IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received March 26, 2020, accepted April 5, 2020, date of publication April 13, 2020, date of current version April 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2987574

Online Scheduling Optimization for DAG-Based Requests Through Reinforcement Learning in Collaboration Edge Networks

YAQIANG ZHANG[1], ZHANGBING ZHOU[1,5], ZHENSHENG SHI[2], LIN MENG[3], AND ZHENJIANG ZHANG[4]

[1]School of Information Engineering, China University of Geosciences (Beijing), Beijing 100083, China
[2]PetroChina Research Institute of Petroleum Exploration and Development, Beijing 100083, China
[3]College of Science and Engineering, Ritsumeikan University, Shiga 525-8577, Japan
[4]School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China
[5]Computer Science Department, TELECOM SudParis, 91000 Evry, France

Corresponding author: Zhangbing Zhou (zbzhou@cugb.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61772479 and Grant 61662021.

ABSTRACT The wide-adoption of edge computing promotes the scheduling of tasks in complex requests upon smart devices on the network edge, whereas tasks are necessary to be offloaded to the cloud when they are intensive in computational and energy resources. Traditional techniques explore mostly the scheduling of atomic tasks, whereas complex requests scheduling on edge servers is the challenge unexplored extensively. To address this challenge, this paper proposes an online task scheduling optimization for DAG-based requests at the network edge, where this scheduling procedure is modeled as Markov decision process, in which system state, request and decision space are formally specified. A temporal-difference learning based mechanism is adopted to learn an optimal tasks allocation strategy at each decision stage. Extensive experiments are conducted, and evaluation results demonstrate that our technique can effectively reduce the system's long-term average delay and energy consumption in comparison with the state-of-art's counterparts.

INDEX TERMS Online DAG-based request optimization, edge computing, temporal-difference learning, Internet of Things.

I. INTRODUCTION

With the increasing capacity of smart devices in the *I*nternet *o*f *T*hings (*IoT*), complex requests are possible to be conducted through the collaboration of IoT devices at the network edge, such that tasks in certain requests are independently assigned to appropriate edge servers. Consequently, edge computing is brought nowadays to schedule tasks upon edge servers in an optimized fashion, in order to guarantee the *Q*uality *o*f *E*xperience (*QoE*), especially when requests are latency-sensitive and network traffic intensive. On the other hand, tasks in requests are mostly offloaded to the cloud, when they are intensive in computational and energy resources and thus can hardly be hosted by edge servers. Generally, tasks to be offloaded to the cloud can be determined before tasks scheduling on the network edge, and this offloading operation can be effectively supported by techniques developed recently by researchers [1]–[4]. Therefore, this paper focuses on the scheduling of remaining tasks upon edge servers, such that the QoE of requests, including temporal/latency constraints specified between tasks and energy consumption by edge servers, are the perspectives to be considered.

Techniques have been developed to support task scheduling at the network edge. Most of current techniques aim to realize high efficiency of edge networks for the scheduling of atomic tasks [5]–[8]. Besides, techniques are developed to support complex requests scheduling on edge servers [9]–[12]. Without loss of generality, *D*irected *A*cyclic *G*raph (*DAG*) is adopted to represent requests with complex structures, where vertices in a DAG represent tasks, while edges represent dependency relations between tasks. Although a sequence or parallel task structure is a particular case of DAG, the general case has gained wider attention [13].

The associate editor coordinating the review of this manuscript and approving it for publication was Tie Qiu.

Currently, most literatures concentrate on scheduling a single DAG-based request in certain time durations. A request is processed step by step according to the partial ordering between tasks. The goal is to achieve a scheduling for a particular request that minimizes system cost. Generally, system only cares about the optimality of current request and it is doubtful whether the global optimization can be achieved. Furthermore, when there are multiple unfinished requests in the system, there will be overlaps in their scheduling process. A challenge in edge networks is to ensure the global optimization of scheduling and reduce long-term consumption in the presence of multiple interacting tasks.

