Dynamic Task Allocation for Cost-Efficient Edge Cloud Computing

Shiyao Ding
Department of Social Informatics
Kyoto University
Kyoto, Japan
Email: ding@ai.soc.i.kyoto-u.ac.jp

Donghui Lin
Department of Social Informatics
Kyoto University
Kyoto, Japan
Email: lindh@i.kyoto-u.ac.jp

***Abstract*—Edge cloud computing systems are widely used to supply various computation services in Internet of Things (IoT). An essential problem is how to efficiently allocate task requests to various edge and cloud servers given task requirements (e.g., response time and required memory space), in order to minimize various costs generated in edge cloud computing. Existing studies on task allocation usually consider the viewpoint of provider cost such as offloading cost, uploading cost and deployment cost. However, the viewpoint of user cost (e.g., server fee) is rarely considered which is becoming an important issue in the deployment of edge cloud computing systems, especially for cost sensitive users like venture companies. In this paper, we study a dynamic task allocation problem in edge cloud computing where both servers' status and arriving tasks would change along with time; the goal is to search the task allocation policy that can minimize user cost. Specifically, we consider a parallel processing case where a task's workload can be infinitely divided among the various servers; this causes a huge solution space and makes the problem hard to solve. Thus, we consider an approximate method from the perspective of server coalitions rather than a single server, and propose a dynamic coalition formation algorithm called coalitional R-learning (CR-learning) to guide several edge servers in forming a coalition dynamically. Simulations verify that our algorithm can significantly reduce user cost comparing with some other existing algorithms while shrinking the solution space.**

***Keywords*-IoT; edge cloud computing; task allocation; Markov decision process; reinforcement learning**

I. Introduction

With the rapid development of Internet of Things (IoT), billions or even trillions of IoT devices will be potentially networked [1]. Since the computation capabilities of IoT devices are always limited, they must rely on external computing resources [2]. Cloud computing usually offers high reliability given its abundant computation resources, and provides scalable and efficient computing services. Cloud access usually, however, incurs huge bandwidth consumption and long delays, and so may not satisfy IoT applications that need very low latency [3]. Edge computing supplements cloud computing by providing cloud-like services closer to the IoT devices. While offering several advantages such as controllable latency and low energy consumption [4], edge servers are not as rich in capacities such as bandwidth, processing speed and memory size as cloud servers. Therefore, edge cloud computing has been proposed as a solution and is attracting a lot of interest as it balances the advantages of edge and cloud computing [5].

In edge cloud computing, an important problem is how to allocate tasks to the various servers to minimize various costs such as offloading cost [4], data transfer cost [6] and deployment cost [7] while satisfying the task requirements. However, the works mentioned rarely consider the viewpoint of user cost such as cloud service fee and edge server electricity fee. User cost has become an issue of more importance, since the cloud services offered by some cloud vendors such as Amazon, Aliyun and Azure, usually come with high user costs. According to a report from TechRepublic in 2019, nearly 60% of organizations overspend their cloud service budgets. This creates tension for users as their cost savings were seen as the core reason for cloud adaption across business and IT sectors [8].

In this paper, we study a dynamic task allocation problem in edge cloud computing. Specifically, we consider a certain mixture of heterogeneous edge and cloud servers that have different attribute values such as computation rate, storage resource and unit fee per second. In each step, the centralized coordinator, which has a global view of the current status of all servers, would confront a task request queue with an uncertain number of tasks, and allocate the tasks to those servers, given that each task's workload can be distributed among several servers. Within a certain period of several steps, the goal is to learn an optimal task allocation policy that can minimize the summation of costs from all servers during the period while satisfying the constraints of both tasks and servers. However, this triggers two major issues: 1) how to handle the dynamic characteristics of the servers and the arrival of task requests in each step; 2) how to cope with a huge solution space since each task's workload can be infinitely divided among the various servers.

To resolve the first issue, we formulate the dynamic task allocation problem as a Markov decision process (MDP) where all servers' current status and arriving task requests are represented as a state, so those dynamics can be treated as state transitions in MDP. As for the second issue, we introduce an approximate method to generate the solution space from the perspective of server coalitions; we propose a dynamic coalition formation algorithm to guide edge

2474-2473/20/$31.00 ©2020 IEEE
DOI 10.1109/SCC49832.2020.00036

server cooperation in performing tasks. Compared to existing studies [4]-[13], the main contributions of this paper are summarized as follows:

- We formulate a dynamic task allocation problem in edge cloud computing as a MDP which can handle the dynamic features of the problem well;
- We propose a dynamic coalition formation algorithm to guide edge server cooperation for task completion so as to minimize user cost while reducing the solution space;
- We run experiments that show our proposed algorithm outperforms some other existing algorithms in reducing user cost.

II. Related Work

Task allocation to minimize various costs in edge cloud computing has been tackled often in recent years. Tao et al. [4] formulated an energy cost minimization problem with the constraints of resource capacity; the cost factor they focused on is the energy consumed by offloading tasks from mobile devices to edge servers. Chen et al. [9] studied mobile edge cloud computing and adopted a game theoretic approach for minimizing offloading cost in a distributed manner. Gu et al. [7] considered a mobile edge cloud computing case and attempted to minimize the joint energy cost including uploading cost, deployment cost and inter-base station communication cost. They formulated the problem as a mixed integer nonlinear program and then linearized it into mixed integer linear programming form. Zhang et al. [10] focused on minimizing the energy cost consumed by the system itself and proposed cost-efficient scheduling for delay-sensitive tasks in edge computing. However, the various energy costs considered in these works are all from the viewpoint of provider rather than the viewpoint of user. And also these studies did not consider the dynamic features of edge cloud computing.

