajectory h under policies π^{cs} and π^{α}.

Compared with ordinary MDP, there are two major differences: 1) each coalition as an entity takes an action rather than an agent. This means that forming different coalitions corresponds to a different action set, which is more complicated than the fixed action set of ordinary MDP. 2) it does cover the concept of coalitions and altering coalition structures incurs a certain cost for pair (s, cs, s', cs') that cannot be evaluated by the reward function of naive MDP which is based on (s, α, s'). Thus, we define a cost function to evaluate the cost incurred by altering the coalition structure. Since CMDP is not a naive MDP, it requires us to propose a new algorithm that can handle its specific properties.

4.2 The Equivalent Policy

Since it is hard to do an optimization with considering both the policies π^{cs} and π^{α} meanwhile, we consider to construct an equivalent policy by using their relationship. Specifically, policy π^{α} usually makes a decision after that coalition structure cs is determined by π^{cs}, thus we can regard cs as a condition of π^{α}. We thus construct equivalent policy π^{eq} which is defined by

$$\pi^{eq} : \mathcal{S} \times \mathcal{A}^{all} \to [0, 1],$$

where $\mathcal{A}^{all} = \bigcup_{cs \in \mathcal{P}^{\mathcal{N}}} \mathcal{A}^{cs}$ includes joint actions from all possible coalition structures ($\alpha \in \mathcal{A}^{all}$). Thus, $\pi^{eq}(s, \alpha)$ is the probability of choosing joint action α from $\mathcal{A}^{all}$ under state s. Based on the relationship $\pi^{eq}(s, \alpha) = \pi^{cs}(s, cs)\pi^{\alpha}(s, cs, \alpha^{cs})$, both policies π^{cs} and π^{α} can be derived from $\pi^{eq}(s, \alpha)$ as follows.

$$\begin{aligned} \pi^{cs}(s, cs) &= \sum_{\alpha \in \mathcal{A}^{cs}} \pi^{eq}(s, \alpha), \\ \pi^{\alpha}(s, cs, \alpha^{cs}) &= \frac{[\sum_{\alpha' \in \mathcal{A}^{cs}} \pi^{eq}(s, \alpha')]\pi^{eq}(s, \alpha)}{\sum_{\alpha \in \mathcal{A}^{all}} \pi^{eq}(s, \alpha)}. \end{aligned} \tag{5}$$

Thus, the optimal policy π^{eq*} is defined by

$$\pi^{eq*} = \arg\max_{\pi^{eq}} \mathbb{E}_{p^{\pi^{eq}}(h)}[J(h)]. \tag{6}$$

4.3 The Value Functions

In the MDP, state-action value $Q(s, a)$ is often used to evaluate the quality attained by taking action a under state s. Then, an optimal policy can be obtained based on the value $Q(s, a)$ (Q-value). Besides the reward, the cost of altering coalition structures also exists in CMDP. Thus, we need to consider two Q-value functions to evaluate the reward and cost, separately. We define the Q-value function $Q^r : \mathcal{S} \times \mathcal{A}^{all} \to \mathbb{R}$ used to evaluate the rewards which is called *state-action value function for reward*, as

$$Q^r(s, \alpha) = \mathbb{E}_{p^{\pi^{eq}}(h)}[R(h) \mid s[1] = s, \alpha[1] = \alpha],$$

where $Q^r(s, \alpha)$ is the expected rewards obtained by following policy π^{eq} under the condition of "$s[1] = s$, $\alpha[1] = \alpha$". By recursion, we can rewrite $Q^r(s, \alpha)$ as follows.

$$\begin{aligned} Q^r(s, \alpha) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \alpha, s') \Big[& r(s, \alpha, s') \\ & + \gamma \sum_{\alpha' \in \mathcal{A}^{all}} \pi^{eq}(s', \alpha') Q^r(s', \alpha') \Big]. \end{aligned} \tag{7}$$