To address the challenge in DAG-based requests scheduling optimization, this paper proposes an online *R*einforcement *L*earning (*RL*) based tasks scheduling mechanism, where tasks allocation strategies are learned incrementally. The system's long-term delay and energy consumption are minimized and temporal constraints between concurrent requests are satisfied. The contributions of this paper are summarized as follows:

- We develop an online scheduling mechanism for requests with complex structures, which is modeled as a *M*arkov *D*ecision *P*rocess (*MDP*). When modeling state and action of this MDP, internal temporal constraints of concurrent requests, network load and processor occupation of edge servers are taken as a component of the system state. To improve the efficiency of decision making, we set up a reward function aiming to reduce the system's long-term delay and energy consumption while improving the QoE of certain requests.
- We develop a *T*emporal-*D*ifference (*TD*) learning based mechanism to learn the optimal tasks allocation strategy at each decision stage. Since state scale of this problem is quite large, calculating a value function may lead to a curse of dimensionality. Therefore, a linear approximation is adopted to contribute the parameterized functional form of value function for each state. At the same time, for the purpose of ensuring the long-term optimal of the system, we use gradient descent method to approximate the value function weight settings.

Extensive experiments are conducted to evaluate the applicability and performance of our technique. Evaluation results indicate that, with the continuous running of the system, task allocation decisions are promising compared with the state-of-art's techniques, and thus minimizing the long-term delay and energy consumption of edge networks.

This paper is organized as follows. Section II reviews and discusses relevant techniques. Section III describes the system model. Section IV formulates the *O*nline *O*ptimization for *D*AG-based *R*equests *P*roblem-*MDP* (OODRP-MDP) problem model. The proposed online edge server collaboration optimization mechanism is discussed in Section V. Section VI presents the experiments we have conducted to evaluate the proposed mechanism and Section VII concludes this work.

II. RELATED WORK

Task scheduling in edge networks has become the focus of recent studies [14]–[17]. The structure of application requests has become more and more complex as users demand more and more functionality. Although tasks in most of literatures are described as an atomic form, tasks with more complex structures such as a DAG are considered in recently works.

Mao et al. [18] summarize the typical topologies of task graphs as sequential dependency, parallel dependency and general dependency. The advantage of tasks graph is that it shows the interdependencies between tasks and allows for better differentiation of subtask characteristics. As the number of mobile devices grows, people are increasingly relying on handheld devices to access web services such as mobile phones. Because of the limited resources and capabilities of mobile devices themselves, users need assistance of other computing resources to handle complex tasks through mobile phones. Zhang et al. [19] discuss optimizing the QoE of ability constrained devices through collaborative mobile computing. They focus on ensuring the QoE, user incentives and other challenges such as networking technologies utilized in multi-devices collaboration and they point out the future directions in real-time device collaborative systems. Calice et al. [20] propose a task splitting and offloading mechanism which leverage nearby computational resources between mobile devices. A request can be split to chunks sizes and offloaded to other devices according to task matching degree. These works primarily consider the offloading of complex structured tasks between constrained devices, with little discussion of offloading to more capable servers. With the development of cloud computing edge computing, tasks can be offloaded to such servers to get better QoE.

There have been further studies on task unloading with complex structure. In [21], [22], heuristic algorithms are applied to optimize concurrent tasks in mobile cloud or edge computing. Jia et al. [21] propose a mechanism to offload tasks to cloud while tasks mentioned are either sequential type or concurrent type. The purpose is to minimize completion time of application on mobile device while a load-balancing heuristic is used to guide the offloading processing. Shu et al. [22] discuss the shortage of pruning method and they propose a heuristic to search out a near-optimal offloading scheme for non-linear topology under real-time constraint. They divide the topological graph into a set of linear paths according to Partial Stochastic Path and a named One-climb Policy is applied to obtain the optimal offloading decision for each path of task graph. Guo et al. [23] propose a mechanism which achieves energy-efficient computation offloading under hard constraint for application completion time. They develop an energy-efficient dynamic offloading and resource scheduling policy which reduces energy consumption and shorten application completion time while satisfying task-dependency requirement and completion time deadline constraint. They also explore major factors that influence the optimization of complex task execution, such as computing workload of a task, maximum completion time of its immediate predecessors and clock frequency or transmission power of the mobile device. In this kind of works, only one task graph is considered at once while in fact, the arrival of tasks may be an overlapping process.