There are also some studies that consider dynamic features in edge cloud computing. Li and Huang [6] formulated a dynamic process for task allocation using MDP approach to balance the tradeoff between energy costs and Quality of Service (QoS) requirements; the energy costs they focused on are mainly associated with sensor hub and data transfer. Guo et al. [11] considered a dynamic continuous time process in edge cloud computing and proposed a task allocation policy for achieving optimal power-delay tradeoff in the system. Although these works considered dynamic processes in edge cloud computing, they are not associated with user cost.

Moreover, there are some studies that consider the cost from the viewpoint of user and also emphasize the dynamic features in their models. Bittencourt et al. [12] introduced a scheduling problem in hybrid clouds to minimize cloud service fee and make briefly surveyed some of the scheduling algorithms used in such systems. Yu et al. [13] utilized MDP to formulate workflow scheduling to minimize execution cost. Although these works took reducing user cost as an optimal goal, what they focused on is how to schedule the sequence of performing the tasks where some tasks can be performed only after some other tasks have been formed; these studies did not consider the strategy of allocating the tasks on servers. However, this paper considers a dynamic task allocation problem in edge cloud computing and takes user cost as the optimization goal while satisfying the task requirements.

III. Preliminaries

A. Edge Cloud Computing System

We refer to the edge computing architecture in [10][14] and extend it to the edge cloud computing system shown in Figure 1; it consists of a centralized coordinator and several heterogeneous edge and cloud servers. In each step, a task queue consisting of several tasks accesses the coordinator, then the coordinator must decide how to allocate those tasks to different servers. We assume each task can be distributed to several servers to be performed in parallel. Since each server will charge a different user cost for performing the same task, task allocation significantly impacts overall costs. In this system, we set server electricity fee as edge server cost and cloud service fee as cloud server cost, so both of them are user cost components. Thus, the goal is to find an optimal policy for allocating tasks to the servers in each step, which can minimize user cost.

Moreover, if a server is regarded as an agent which can observe state (e.g., server status and arriving task queue), the dynamic task allocation process can be formulated as MDP, a classic model to cope with discrete time decision processes, which is described in the next part. Moreover, the tasks whose requirements cannot be satisfied by edge servers with limited capacities have to be offloaded to cloud servers, which causes a higher fee than that of offloading them to edge servers. Thus, we consider to propose a dynamic coalition model based on coalition structure generation (CSG) to make edge servers form several coalitions to cooperate to perform those tasks where the coalitions would dynamically change in each step to fit with each state.

B. Markov Decision Process

Markov decision process (MDP) is a traditional mathematical framework for modeling decision making in a dynamic discrete time process. In MDP, an agent observes current state s of the environment and chooses action a based on its policy $\pi(s, a)$. The environment then probabilistically transfers to the next state s' and the above process is repeated. In each transition (s, a, s'), the agent obtains an immediate reward/cost $r(s, a, s')$ and the agent's goal is to learn an optimal policy that can maximize/minimize its accumulated rewards/costs $\frac{1}{T}\sum_{t=1}^{T} r(s_t, a_t, s_{t+1})$ during a period, where T is the length of the period.

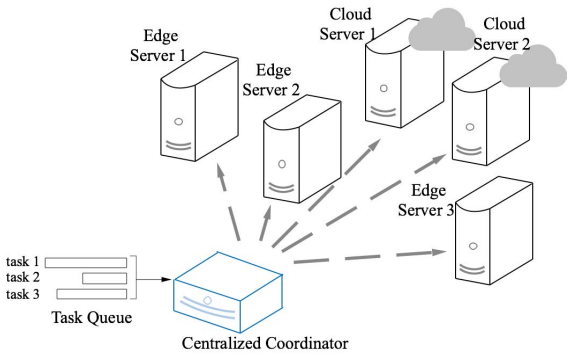

Figure 1. Edge cloud computing architecture

Specifically, MDP can be denoted as the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, r >$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function where $\mathcal{T}(s, a, s') = Prob(s'|s, a)$ is the probability of transitioning to the next state $s' \in \mathcal{S}$, after taking action $a \in \mathcal{A}$ given the current state $s \in \mathcal{S}$, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function.

MDP problems can be well solved via reinforcement learning (RL) which guides the agent in learning the optimal policies by trial-and-error; many RL algorithms have been proposed [16].

C. Coalition Structure Generation

If there are multiple resource-limited agents, the agents should form coalitions to aggregate their resources, which yields more rewards for them than that without forming coalitions. Coalition structure generation (CSG) is an effective model for studying such agent cooperation problems. It focuses on partitioning a set of agents into mutually disjoint coalitions so as to maximize the total reward [17]. In CSG, each coalition corresponds to a value based on a characteristic function and the goal of CSG is to find an optimal coalition structure that can maximize the sum of rewards from all coalitions.