We define the Q-value function $Q^c : \mathcal{S} \times \mathcal{A}^{all} \to \mathbb{R}^-$ used to evaluate the cost of altering coalition structures, which is called as *state-action value function for cost*, as

$$Q^c(s, \alpha) = \mathbb{E}_{p^{\pi^{eq}}(h)}[C(h) | s[1] = s, \alpha[1] = \alpha],$$

where $Q^c(s, \alpha)$ is the expected cost obtained by following policy π^{eq} under the condition of "$s[1] = s$, $\alpha[1] = \alpha$". By recursion, we can rewrite $Q^c(s, \alpha)$ as follows.

$$\begin{aligned} Q^c(s, \alpha) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, \alpha, s') \Big[& \sum_{cs' \in \mathcal{P}^{\mathcal{N}}} \pi^{cs}(s, cs') \\ & cost(s, cs, s', cs') + \gamma \sum_{\alpha' \in \mathcal{A}^{all}} \pi^{eq}(s', \alpha') Q^c(s', \alpha') \Big]. \end{aligned} \tag{8}$$

Since the objective function $J(h)$ consists of both $R(h)$ and $C(h)$. We also consider a Q-value to evaluate $J(h)$ which is

defined as *weighted state-action value function*. By calculation, it can be written in terms of Q^r and Q^c as follows.

$$Q^{\omega}(s,\alpha) = Q^{r}(s,\alpha) + \omega Q^{c}(s,\alpha). \tag{9}$$

Based on Eqs. (7) and (8), $Q^{\omega}(s,\alpha)$ can be written as

$$\begin{aligned} Q^{\omega}(s,\alpha) = \sum_{s'\in\mathcal{S}} \mathcal{T}(s,\alpha,s') \Big[& r(s,\alpha,s') \\ & + \omega \sum_{cs'\in\mathcal{P}^{\mathcal{N}}} \pi^{cs}(s',cs') cost(s,cs,s',cs') \\ & + \gamma \sum_{\alpha'\in\mathcal{A}^{all}} \pi^{eq}(s',\alpha') Q^{\omega}(s',\alpha') \Big], \end{aligned} \tag{10}$$

where $Q^{\omega}(s,\alpha)$ includes both terms of reward $r(s,\alpha,s')$ and cost $cost(s,cs,s',cs')$. We can solve $Q^{\omega}(s,\alpha)$ to obtain an optimal policy that maximizes $J(h)$.

5. Algorithm

As illustrated in the above section, CMDP cannot be solved by the classical algorithms developed for solving MDP. We first introduce a basic solution called coalitional Q-learning (CQL) as first seen in [13]. Then, we introduce our proposed method DCQL in this paper which can handle a larger state space than CQL.

5.1 Coalitional Q-Learning (CQL)

Q-learning, is a classical algorithm that can guide agents in learning an optimal Q-value for decision-making [15]. We refer to the framework of Q-learning and consider the features of CMDP to propose an algorithm called coalitional Q-learning (CQL) to solve CMDP in [13]. First, based on Eq. (10), we further introduce an *optimal weighted state-action value function* $Q^{\omega*}$ by applying a maximization operator which is defined as follows

$$\begin{aligned} Q^{\omega*}(s,\alpha) = & \sum_{s'\in\mathcal{S}} \mathcal{T}(s,\alpha,s') \\ & [r(s,\alpha,s') + \max_{\alpha'} \{\omega cost(s,cs,s',cs^{-1}(\alpha')) + \gamma Q^{\omega*}(s',\alpha')\}], \end{aligned} \tag{11}$$

where $cs^{-1} : \mathcal{A}^{all} \to \mathcal{P}^{\mathcal{N}}$ is a function to obtain the cs given as α^{cs}. In Q-learning algorithm, there also exists a maximization operator to choose the maximized Q-value of next state to update the Q-value of current state. However, in Eq. (11), the maximization operator needs also to consider the corresponding cost of changing coalition structures rather than only Q-value, as shown in Fig. 3.

In the CQL algorithm, the value of $Q^{\omega*}$ are updated as follows.