In task scheduling context, according to the way it optimizes the target problem, we divide it into two types, online optimizing and offline optimizing. Compared with offline problem, an online problem doesn't use the global information but the information in current state to optimize the task scheduling [24]. Since online optimization problems are often modeled using an MDP, solving specific problems can be converted into the process of finding an optimal solution of MDP. For this reason, RL becomes popular solving online optimizing problem as it can effectively solve an MDP [25]–[27]. The advantage of RL mainly include several aspects: RL has strong self-learning ability. Through action-return, RL interacts with the environment and constantly approaches the optimal target; RL requires less environment information than DP and only partial information can be applied to the optimization process; RL can also solve the problem with large-scale of system state and action set through function approximation [28]–[30].

The idea of edge servers collaboration is still relatively new. Current work mainly considers a strong edge server serving the network area it covers while through edge servers collaboration, the execution efficiency of DAG-based tasks can be effectively improved. In our previous work [31], we have explored the possibilities of edge servers collaboration. In that paper, concurrent requests represented in terms of SQL queries are rewritten as atomic queries and these atomic queries are optimally assigned to edge servers through adopting an algorithm inspired by minimum spanning tree. It tries to distribute the tasks evenly on edge servers, while taking into account several factors that affect QoE. The shortage is that it doesn't take into account the interplay of requests arriving at different times. In general, there is still a lack of studies on online optimization of concurrent DAG-based application requests based on collaboration of edge servers. Our main work in this paper focuses on solving such problems.

III. SYSTEM MODEL

In this section, the basic concepts of an collaboration edge networks and a necessary description for the system are illustrated. Some of the symbols are listed in Table 1.

A. IOT EDGE NETWORKS SYSTEM

As shown in Figure 1, users send their request across a nearby Base Station (*BS*) [32], [33]. As a basic component of the IoT edge network, Edge Servers (*ES*) are responsible for storing IoT sensing data from IoT devices and processing user requests. In general infrastructure construction practice, *ES* often located near a *BS*. To keep it simple, we use *ES* to represent the combination of them. Depending on the function of base station network, *ES* can establish communication links with all others edge servers and exchange data. This becomes the basis for collaboration in edge networks.

TABLE 1. Notations.

Symbol	*Meaning*
BS	Base Station
ES	Edge Server
N	Number of Edge Node
Req	User Request
$SubTask$	Subtask of a user Request
Dt_n	Data required by a Req from ES_n
TC	Time constraint from related SubTask
$Proc$	Task scale for a SubTask
Q	Data package size
$T_{ij}(Q)$	Time to transmit Q units of data between ES_i and ES_j
$TP(Q)$	Task process time with Q units of data
$ET_{ij}(Q)$	Energy cost when transmit Q units of data
$EP(Q)$	Energy cost when process Q units of data
E_{elec}	Energy cost when transmitting a unit of data
B_{ij}	Bandwidth between two edge servers
d_{ij}	System delay
Z	Number of CPU cycles required to process a unit of data
F	Frequency of a processor
t_{idle}	Processor idle time period
$P1$	Energy cost per unit of time when a processor is busy
$P2$	Energy cost per unit of time when a processor is free
$BTP(E)$	Beginning time of an event E
$ETP(E)$	End time of an event E
$c_d(i,t)$	Delay of ES_i at time t
$c_e(i,t)$	Energy cost of ES_i at time t
CP_i^t	Processor free time point of ES_i observed at t
$L_{i,j}^t$	Channel idle moment between ES_i and ES_j observed at t
RT_t	Set of subtasks that can be executed at t
$s(t)$	System state at time t
$a(t)$	Allocation decision at time t

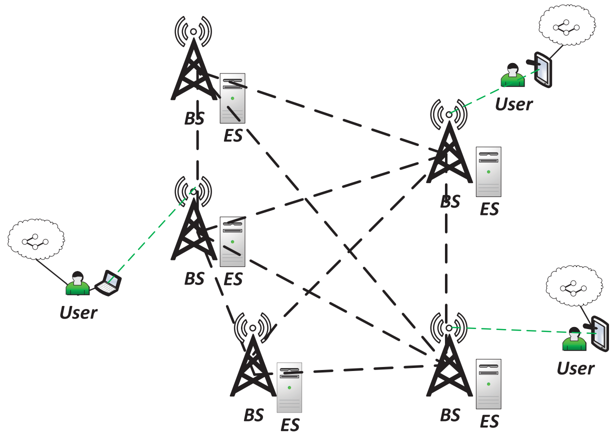


FIGURE 1. Network structure.