We assume a set of agents $\mathcal{N} = \{1, ..., N\}$ where several agents can form a coalition c which is a subset of $\mathcal{N}$, i.e., $c \subseteq \mathcal{N}$; thus the set of all possible coalitions c is a power set of $\mathcal{N}$, which is denoted as $2^{\mathcal{N}}$. Since it is assumed that an agent can be allocated to just one coalition, $\mathcal{N}$ can be divided to several disjoint subsets, where each division is called as a *coalition structure*, denoted by cs, i.e., $cs = \{c_1, ..., c_{|cs|} \mid \forall i \neq j,\ c_i \cap c_j = \varnothing \wedge \bigcup_{i=1}^{|cs|} c_i = \mathcal{N}\}$. In CSG, each coalition c corresponds to value $v_{cha}(c)$ based on characteristic function $v_{cha} : 2^{\mathcal{N}} \rightarrow \mathbb{R}$ and the goal is to search the coalition structure, cs^*, that maximizes the sum of the values of all coalitions, i.e., $cs^* = \text{argmax}_{cs \in \mathcal{P}^{\mathcal{N}}} \sum_{c_i \in cs} v_{cha}(c_i)$.

Table I
NOTATIONS AND DEFINITIONS

$\mathcal{SV}$	The set of all servers
sv_i	The i-th server
cr_i	The i-th server's computation rate
sr_i	The i-th server's available storage resource
sr_{it}	The i-th server's available storage resource at step t
sr_i^{max}	The i-th server's maximum of storage resource
dt_i	The i-th server's delay time
uc_i	The i-th server's unit cost in ON status
$\mathbf{ta}_t$	The queue of arrived tasks at step t
ta_j	The j-th task
ta_{tj}	The j-th task at step t
w_j	The j-th task's workload
rt_j	The j-th task's required response time
rsr_j	The j-th task's required storage resource
x_j^i	The binary variable to denote whether the j-th task is allocated to the i-th server
$\mathbf{x^j}$	The vector to denote the j-th task's allocation on all servers
$\mathbf{x}$	The action: a task allocation strategy
s_t	The state at step t
$\mathbf{sr}_t$	The current available storage resource of all servers at step t
$\mathcal{A}(s)$	The set of all possible actions given the state s
$\pi(s, \mathbf{x})$	The probability of taking action $\mathbf{x}$ given state s
$\mathcal{T}(s, \mathbf{x}, s')$	The probability of transition to state s' given current state s and action $\mathbf{x}$
$cost_i(s, \mathbf{x}, s')$	The server i's immediate cost in transition $(s, \mathbf{x}, s')$
$cost_t^{\mathcal{SV}}$	The sum of all servers' immediate costs at step t
$Cost(h)$	The sum of all servers' immediate costs along the period h

IV. MODEL

In this section, we formulate a dynamic task allocation model in an edge cloud computing system as shown in Figure 1. Correspondingly, all the notations and definitions are summarized in Table I.

Server: We consider an edge cloud computing system consisting of n servers $\mathcal{SV} = \mathcal{SV}^e \cup \mathcal{SV}^c = \{sv_1, sv_2, ..., sv_n\}$ where $\mathcal{SV}^e$ is the set of edge servers and $\mathcal{SV}^c$ is the set of cloud servers. Each server $sv_i \in \mathcal{SV}$ is denoted by vector $sv_i = [cr_i, sr_i, dt_i, uc_i]$ where cr_i is the *computation rate* of server i (assumed to be a constant), sr_i is the available *storage resource* of server i that is a variable altered by pushing/popping tasks where we denote sr_i^{max} as the maximum of available storage resource (i.e., no task is allocated on server i), dt_i is a constant to represent the *delay time* when a task is offloaded to server i, uc_i is server i's *unit cost* when the server is in ON status (we assume the server's cost will be zero, if it is in OFF status).

Although, edge and cloud servers can be represented by the same tuple, their parameter values are substantially different; cloud servers have significantly greater computation resources than edge servers, while edge servers usually have lower delay time than cloud servers.

Task: At each step, a task queue consisting of some tasks arrives, which is denoted as $\mathbf{ta}$=$\{ta_1, ta_2, ..., ta_{|\mathbf{ta}|}\}$ with ta_j=$[w_j, rt_j, rsr_j]$ where w_j is task j's *workload* required

to be performed, rt_j is the longest acceptable *response time* of task j, and rsr_j is task j's *required storage resource* if it is allocated on a server.

Dynamic Task Allocation: The dynamic characteristics of the model are from two parts: 1) one is that the server's available storage resources are altered by pushing/popping tasks; 2) the other is that, at each moment, arriving task requests are uncertain. Thus, the task allocation strategy should suit the dynamic characteristics, which can be regarded as a discrete time dynamic decision problem. MDP, a classical model formulating discrete time dynamic decision process, is used to formulate the problem in this paper. First, we introduce

$$s_t = \big[\mathbf{sr}_t, \mathbf{ta}_t\big]$$

to denote *state* in step t: the first part $\mathbf{sr}_t = \big[sr_{1t}, ..., sr_{|\mathbf{sr}_t|t}\big]$ includes the current available storage resource of all servers at step t; the second part $\mathbf{ta}_t = \big[ta_{t1}, ..., ta_{t|\mathbf{ta}_t|}\big]$ is the server request queue arriving at step t;