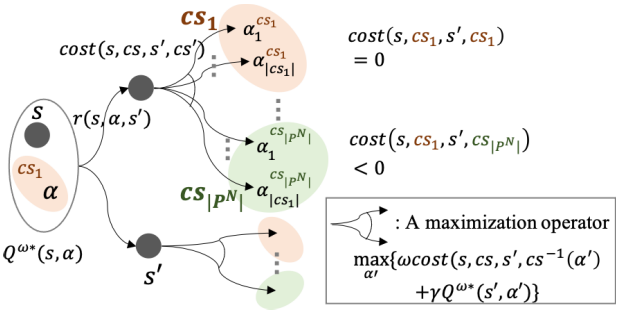


Fig. 3 The definition of optimal weighted state-action value function. Unlike Q-learning where the maximization operator only considers Q-value, the maximization operator of CQL needs also to consider the corresponding cost of changing coalition structures rather than only Q-value. As shown in the right part, the cost will be zero if the coalition structure does not change, the cost corresponds to a negative value otherwise.

$$\begin{aligned} Q^{\omega*}(s[t],\alpha[t]) \leftarrow & Q^{\omega*}(s[t],\alpha[t]) + \sigma[r[t+1] \\ & + \max_{\alpha'} \{\omega cost(s[t],cs[t],s[t+1],cs^{-1}(\alpha')) \\ & + \gamma Q^{\omega*}(s[t+1],\alpha')\} - Q^{\omega*}(s[t],\alpha[t])], \end{aligned} \tag{12}$$

where σ is the learning rate.

5.2 Deep Coalitional Q-Learning (DCQL)

Coalitional Q-learning is based on a tabular RL method which demands maintenance of a Q-table. Although tabular RL methods have good performance in many RL tasks, it becomes impractical for RL tasks with large state spaces since it is hard to maintain a huge table. For instance, in our motivating scenario, the state would increase with the number of tasks in the task set. To tackle this problem, we refer to deep RL (DRL) algorithms which combine deep neural networks with reinforcement learning. They have been verified as capable of dealing with large state spaces. Deep Q-network (DQN), is a classical DRL method proposed by Volodymyr et al. [16]; a neural network is used to approximate the value of $Q(s,a)$. Inspired by the idea of DQN, we improve CQL and propose a novel algorithm called DCQL for CMDP with large state space. Specifically, the optimal weighted state-action value $Q^{\omega*}$ is calculated by a deep neural network. It uses the tuple of $(s_j,\alpha_j,r_j,cost_j,s_{j+1})$ obtained at each step to train the network with the goal of minimizing the following loss equation.

$$L(\theta) = \mathbb{E}_{s,a\sim\pi}\Big[(y_j - Q^{\omega*}(s_j,\alpha_j;\theta))^2\Big], \tag{13}$$

where

$$y_j = \begin{cases} r_j + \omega cost_j & \text{if } s_{j+1} \text{ is terminal state,} \\ r_j + \gamma \max_{\alpha'}(\omega cost_j + Q^{\omega*}(s_{j+1},\alpha';\theta)) & \text{otherwise.} \end{cases}$$

where θ represents the parameters of the neural network.

Further, we use the motivating scenario in Sect. 2 to illustrate DCQL, see in Fig. 4. The statuses of all servers and task set are regarded as a state which allows DCQL to output each action's $Q^{\omega*}$ value. Based on the $Q^{\omega*}$ value,

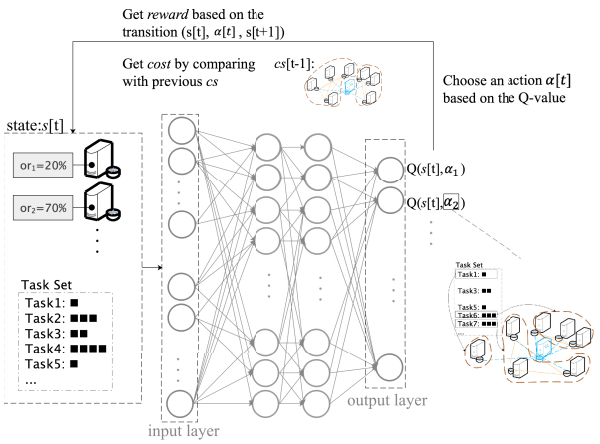


Fig. 4 The framework of deep coalitional Q-learning.

an action is chosen and the state transfers to the next state. Then, we can obtain the reward and cost to train this network through minimizing the loss function defined by Eq. (13).