We use a discrete time system where time is divided into equal length time slots. In each time slot, we consider that a user request or an application comes according to a distribution and unprocessed tasks are placed in a queue. In the beginning of each time slot, there is a certain probability that one or no request will arrive in the system. Then system would decide how to respond to user requests in a very short period of time during the initial phase, which is almost negligible compared to the whole time slot. In this paper, user requests or applications are not answered by only one *ES* because they are always made up of different structures in practice, which requires more sophisticated analysis. In fact, a user request or an application with complex structure are usually completed over multiple time slots, not over a single time period. We look more deeper in this view. The structure of user requests or applications will discuss later.

B. IOT USER REQUEST MODEL

For a user request *Req*, it consists of two parts: a set of subtasks and a set of time constraint between subtasks. It can be expressed as:

$$Req = \{\{SubTask_k\}_{k\in K}, \{TC_w\}_{w\in W}\} \tag{1}$$

where there are K number of subtasks in this *Req* and W number of links *TC* between subtasks with time constraints. We consider that a user request has a structure like a DAG.

For a node which represents a *SubTask* in DAG, it consists of three parts: a set of required data Dt_n from different data sources $n \in N$, a set of input links *TC* and the process *Proc* which represents a function to deal with the input data. We have:

$$SubTask_i = \{\{Dt_n\}_{n\in N}, \{TC_w\}_{w\in W_i}, Proc\} \tag{2}$$

where W_i in equation (2) denotes the number of links point at $SubTask_i$. Figure 2 shows an example of a DAG-based request structure. For any subtask in a request, there must be some kinds of data that are required from other resources. $t_1, t_2 \ldots$ mentioned in the figure indicate time constraints in *TC*. Two *SubTasks* connected by a *TC* mean that the subsequent task need to be executed at some time after the preorder task completes.

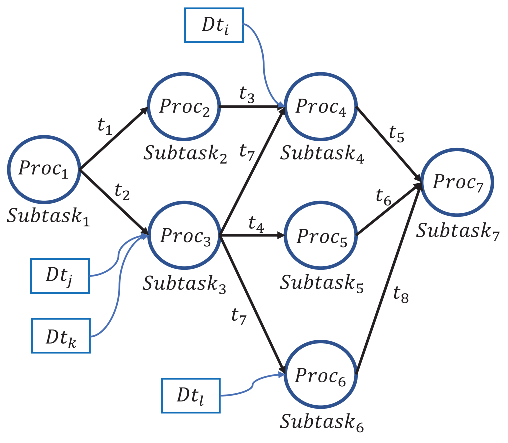


FIGURE 2. DAG-based user task structure.

C. DATA TRANSMISSION MODEL

1) TIME CONSUMPTION

ESs connect each other by adopting the *O*rthogonal *F*requency *D*ivision *M*ultiple *A*ccess (*OFDMA*) technique. An edge server's channel is divided into subcarriers of different frequency bands, each of which is responsible for communication with another specific *ES*. The network communication mode between servers is two-way propagation mode. B_{ij} denotes bandwidth between two edge servers ES_i and ES_j and there is a system delay d_{ij} that is influenced by network condition. Assume that a package with size of Q will be transmitted between edge servers, we can calculate the transmission time as follow:

$$T_{ij}(Q) = \frac{Q}{B_{ij}} + d_{ij} \tag{3}$$

2) ENERGY CONSUMPTION

In data transmission processing, the energy consumption for data package happens at the receiver and sending ends. As base stations connect with each other by central switchboard, we consider that the energy consumption on both sides are the same when a data package is transmitted. It can be calculated as follow:

$$ET_{ij}(Q) = E_{elec} * Q \tag{4}$$

where E_{elec} denotes energy cost when transmitting a unit of data within two edge servers.