As for state s_t, task allocation strategy $\mathbf{x}_t$ that allocates the tasks of $\mathbf{ta}_t$ to the servers in $\mathcal{SV}$ would be taken. Specifically, $\mathbf{x}_t$ can be denoted by a matrix as follows,

$$\mathbf{x}_t = \big[\mathbf{x^1}, ..., \mathbf{x^j}, ..., \mathbf{x}^{|\mathbf{ta_t}|}\big]^\top,$$

where $\mathbf{x^j} = [x_1^j, ..., x_i^j, ..., x_{|\mathcal{SV}|}^j]$ denotes task $ta_{tj} \in \mathbf{ta}_t$ workload distribution among all servers $sv \in \mathcal{SV}$ and its element $x_i^j \in [0, 1]$ denotes the allocated partitioning on server sv_i. For instance, x_i^j=0.8 means that 80% of task ta_{tj}'s workload is allocated to server i; it satisfies

$$\sum_{i=1}^{|\mathcal{SV}|} x_i^j = 1. \tag{1}$$

Then, task allocation strategy $\mathbf{x}$ is regarded as an action in MDP and all possible tasks allocation strategies in state s_t can be represented by an action set as follows,

$$\mathcal{A}(s_t) = \{\mathbf{x}_1, ..., \mathbf{x}_{|\mathcal{A}(s_t)|}\}.$$

Then, we consider a policy function $\pi : \mathcal{S} \times \mathcal{A}(s) \to [0, 1]$ to guide how to choose action $\mathbf{x}_t$ from $\mathcal{A}(s_t)$ given state $s_t \in \mathcal{S}$ ($\mathcal{S}$ is the set including all possible states); it is defined by $\pi(s, \mathbf{x}) = Prob(\mathbf{x}|s)$ which is the conditional probability of choosing action $\mathbf{x}$ given the condition of state s.

The environment transfers to the next state s_{t+1} upon completion of action $\mathbf{x}_t$ based on transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}(s) \times \mathcal{S} \to [0, 1]$; this means the environment probabilistically transfers to the next state s' depending on current state s and action $\mathbf{x}$, i.e., $\mathcal{T}(s, \mathbf{x}, s')$=$Prob(s'|s, \mathbf{x})$. As for the transition $(s_t, \mathbf{x}_t, s_{t+1})$, we consider a corresponding summation of costs from all servers: $cost^{\mathcal{SV}}(s_t, \mathbf{x}_t, s_{t+1})$; we denote it as $cost_t^{\mathcal{SV}}$ hereafter; it can be solved by

$$cost_t^{\mathcal{SV}} = \sum_{i=1}^{|\mathcal{SV}|} cost_i(s_t, \mathbf{x}_t, s_{t+1}),$$

where

$$cost_i(s_t, \mathbf{x}_t, s_{t+1}) = \begin{cases} uc_i & \text{if server } i\text{'s status in } s_{t+1} \text{ is ON,} \\ 0 & \text{otherwise.} \end{cases}$$

Let us consider a certain trajectory h with T steps and assume the timeslot between any two adjacent steps is one second; h can be represented as follows,

$$h = [s_1, \mathbf{x}_1, s_2, \mathbf{x}_2, ..., s_T, \mathbf{x}_T, s_{T+1}].$$

Then, h with T steps corresponds to a period (e.g., 10 seconds, 1minute) and the sum of the immediate costs in the period is $\sum_{t=1}^{T} cost_t^{\mathcal{SV}}$ which is what we want to minimize. Since, length T is given, then minimizing that cost summation is equivalent to minimizing its average which is defined as:

$$Cost(h) = \frac{1}{T}\sum_{t=1}^{T} cost_t^{\mathcal{SV}} = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{|\mathcal{SV}|} cost_i(s_t, \mathbf{x}_t, s_{t+1}). \tag{2}$$

In order to minimize $Cost(h)$, we need to search for an optimal policy π^* that corresponds to the minimum expected value of $Cost(h)$, i.e.,

$$\pi^* = \arg\min_{\pi} \mathbb{E}_{p^\pi(h)}\big[Cost(h)\big], \tag{3}$$

where $\mathbb{E}_{p^\pi(h)}$ denotes the expected value over trajectory h drawn from $p^\pi(h)$, and $p^\pi(h)$ denotes the probability density of observing trajectory h under policies π:

$$p^\pi(h) = p(s_1)\prod_{t=1}^{T} \pi(s_t, \mathbf{x}_t)\mathcal{T}(s_t, \mathbf{x}_t, s_{t+1}). \tag{4}$$

where $p(s_1)$ is the probability of initial state.

Constraints: We have denoted task allocation $\mathbf{x} = \big[\mathbf{x^1}, ..., \mathbf{x^j}, ..., \mathbf{x}^{|\mathbf{ta}|}\big]^\top$ as an action in the above part. However, action $\mathbf{x}$ must satisfy both the task and server requirements. Specifically, task ta_j must be accomplished within the required response time rt_j i.e.,

$$rt_j \le max\big\{\frac{x_i^j w_j}{cr_i} + dt_i\big\}_{i\in|\mathcal{SV}|}. \tag{5}$$

Obviously the total storage resource requirement of all tasks scheduled to server sv_i cannot exceed the storage resource of server sv_i. That is,

$$\forall\ i \in \{1, ..., |\mathcal{SV}|\}, \quad \sum_{j=1}^{|\mathbf{ta}|} x_i^j rsr_j \le sr_i. \tag{6}$$

Example 1: We consider an edge cloud computing system consisting of three edge servers and two cloud servers. Specifically, we take RaspberryPi-2 as edge servers $sv_{1,2}$ and RaspberryPi as edge server sv_3 , and take Amazon EC2 h1.4xlarge as cloud servers $sv_{4,5}$. We refer to published values in determining the value of $(cr_i, sr_i^{max}, dt_i, uc_i)$ for each server i, see Table II (MIPS is Million Instructions Per

Table II
EDGE AND CLOUD SERVERS IN EXAMPLE 1

$server$	cr(MIPS)	sr^{max}(GB)	dt(s)	uc($\$/s$)
sv_1(RaspberryPi-2)	1538	1	0.1	3e-07
sv_2(RaspberryPi-2)	1538	1	0.1	3e-07
sv_3(RaspberryPi)	847	0.5	0.1	3e-07
sv_4(EC2 h1.4xlarge)	5900	64	2	8e-05
sv_5(EC2 h1.4xlarge)	5900	64	2	8e-05

Second).