Then, we analyze the computation time of DCQL algorithm with the CQL algorithm. Both algorithms consist of two phases: learning phase and executing phase.

- 1) Learning phase: DCQL algorithm trains a deep neural network by back propagation based on Eq. (13). CQL algorithm maintains a Q-table and updates the Q-values for each state-action pair based on Eq. (12).
- 2) Executing phase: DCQL algorithm inputs the current state and outputs the corresponding Q-values for each action by performing forward propagation. Then it chooses an action based on the Q-values. CQL algorithm searches for the Q-values corresponding to the current state in the Q-table and chooses an action based on the Q-values.

Although DCQL algorithm incurs a longer calculation time than the CQL algorithm in the learning phase, they take similar time to choose an action in the executing phase. In actual experiments, we care only about the executing phase once it has been trained well. Specifically, DCQL is given by Algorithm 1.

Algorithm 1 Deep Coalitional Q-learning (DCQL)

1: Initialize replay memory D
2: Initialize weighted action-value function $Q^{\omega*}$ with random weights θ
3: **for** episode m=1, M **do**
4: Generate initial state $s[1]$
5: **for** step t =1, T **do**
6: Randomly select an action $\alpha[t]$ from $\mathcal{A}^{all}(s[t])$ based on an ε-greedy policy
7: otherwise select
8: $\alpha[t] = \arg\max_{\alpha'} \{\omega cost(s[t-1], cs[t-1], s[t], cs^{-1}(\alpha')) + \gamma Q^{\omega*}(s[t], \alpha'; \theta)\}$
9: Execute action $\alpha[t]$ and observe reward $r[t] = r(s[t], \alpha[t], s[t+1])$ and cost $cost[t] = cost(s[t-1], cs[t-1], s[t], cs[t])$, and then transfer to the next state $s[t+1]$
10: Store transition $(s[t], \alpha[t], r[t], cost[t], s[t+1])$ in D and set $s[t] = s[t+1]$
11: Sample random mini-batch of transitions $(s_j, a_j, r_j, cost_j, s_{j+1})$ from D
12: Calculate y_j based on Eq. (13) and perform a gradient descent step to update θ
13: **end for**
14: **end for**

6. Evaluation

In this section, we run the experiments of dynamic coalition formation problem in edge computing to verify the effectiveness of our proposed DCQL algorithm.

6.1 Evaluation Setting

We refer to related papers [12], [17]–[20] to define the parameters of the following edge computing systems. And also, we make some simplifications which do not degrade the validity of our model. The specific setting is stated as follows.

Server: DCQL aims to solve a large state space problem in this paper. Since each server has 100 statuses of CPU occupancy rate, it corresponds to 10^6 joint statuses even if only 3 edge servers are considered. Thus, we take an edge computing system composed of 3 edge servers. We define the set of servers as $SV = \{sv_1, sv_2, sv_3\}$ where each server's parameter is denoted by the vector

$$sv_i = [or_i, c_i, k_i], \tag{14}$$

where or_i is the CPU occupancy rate, c_i is server i's coefficient about CPU speed (e.g., workload performed per CPU cycle), k_i is server i's server i's coefficient about energy cost (e.g., energy consumed per CPU cycle) which can be calculated by the measurement method stated in [18]. Although the parameter values can be set based on some physical servers like Raspberry Pi, it would not influence the effectiveness of our proposed method. Thus, we set each server as $c_i = 1$ and $k_i = 1$ for simplicity.