D. DATA PROCESSING MODEL

1) TIME CONSUMPTION

We set that each processor of an edge server has ability to process tasks. Delay of task processing is proportional to the amount of required data received by the server. F denotes CPU frequency of an edge server and Z denotes the number of CPU cycles required to process a unit of data. If a task with data of size Q is processed, the process time *TP* is shown as:

$$TP(Q) = \frac{Z * Q}{F} \tag{5}$$

2) ENERGY CONSUMPTION

When it processing a task, the energy costs are calculated by:

$$EP(Q) = TP(Q) * P1 + t_{idle} * P2 \tag{6}$$

where $P1$ denotes the energy consumption per unit of time when the processor is busy and $P2$ is the energy consumption when it is free. t_{idle} denotes CPU idle time.

IV. PROBLEM DEFINITION

In this section, we formulate the online optimization for complex requests in collaboration edge computing networks as an average return MDP in infinite time domain.

A. SYSTEM STATE

As we have mentioned above, a discrete time system is applied in this paper. The various system states we observe are based on each time slot and that always happens in the beginning of each time slot. $s(t)$ represents the system state at time slot t. In each time slot $t \in \{1, 2, 3, \ldots\}$, ES_i can be observed for its CPU idle time CP_i^t, as well as its occupancy on each channel connected to other servers $L_{i,j}^t$. i and j mark the number of ES. We use a three tuple to describe the system state:

$$s(t) = \{CP_i^t, L_{i,j}^t, RT_t\}_{i,j\in N} \tag{7}$$

where $\{CP_t^i\}_{i\in N}$ can be expressed as a $1 \times N$ matrix and each element represents an ES's processor free time point. $\{L_{i,j}\}_t, i, j \in N$ is an $N \times N$ matrix where the row and column number corresponding to an element of the matrix represents the idle time of the network transmission between the corresponding two edge servers.

The third part $RT_t = \{SubTask_m^{Ready_t}\}_{m\in M}$ represents the set of subtasks that can be executed at the current time slot. According to equation (2), due to the impact of request structure and temporal dependencies between subtasks, the execution of a request's subtasks have sequential relationships, which usually happens within multiple time slots. The observed executable subtasks often belong to several different user requests or applications. $\{Dt_n\}_{n\in N}$ in a $SubTask_m^{Ready_t}$ can be expressed as an $1\times N$ matrix where each element represents the size of data package from a specific edge servers. A $SubTask_m^{Ready_t}$ also means that its preorder tasks have been done and their results are existing in specific ESs. For all $\{TC_w\}_{w\in W_i}$ belonging to this task, the key is to find the link with most pressing time constraint. At the same time, it is clear at which ES the starting $SubTask$ of this link is executed. So an array of two variables can be used to represent the state of a task. A $SubTask_m^{Ready_t}$ can be expressed as a 1×2 matrix. Since the maximum value of M is N, we represent the RT_t as a $2 \times N$ matrix. If M is less then N, we use zero to represent the remaining. The system state $s(t)$ at time slot t can be expressed as an $N+(N\times N)+2\times N$ matrix. A visual image of $s(t)$ is shown in Figure 3.

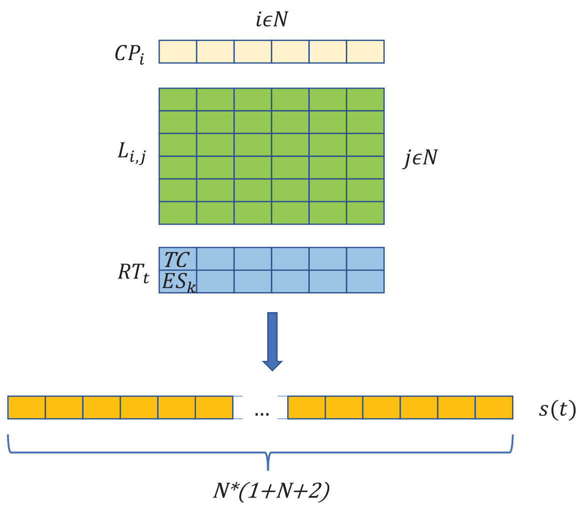


FIGURE 3. System state $s(t)$.