We assume two types of tasks: $type_1$ $(9000MI, 9s, 0.1GB)$ has high computation workload and high latency delay tolerance, that can be allocated only to cloud servers (MI is Million Instructions); $type_2$ $(1000MI, 2s, 0.1GB)$ has low computation workload but low latency delay tolerance, and can be allocated only to edge servers. Then, at each step a task request queue $\mathbf{ta}$ consisting of several of the two types of tasks would arrive for allocation.

Let us assume a simple case where the length T of trajectory h is 2. Then, at step t=1, the available storage resource of all servers is 0 and with the arrival of two tasks $ta_1 = type_1$ and $ta_2 = type_2$ we have state s_1=$[[0,0,0,0,0],[ta_1,ta_2]]$. We can solve the action space $\mathcal{A}(s_1)$ and choose an action from it. If ta_1 is allocated to sv_3 and ta_2 is allocated to sv_1, the resulting cost $cost_1^{\mathcal{SV}}$= 3e-07+8e-05=8.03e-05 in step t=1. At step t=2, suppose that only one task $ta_1 = type_1$ arrives, and the state transits to s_2=$[[0.1,0,0,0.1,0],[ta_1]]$. As $task_1$ can be allocated to sv_4 or sv_5, we have $cost_2^{\mathcal{SV}}$=16.03e-05 if it is allocated to sv_5 or $cost_2^{\mathcal{SV}}$=8.03e-05 if it is allocated to sv_4. The corresponding average costs, $Cost(h)$, are also different. Thus, the goal is to search for a policy that can minimize $Cost(h)$.

V. ALGORITHM

As stated in the above section, as for a state s, infinite possible actions $\mathbf{x} \in \mathcal{A}(s)$ exist since each task can be performed by infinitely divided among the servers. Although some studies simplify this problem by assuming that each task can just be allocated to one server [10][18], these approaches might not attain a satisfying solution. This is because some tasks with high workload cannot be performed by just one edge server and they have to be offloaded to the cloud servers, whose charges are usually much higher than those of edge servers. Thus, we consider to generate the action space from the perspective of server coalitions: several servers can form a coalition to be as a joint server, then a task allocated to the joint server would be divided proportionally according to each member's computation rate.

First, we take the situation in [10][18] as a benchmark where each task can be allocated only to one server and apply, *R-learning* algorithm, a classical RL algorithm to solve it. Then, we combine CSG with R-learning algorithm to propose a dynamic coalition formation algorithm called *coalitional R-learning* algorithm. Although some other RL algorithms for average reward MDPs can also be utilized, it is not the core of this paper. What we focus on in this paper is to propose a general framework for combining both coalition structure generation and RL algorithms to guide agents how to form coalitions dynamically.

A. R-learning Algorithm

Since the dynamic task allocation problem is cast in average reward MDP form where the goal is to learn an optimal policy π that can minimize the average reward defined as Eq. (2).

Average reward MDPs can be solved by many RL algorithms, e.g., R-learning [19], SMART [20], RVI Q-learning [21]. In this paper, we apply R-learning to solve it. First, we define average reward ρ^π under a policy π when $T \to \infty$ as follows,

$$\rho^\pi = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} cost_t^{\mathcal{SV}}.$$

A state-action value function is normally used to learn the optimal policy; it evaluates the quality that results from taking particular actions in certain states. The state-action value function $Q^\pi : \mathcal{S} \times \mathcal{A}(s) \to \mathbb{R}$ is defined by

$$Q^\pi(s,\mathbf{x}) = \sum_{t=1}^{\infty} \mathbb{E}_{p^\pi(h)}\big[cost_t^{\mathcal{SV}} - \rho^\pi \mid s_1 = s, \mathbf{x}_1 = \mathbf{x}\big],$$

where "$|s_1 = s, \mathbf{x}_1 = \mathbf{x}$" means the initial state and the action are fixed on state s and $\mathbf{x}$, respectively. Then, both $Q^\pi(s,\mathbf{x})$ and ρ^π can be updated according to the R-learning algorithm as follows.

$$Q^\pi(s_t,\mathbf{x}_t) \leftarrow Q^\pi(s_t,\mathbf{x}_t) + \alpha\big[cost_t^{\mathcal{SV}} - \rho_t^\pi + \min_{\mathbf{x}'} Q^\pi(s_{t+1},\mathbf{x}') - Q^\pi(s_t,\mathbf{x}_t)\big],$$

$$\rho_t^\pi \leftarrow \rho_t^\pi + \beta\big[cost_t^{\mathcal{SV}} - \rho_t^\pi + \min_{\mathbf{x}'} Q^\pi(s_{t+1},\mathbf{x}') - Q^\pi(s_t,\mathbf{x}_t)\big],$$

where $\alpha, \beta \in (0,1]$ are learning rates. Given state s_t, action $\mathbf{x}$ with the minimum of $Q^\pi(s_t,\mathbf{x})$ would be taken, i.e.,

$$\mathbf{x}_t = \arg\min_{\mathbf{x}} Q^\pi(s_t,\mathbf{x}).$$

B. Coalitional R-learning Algorithm

In the above part, in order to avoid creating an infinite action space, the problem was simplified by assuming each task can be allocated to just one server, which might cause higher cost than necessary, as those tasks that cannot be performed by just one edge server must be offloaded to cloud servers. Thus, we consider to distribute the tasks from the perspective of server coalitions: first, let some edge servers form several coalitions; then, each task would be allocated to just one coalition or one cloud server; finally, the task allocated to one coalition would be proportionally distributed among the coalition members.