Task: As for setting the task set, we refer to [12], [17] to denote CPU cycles as the task workload. Although some other task parameters like RAM and Disk can also be considered, we ignore them for simplicity and focus on workload in this paper. That is because high workload has already well reflected the necessity of dynamic coalition formation and ignoring the other parameters does not degrade the validity of our model. We consider several task sets with different number of tasks and three types of tasks with workload: 1000, 2000 and 3000 million CPU cycles, separately. We assume 1000 million CPU cycles as a unit workload and it can be allocated to only one server at a time. For instance, a task with 2000 million CPU cycles requires two servers. Then, the server can execute the tasks in parallel by applying a time-slice mechanism. Without loss of generality, the number of each type task is randomly generated in each task set. Each episode is concluded when all tasks in the task set have been completed.

Then, we formulate the problem as a CMDP and the corresponding elements of CMDP in this problem are given

as follows.

State: The dynamic of edge computing is represented as state transition. Since the dynamic is driven by the factors of server status and task set status, we construct the state $s[t]$ at step t by including $or_i[t]$ from all servers at step t and the current task set status at step t, i.e., $s[t] = [[or^1[t], or^2[t], or^3[t]], [n^{ty_1}[t], n^{ty_2}[t], n^{ty_3}[t]]]$, where ty_j is j type tasks whose workload includes j units and n^{ty_j} represents the number of type ty_j tasks left in the task set.

Action: In CMDP, at each step the entity choosing the action is the coalition, thus we denote $\mathcal{A}^c$ as the action set of each coalition c. In this case, $\mathcal{A}^c$ is the set of tasks that edge coalition c is capable to execute which also includes the action to choose no tasks, i.e., $a^c = 0$. Thus, the coalition action set is defined by

$$\mathcal{A}^c = \{ty_j \mid j = |c|\} \cup \{0\}.$$

Reward function: The goal is to perform all the tasks as soon as possible while minimizing the cost of energy consumption and the cost incurred by coalition structure alteration. Thus, when all tasks have been performed, a large positive reward is given. Specifically, the reward function is defined as follows.

$$r(s, \alpha, s') = \begin{cases} -energy(s, s') + 100, & \text{if } s' \text{ is absorb state,} \\ -energy(s, s'), & \text{otherwise.} \end{cases}$$

where 100 is the reward of absorb state in which all tasks have been accomplished, $energy(s, s')$ is the server energy consumed in performing the tasks; we refer [19] to define it as

$$energy(s, s') = \sum_i k_i * or_i. \tag{15}$$

Then we refer [20] to approximate server i's occupancy rate or_i by the task number ta_n allocated on server i which is given by $or_i = \frac{ta_n}{ta_{max}}$ where ta_{max} is the maximum task number can be allocated on it (we assume $ta_{max} = 100$ in this paper).

Cost function: As for calculating the cost of changing coalition structure, we assume it is determined by the number of changes in agent relationships. Specifically, we use an undirected graph to describe a coalition, thus a coalition structure can be represented as several undirected graphs. In an undirected graph, each agent is represented as a node and any two agents can share one edge. Then, altering the coalition structure means adding/deleting edges of the undirected graphs. Thus, the cost of altering a coalition structure can be calculated by the number of edge changes. First, we define set N_G^{cs} to describe each agent's neighborhood (the members in the same coalition) which is defined by

$$N_G^{cs} = \{N_G^{cs}(i) = \{k | k, i \in c_j \in cs \wedge k \neq i\} \mid \forall i \in \mathcal{N}\}.$$

Then we consider set $N_G^{\mathcal{P}^\mathcal{N}}$ which includes all possible N_G^{cs}, i.e. $N_G^{\mathcal{P}^\mathcal{N}} = \{N_G^{cs_1}, N_G^{cs_2}, \ldots, N_G^{cs_{|\mathcal{P}^\mathcal{N}|}}\}$. Furthermore, we introduce function $D : N_G^{\mathcal{P}^\mathcal{N}} \times N_G^{\mathcal{P}^\mathcal{N}} \to \mathbb{R}$ to quantify the difference between two coalition structures, as defined by

$$D(N_G^{cs}, N_G^{cs'}) = \frac{1}{2} \sum_i \left| N_G^{cs}(i) \triangle N_G^{cs'}(i) \right|,$$