B. ACTION SET

In each time slot, the system would take action to allocate a set of subtasks RT_t to corresponding edge servers. ES can accept a task or not. That can be expressed as a binary sequence of length N. But that description could not show the specific assignment of corresponding edge servers. Further, we have defined the maximum length of the tasks queue $MaxR$, if $MaxR > N$, maybe one or more ES would be assigned to more than one subtasks. It is natural that if an assigned task cannot be executed immediately, it is best to reassign it at the next available time. According to this, we set the maximum length of the queue $MaxR = N$. It means that each ES would be assigned with at most one tasks or not. Action token at time slot t is $a(t) = \{ES_i\}, i \in N$ and it is a permutation of all active ESs. As an example, we assume that there are 6 ES in networks, that means the maximum length of the queue is 6. If there are four subtasks available at a time, then we will assign them to four ESs. An assignment of the four tasks $\{5, 3, 2, 1, 4, 6\}$ means that only ESs numbered $\{5, 3, 2, 1\}$ are assigned with tasks while $\{4, 6\}$ are free at this time slot. If there are eight or more tasks can be executed at a time slot, except the first six tasks, redundant tasks will be handled in the next time slot.

C. TRANSITION

The transition function is determined by current system state $s(t)$ and action $a(t)$. When we take an action at time t, its essence is that CP^t and L^t are partly or completely updated while RT_t is completely updated. The ES assigned to a task will access the corresponding data and then process them locally and its CPU idle time will change accordingly, as well as the channel idle time for data transmission. The CPU idle time of a ES without assigned any tasks is automatically updated to the next moment. For RT_t part, the tasks represented by all rows are zeroed out by assignment, and at the beginning of the next moment, new available tasks that can be done is observed and their parameters are filled in RT_{t+1}. The transition function can be expressed as follow:

$$P^a_{s(t)s(t+1)}(t) = P[S_{t+1} = s(t+1)|S_t = s(t), A_t = a(t)] \tag{8}$$

D. REWARD

The proposed edge servers collaboration optimization is a multi-objective optimization problem, where meeting the internal time constraint of user requests and minimizing the long term system costs are two major problems. Quality of Experience (QoE) often shows that weather a request is satisfied by edge networks. We use T_m to indicate the QoE of an user request. If it is satisfied, $T_m = 1$, else $T_m = 10$. That means, if a task is finished before its time constraint, it is a positive return, otherwise it is a negative return. System cost mainly contain two parts, time cost and energy cost. For a long-running system, it is more realistic to focus on average costs of system over a long period of time than on minimum costs over a short period of time. First we define the delay of an ES_i after an action $a(t)$ has been taken at time t as:

$$c_d(i, t) = ETP(TP(Q_{i,t})) - BTP(t) \tag{9}$$

where $BTP(E)$ denotes the beginning time of an event E. $ETP(E)$ denotes the end time of an event E. $Q_{i,t}$ is the size of the data processed by ES_i at time t. In fact, the delay mainly consists of total transfer time width of data packages, waiting time after the transfer and processing time of the task.

The energy consumption of an ES_i at time t is mainly generated by process of data transmission and dealing with a task at processor. $c_e(i,t)$ denotes the energy consumption. We have:

$$c_e(i,t) = ET_{i\{j \in N\}}(Da) + EP(Dr) \tag{10}$$

where Dr denotes sum of the amount of data requested by the task. D_a denotes the amount of data that passes through this server.