First, we refer to the framework of CSG to deal with how to form coalitions among edge servers $sv_i \in \mathcal{SV}^e$ where one coalition consisting of some edge servers can be regarded as a subset of $\mathcal{SV}^e$, i.e., $c \subseteq \mathcal{SV}^e$. Then, the power set $2^{\mathcal{SV}^e}$ of $\mathcal{SV}^e$ includes all possible joint edge servers c. Since it is assumed that an agent can be allocated to just one coalition, $\mathcal{SV}^e$ can be divided to several disjoint subsets, where each division is called a coalition structure, denoted by cs, i.e.,

$$cs = \{c_1, ..., c_{|cs|} \mid \forall i \neq j,\ c_i \cap c_j = \varnothing \wedge \bigcup_{i=1}^{|cs|} c_i = \mathcal{SV}^e\}.$$

We let $\mathcal{P}^{cs}$ denote a set of all possible coalition structures, i.e., $\mathcal{P}^{cs} = \{cs_1, ..., cs_{|\mathcal{P}^{cs}|}\}$. Then, each coalition of edge servers and each cloud server would be regarded as an entity and each task is assumed to be allocated to just one entity; this corresponds to a new set $\mathcal{SV}^{cs}$ as follows,

$$\mathcal{SV}^{cs} = \{c_1, ..., c_{|cs|}\} \cup \mathcal{SV}^c.$$

Then, in the state s, each $\mathcal{SV}^{cs}$ corresponds to an action space under coalition structure cs defined as $\mathcal{A}^{cs}(s)$. Considering all possible $cs \in \mathcal{P}^{\mathcal{SV}^e}$, we define a joint action space as $\mathcal{A}^{all}(s) = \mathcal{A}^{cs_1}(s) \cup ... \cup \mathcal{A}^{cs_{|\mathcal{P}^{cs}|}}(s)$. Then, in state s, the coordinator would choose action $\mathbf{x}$ from $\mathcal{A}^{all}(s)$. After the task is allocated to a coalition, the task is proportionally divided among the members of the coalition. Although, there are different methods for dividing tasks among a coalition, in this paper we consider a deterministic distributed strategy: task partitioning among the members (edge servers) depends on available storage resource, which means that a server with greater available storage resource would have a greater share of the tasks.

Thus, we combine CSG with R-learning to propose coalitional R-learning (CR-learning) algorithm; it can guide edge servers to form coalitions dynamically while learning the optimal policy, which is shown as Algorithm 1. Compared with R-learning, a task which cannot be performed by just one edge server might be offloaded to a coalition of edge servers rather than a cloud server, which corresponds to lower user cost. Thus, the proposed algorithm can cope with the issues of dynamic features and huge solution space well.

Algorithm 1 Coalitional R-learning

1: Initialize: the set $\mathcal{SV}^{cs}$ of coalition structures given the set $\mathcal{SV}$ of servers, the values of ρ^π and all $Q(s, \mathbf{x})$ to zero
2: **for** episode m=1, M **do**
3: Set the initial state
4: **for** step t =1, T **do**
5: Generate a task request queue $\mathbf{ta}_t$ based on the given distribution and observe all servers' current storage resource $\mathbf{sr}_t$
6: Based on the state $s_t = [\mathbf{sr}_t, \mathbf{ta}_t]$, solve the action space $\mathcal{A}^{cs_j}(s_t)$ for each coalition structure cs_j
7: Solve joint action space $\mathcal{A}^{all}(s_t)$ and select an action $\mathbf{x}_t \in \mathcal{A}^{all}(s_t)$ according to the policy π: $\mathbf{x}_t = \arg\min_{\mathbf{x}} Q(s_t, \mathbf{x})$
8: Execute action $\mathbf{x}_t$ and observe immediate cost $cost_t^{\mathcal{SV}}$ and new state s_{t+1}
9: Update the value of $Q(s_t, \mathbf{x}_t)$ and ρ_t^π as follows
$Q(s_t, \mathbf{x}_t) \leftarrow Q(s_t, \mathbf{x}_t) +$
$\sigma\big[cost_t^{\mathcal{SV}} - \rho_t^\pi + \min_{\mathbf{x}'} Q(s_{t+1}, \mathbf{x}') - Q(s_t, \mathbf{x}_t)\big]$
$\rho_t^\pi \leftarrow \rho_t^\pi +$
$\beta\big[cost_t^{\mathcal{SV}} - \rho_t^\pi + \min_{\mathbf{x}'} Q(s_{t+1}, \mathbf{x}') - Q(s_t, \mathbf{x}_t)\big]$
10: **end for**
11: **end for**

VI. Experiment

In this section, we conduct the performance evaluation of CR-learning algorithm for solving the cost efficient problem in edge cloud computing. We set R-learning algorithm and an approximate linear programming (LP) algorithm [10] as benchmarks to show the effectiveness and improvement of our proposed algorithm. Although the goal of the approximate LP is also to minimize the cost of edge computing systems, unlike the algorithms based on MDP, it just solves an approximate optimal solution deterministically at each state ignoring the influence to the state transition.