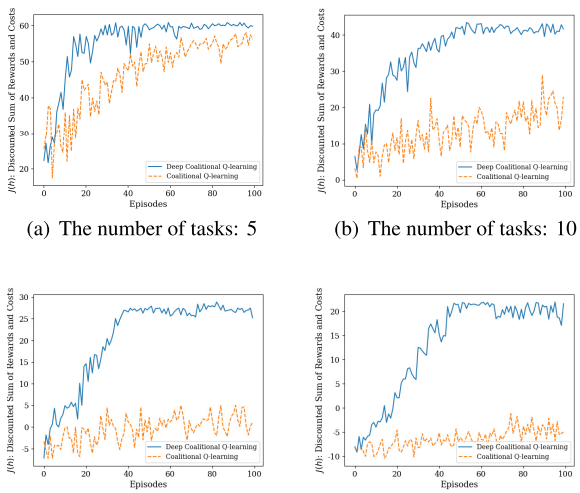
where $\triangle$ is symmetric difference calculation used to evaluate the number of edges altered. Thus, the cost of altering coalition structures can be represented by the value of $D(N_G^{cs}, N_G^{cs'})$. We define cost function $cost$ as the inverse number of $D(N_G^{cs}, N_G^{cs'})$ as follows

$$cost(s, cs, s', cs') = -D(N_G^{cs}, N_G^{cs'}).$$

6.2 Evaluation Results

We check the performances of our DCQL algorithm by comparing it with the CQL algorithm. We set the same hyperparameters for all algorithms with $\alpha = 0.01$ and $\gamma = 0.9$. We consider task sets consisting of 5 tasks, 10 tasks, 15 tasks and 20 tasks. For each task set setting, we ran the following simulations for 100 episodes and each episode's maximum step is 100 (the episode is terminated even though some tasks in the task set are not executed); the results are shown in Fig. 5. As shown in Fig. 5 (a), since the task set has 5 tasks which corresponds to a small state space, CQL can learn effectively and matches the performance of DCQL. However, CQL cannot learn effectively when the task number increases which corresponds to a large state space, as shown in Fig. 5 (b)–(d). DCQL has better performances than CQL since it can cope with large state spaces well.

In order to compare these two algorithms visually, we focus on the final learning results (the average of the last 10 episodes) for all four task sets. With the increase of task number, it would naturally cost more steps to get the absorb state (all tasks have been executed completely). Correspondingly, the optimal value of $J(h)$ would decrease since



(a) The number of tasks: 5 (b) The number of tasks: 10

(c) The number of tasks: 15 (d) The number of tasks: 20

Fig. 5 Comparing the performances of deep coalitional Q-learning with coalitional Q-learning.

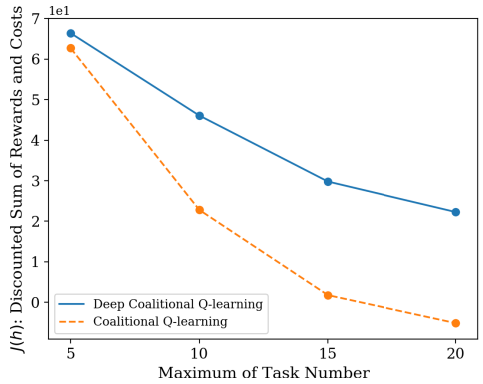


Fig. 6 Comparing the final learning results of deep coalitional Q-learning with coalitional Q-learning.