Moreover, we investigate how the different tasks' parameters would affect the effectiveness of the algorithms. Specifically, we take an edge computing system given as Example 1, which composes of three edge servers and two cloud servers. Without loss of generality, the value of tasks' parameters are all different, which includes number of tasks in arriving task queue, task workload and task storage.

A. Performance Evaluation

We first check the performance of results obtained by our CR-learning algorithm by comparing with R-learning and the approximate LP method. In order to simulate the uncertain number $|\mathbf{ta}_t|$ of tasks in each arriving task queue, we assume $|\mathbf{ta}_t|$ is identically determined by a probability distribution. In this case, we use a uniform distribution among $(0, max_{number})$ where max_{number} is the maximum number of tasks, i.e., $|\mathbf{ta}_t| \sim unif(0, max_{number})$.

Then, we denote each period as an episode during the learning process and set the length of one episode as T=10. Since, the number of arriving tasks in each step definitively influences user cost, it is hard to directly compare the costs of two episodes with different number of arriving tasks. Thus, we take 200 episodes as one *round* and take the average of user costs in one round as the result. We set the learning rates α and β as 0.9, and use the two types of tasks given in Example 1 (max_{number}=10). We run the following simulations for 10,000 episodes which correspond to 50 rounds. The results shown in Figure 2 indicate the approximate LP method's performance would not improve during the iterations because it considers a deterministic

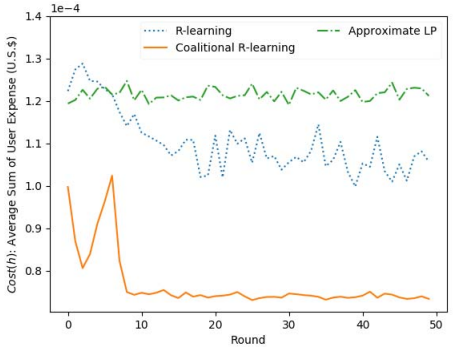

Figure 2. Reduce user cost in edge cloud computing via coalitional R-learning algorithm, R-learning algorithm and approximate LP method

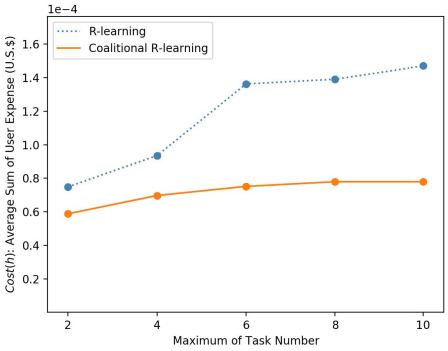

Figure 3. The effect of task number

optimal policy at each state without learning. Moreover, the deterministic optimal policy of the approximate LP method does not consider dynamic feature, thus its performance would be inferior to R-learning which can learn how to cope with the dynamic features. The results also indicate our proposed CR-learning algorithm can decrease user cost by around 30% relative to R-learning, because some tasks that cannot be performed by a single server can be performed by an edge coalition, rather than being offloaded to cloud servers.

B. Parameter Analysis

Next, we analyze the effect of parameters on CR-learning algorithm performance comparing with the R-learning algorithm. The parameters include number of tasks in arriving task queue, task workload and task storage.

1) The Effect of Task Number: In this experiment, we evaluate the performance of our CR-learning algorithm with different maximum number max_{number} of tasks in a task queue. We keep the task workload as 9000MI and set task storage as 0.2GB, then we change the value of the max_{number} from 2 to 10 in steps of 2. We run 10,000 episodes and take the average of the last 1000 episodes as the learning results.

As shown in Figure 3, the results confirm that our algorithm offers lower user cost than R-learning under different maximum number of tasks in a task queue. This advantage increases with the increase of number of tasks to be assigned. This is because as more tasks arrive, the edge coalition can accept the increase in task numbers without offloading them to the cloud servers, which would be the expensive alternative.

2) The Effect of Task Workload: In this experiment, we evaluate the performance of our CR-learning algorithm with different task workloads. We keep the maximum of task number max_{number} at 10 and change task workload from 8000MI to 20000MI in steps of 4000MI; We run 10,000 episodes and take the average of the last 1000 episodes as the learning results.

As shown in Figure 4, the results confirm that our algorithm

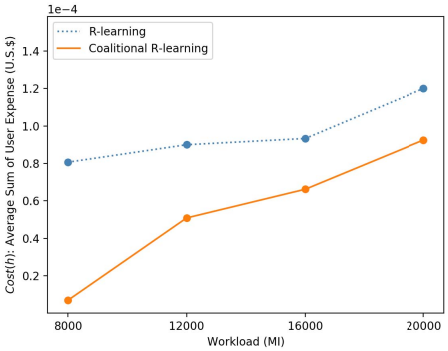

Figure 4. The effect of task workload

offers lower user cost than R-learning under different task workloads. However, this kind of advantage decreases with the increase of task workload. This is because more tasks have to be offloaded to cloud servers when the total workload exceeds the limits of edge coalitions can perform, which invokes more cloud servers and thus increases the cost.

3) The Effect of Task Storage: In this experiment, we evaluate the performance of our CR-learning algorithm with different storage of tasks. We keep the maximum of task number max_{number} at 10 and set the task workload to 9000MI, then we set task required storage as 1MB, 10MB, 100MB, 1GB, respectively. We run 10,000 episodes and take the average of the last 1000 episodes as the learning results.

As shown in Figure 5, we can conclude our algorithm can offer lower user cost than R-learning under different storage of tasks. This advantage increases with the storage resource requirement. This is because the edge server coalitions have enough storage resources to handle the tasks with high storage resource requirements, which can only be allocated to cloud servers in R-learning.

To summarize the above experiments, our algorithm can

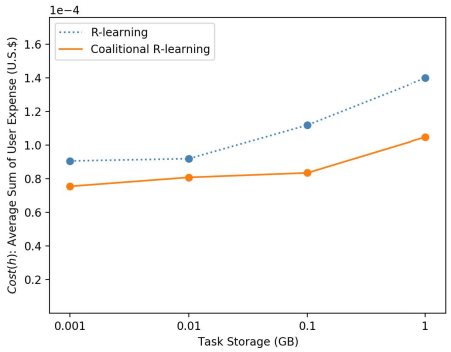

Figure 5. The effect of task storage

guide the edge servers to form coalitions to perform tasks that cannot be processed by just one edge server, without offloading the tasks to the cloud servers. Moreover, the effectiveness of our proposed CR-learning algorithm in reducing user cost has been verified by experiments with various parameter settings.

VII. Conclusion

This paper studied a dynamic task allocation problem of edge cloud computing, with the target of minimizing user cost. A dynamic coalition formation algorithm called coalitional R-learning is proposed to guide edge server cooperation so as to minimize user cost while reducing the solution space. We validated our approach by comparing it with other classical algorithms using parameters taken from published information of actual devices. The results showed our approach achieves significantly lower user cost than other algorithms. We plan to conduct deeper researches like computational complexity analysis and intend to examine the use of some state-of-art reinforcement learning algorithms to enhance the proposal's effectiveness.

Acknowledgement

This research was partially supported by a Grant-in-Aid for Scientific Research (A) (17H00759, 2017-2020), a Grant-in-Aid for Scientific Research (B) (18H03341, 2018-2020), and a Grant-in Aid for Challenging Research (Exploratory) (20K21833, 2020-2022) from the Japan Society for the Promotion of Science (JSPS).

References

[1] J. Pan and J. McElhannon, "Future edge cloud and edge computing for Internet of Things applications," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, 2017.

[2] C. Chang, S. N. Srirama, and R. Buyya, "Internet of Things (IoT) and new computing paradigms," *Fog and Edge Computing: Principles and Paradigms*, pp. 1–23, 2019.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *ACM Workshop on Mobile Cloud Computing*, pp. 13–16, 2012.

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

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

[6] S. Li and J. Huang, "Energy efficient resource management and task scheduling for iot services in edge computing paradigm," in *IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 846–851, 2017.

[7] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 1, pp. 108–119, 2015.

[8] L. L. Dhirani, T. Newe, E. Lewis, and S. Nizamani, "Cloud computing and Internet of Things fusion: Cost issues,"in *International Conference on Sensing Technology*,pp.1–6,2017.

[9] X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu, "Efficient resource allocation for on-demand mobile-edge cloud computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 9, pp. 8769–8780, 2018.

[10] Y. Zhang, X. Chen, Y. Chen, Z. Li, and J. Huang, "Cost efficient scheduling for delay-sensitive tasks in edge computing system," in *IEEE International Conference on Services Computing*, pp. 73–80, 2018.

[11] X. Guo, R. Singh, T. Zhao, and Z. Niu, "An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems," in *IEEE International Conference on Communications*, pp. 1–7, 2016.

[12] L. F. Bittencourt, E. R. Madeira, and N. L. Da Fonseca, "Scheduling in hybrid clouds," *IEEE Communications Magazine*, vol. 50, no. 9, pp. 42–47, 2012.

[13] J. Yu, R. Buyya, and C. K. Tham,"Cost-based scheduling of scientific workflow applications on utility grids," in *International Conference on e-Science and Grid Computing*,pp.8–16,2005.

[14] T.Verbelen, P.Simoens, F. DeTurck, and B.Dhoedt,"Cloudlets: Bringing the cloud to the mobile user," in *Workshop on Mobile Cloud Computing and Services*,pp.29–36,2012.

[15] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, 1957.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[17] T. Rahwan, T.-D. Nguyen, T. Michalak, M. Polukarov, M. Croitoru, and N. R. Jennings, "Coalitional games via network flows," in *International Joint Conference on Artificial Intelligence*, pp. 324–331, 2013.

[18] Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue, "An approach to QoS-based task distribution in edge computing networks for iot applications," in *IEEE International Conference on Edge Computing*, pp. 32–39, 2017.

[19] A. Schwartz, "A reinforcement learning method for maximizing undiscounted rewards," in *International Conference on Machine Learning*, vol. 298, pp. 298–305, 1993.

[20] T. K. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck, "Solving semi-markov decision problems using average reward reinforcement learning," *Management Science*, vol. 45, no. 4, pp. 560–574, 1999.

[21] J. Abounadi, D. Bertsekas, and V. S. Borkar, "Learning algorithms for markov decision processes with average cost," *SIAM Journal on Control and Optimization*, vol. 40, no. 3, pp. 681–698, 2001.