taking more steps to get the absorb state corresponds to a smaller discount value of γ^{t-1} based on Eqs. (1)–(3). Thus, the $J(h)$ value of DCQL naturally decreases with the increase of task number, even though it can learn an optimal solution in each task set. Then, DCQL's superiority over CQL majorly depends on the CQL's performance. As shown in Fig. 6, the percentage improvements attained by our method over CQL are 9.1%, 101.3%, 25.1 times, 26.1 times for the task sets with task numbers of 5, 10, 15 and 20. When the task number increases from 5 to 15, CQL's performance decreases quickly since a large state space problem is incurred. Thus, DCQL's superiority over CQL increases quickly. However, there exists a lower-bound of $J(h)$ value which corresponds to a worst case: the algorithm almost learns nothing and so the tasks are rarely executed. We terminate the episode once the maximum step is reached, thus $J(h)$ would not go to negative infinity with the increase of task number even though in the worst case. In task sets with 15 and 20 tasks, we can see CQL has the similar performances that can almost learn nothing (confirmed by Fig. 5 (c)(d)), which are close to the lower-bound of $J(h)$ value. Thus, DCQL's superiority over CQL increases slowly when the task number increases from 15 to 20.

6.3 Discussion

Scale of edge servers: To summarize the above experiments, our proposed algorithm offers an effective solution to dynamic coalition formation problem in edge computing. In this paper, we focus on solving the problem of large state spaces rather than large action spaces. Thus, we consider the case of a fixed number of edge servers that corresponds to a large state space and a small action space. Specially, the action space scale mainly depends on the number, B_n, of coalition structures which is called Bell number. B_n exponentially increases as the number of servers which is calculated by $B_{(n+1)} = \sum_{k=0}^{n} \binom{n}{k} B_k$. Currently, DCQL cannot cope with large action spaces, since DCQL is based on a DQN algorithm that is effective only in small action spaces. However, the scale of edge servers is also an important factor for real-world deployment. Our future work is to improve DCQL so that it can handle large action spaces.

Real-world deployment: Although this paper confirms effectiveness of our proposal by only running simulations, it is also practical to permit real deployment. Specifically, CPU statuses can be monitored by the methods in [21] and the task workload can be obtained using the method in [22]. Moreover, the parameter values can be set based on the physical servers like Raspberry Pi. Actually, there could be some gaps from the model to the actual physical servers such as communication delay and the computation resource limitations which cannot be ignored; thus, we intend to tackle these gaps in our future work.

7. Related Work

The cooperation problem in edge computing has been examined in many studies. Cao et al. [5] consider mobile edge computing systems with the aim of improving the energy efficiency for latency-constrained computation. They set a server as helper and allow other edge servers to offload computation tasks to the helper which cooperatively computes these tasks. Zhang et al. [7] consider a joint computation and communication user cooperation problem in edge computing, where one user can share its own computation resources with the others to improve overall user performance. Yuan et al. [8] consider a cooperative edge computing platform among different stakeholders and propose a blockchain system based method to solve the issues of trust and incentive among edge servers. Although the above studies consider a cooperation setting in edge computing, they are based on a static cooperation way which is unlikely to well support the dynamic environments expected in edge computing.

As for the dynamic coalition formation problem, our previous work [4] considers this problem, but it uses only a naive MDP that ignores some coalition features like the cost of changing the coalition structure. Then, in [13], we proposed CMDP to overcome this deficiency as well as a basic solution of CQL. To handle the large state spaces in edge computing, we propose a DCQL algorithm in this paper. Although, there are some related studies that combine MDP with coalition formation [23], [24], they treat it as a repeated CSG with multiple-stages rather than a dynamic Markov transition process. Thus, these works do not consider the dynamic coalition formation problem essentially.

8. Conclusion

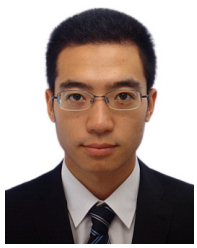
This paper studied a cooperation problem in edge computing. We proposed a new cooperation approach by guiding the edge servers to dynamically form coalitions to perform tasks cooperatively. We addressed its issues and formulated it by a theoretic model called CMDP. Since, the basic solution of CMDP, CQL, cannot handle the large state spaces, we proposed a novel algorithm called DCQL to solve it and verified its effectiveness in different experiment settings.

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] 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.
[2] 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.
[3] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for energy-efficient mobile edge computing." IEEE Internet Things J., vol.6, no.3, pp.4188–4200, 2019.
[4] S. Ding and D. Lin, "Dynamic task allocation for cost-efficient edge cloud computing," IEEE International Conference on Services Computing, pp.218–225, 2020.
[5] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for mobile edge computing," 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, pp.1–6, 2018.
[6] Y. Li, G. Xu, J. Ge, X. Fu, and P. Liu, "Communication and computation cooperation in wireless network for mobile edge computing," IEEE Access, vol.7, pp.106260–106274, 2019.
[7] Y. Yu, J. Zhang, and K.B. Letaief, "Joint subcarrier and CPU time allocation for mobile edge computing," IEEE Global Communications Conference, pp.1–6, 2016.
[8] L. Yuan, Q. He, S. Tan, B. Li, J. Yu, F. Chen, H. Jin, and Y. Yang, "CoopEdge: A decentralized blockchain-based platform for cooperative edge computing," Proc. Web Conference, pp.2245–2257, 2021.
[9] G. Greco and A. Guzzo, "Constrained coalition formation on valuation structures: Formal framework, applications, and islands of tractability," International Joint Conference on Artificial Intelligence, pp.5612–5616, 2018.
[10] T. Rahwan and N. Jennings, "Coalition structure generation: Dynamic programming meets anytime optimisation," Proc. Twenty-Third AAAI Conference on Artificial Intelligence, pp.156–161, 2008.
[11] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohmé, "Coalition structure generation with worst case guarantees," Artificial Intelligence, vol.111, no.1-2, pp.209–238, 1999.
[12] 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.
[13] S. Ding and D. Lin, "A coalitional Markov decision process model for dynamic coalition formation among agents," IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, pp.308–315, 2020.
[14] M.L. Littman, "Markov games as a framework for multi-agent reinforcement learning," Machine Learning Proceedings, pp.157–163, 1994.
[15] C.J.C.H. Watkins and P. Dayan. "Q-learning," Machine Learning, vol.8, no.3-4, pp.279–292, 1992.
[16] 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.
[17] X. Liu, J. Yu, J. Wang, and Y. Gao, "Resource allocation with edge computing in IoT networks via machine learning," IEEE Internet Things J., vol.7, no.4, pp.3415–3426, 2020.
[18] 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.
[19] F. Armenta-Cano, A. Tchernykh, J.M. Cortés-Mendoza, R. Yahyapour, A.Y. Drozdov, P. Bouvry, D. Kliazovich, and A. Avetisyan, "Heterogeneous job consolidation for power aware scheduling with quality of service," Russian Supercomputing Days 2015, pp.687–697, 2015.
[20] H. Aydin, R. Melhem, D. Mossé, and P. Mejía-Alvarez, "Power-aware scheduling for periodic real-time tasks," IEEE Trans. Comput., vol.53, no.5, pp.584–600, 2004.
[21] C. Ludmila and R. Gardner. "Measuring CPU overhead for I/O processing in the Xen virtual machine monitor," USENIX Annual Technical Conference, General Track, vol.50, pp.387–390, 2005.
[22] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," ACM SIGMETRICS Performance Evaluation Review, vol.40, no.4, pp.23–32, 2013.
[23] G. Chalkiadakis and C. Boutilier, "Bayesian reinforcement learning for coalition formation under uncertainty," Proc. Third International Joint Conference on Autonomous Agents and Multiagent Systems, vol.3, pp.1090–1097, 2004.
[24] G. Chalkiadakis, E. Markakis, and C. Boutilier, "Coalition formation under uncertainty: Bargaining equilibria and the Bayesian core stability concept," Proc. 6th International Joint Conference on Autonomous Agents and Multiagent Systems, pp.1–8, 2007.



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 and AAMAS. His papers appear in major international conferences like IEEE SCC, IEEE ICWS, ICSOC, and journals including IEEE TSMC, ACM TAAS, and ACM TALLIP.