Based on above premises, the long term average weight sum of delay and energy cost C is given as:

$$\begin{aligned} P1 \quad min(C) &= min \sum_{i=1}^{N} (w_1 \overline{c_d(i)} + w_2 \overline{c_e(i)}) \\ &= min \sum_{i=1}^{N} \frac{1}{t} \lim_{t\to\infty} [w_1 \sum_{t=1}^{\infty} c_d(i,t) \\ &\quad + w_2 \sum_{t=1}^{\infty} c_e(i,t)] \\ &= min \lim_{t\to\infty} \frac{1}{t} \sum_{t=1}^{\infty} \sum_{i=1}^{N} [w_1 c_d(i,t) + w_2 c_e(i,t)] \\ &= min \lim_{t\to\infty} \mathbb{E} \sum_{i=1}^{N} [w_1 c_d(i,t) + w_2 c_e(i,t)] \end{aligned} \tag{11}$$

where $\overline{c_d(i)} = \frac{1}{t} \lim_{t\to\infty} \sum_{t=1}^{\infty} c_d(i,t)$, $\overline{c_e(i)} = \frac{1}{t} \lim_{t\to\infty} \sum_{t=1}^{\infty} c_e(i,t)$ and w_1, w_2 are the weights. The optimization target **P1** is minimizing expectation of C of the system in infinite time domain.

V. PROPOSED MECHANISM

A. TD LEARNING WITH APPROXIMATION

Different from Monte-Calo (MC) method, in a TD learning method, people don't need to sample from whole sequence and it can learn directly for raw experience without a model of environment's dynamics [34]. In practice, many MDP problems are with an infinite state. A lookup table or MC with all sampling are unsuitable. It is impractical to sample the whole sequence because there is no ending for a infinite sequence. In a TD learning model, it only needs information about a few continuous states instead of true value sample from a whole sequence to update the value function. It calculates the TD error in each step which is the estimate made at that time slot. By minimizing the TD error, we improve the estimating of value function and get the policy with optimal.

Furthermore, an MDP problem with a large scale of state set or action set is not handled in a normal TD learning method. To calculate the action value function or value function, it needs a lookup table which stores values of all state action pairs. It's not practical to store and calculate the value function for all possible states actions because of the curse of dimensionality. As an example, the problem proposed in this paper whose state consists of edge server channel and processor condition, cannot be enumerated as time is infinite. In order to get the values, the idea of value function approximation can effectively solve large-scale problems. It computes the value function in input state by introducing a network of value functions described by parameter form. This network of value function can be described by a linear parameter function. In approximation of value function, a state action pair value can be calculated immediately although it's not the actual value. To improve the approximation value it would use the idea of gradient descent. The gradient function is same with approximation value function and the purpose is to find the minimum of the function. That means parameters should be adjust in direction of gradient descent until it find one with the minimal mean-squared error between approximate value function and true value function.

Reward can reflect the quality of policy taken in system. There are three kinds of reward in reinforcement learning method, finite time domain reward, infinite time domain discount reward and infinite time domain average reward setting. While the first reward setting is applied for episodic problem, last two model are suitable for continuing problem. In a discount reward setting, agent focuses more on recent rewards while rewards at a distant time have a much smaller impact on present state. In an average reward setting, it is equally important to consider the immediate reward of all moments as it calculate the mean of each rewards.

B. LINEAR VALUE FUNCTION APPROXIMATION-BASED ONLINE OPTIMIZATION FOR CONCURRENT REQUESTS

As system state is a collection that cannot be enumerated, we use linear value function approximation-based optimization algorithm to address this problem. An average reward setting in the proposed MDP is applied.

As shown in equation (14), **P1** is the target of the proposed online optimization for concurrent requests allocation based on collaboration of edge servers. The subitem of **P1** is the weight sum of energy consumption and delay for all ES within a time slot period. We have set this as an immediate reward R_t for it reflects the total cost of the system in current phase. When a task's temporal constraints are not met, the system's immediate return is negative and the system gains when all tasks meet their temporal constraints. We set the immediate reward R_t at $\{s(t), a(t)\}$ as:

$$R_t = T_m \sum_{i=1}^{N} [w_1 c_d(i,t) + w_2 c_e(i,t)] \tag{12}$$

$$T_m = \begin{cases} 1 & \text{QoE is satisfied} \\ 10 & \text{QoE isn't satisfied} \end{cases} \tag{13}$$

In a TD learning method, it would calculate value function $q(s, a)$ in each steps while its format is average reward. In each time slot we have:

$$\begin{aligned}\overline{R} &= \lim_{t\to\infty}\frac{1}{t}\sum_{t=1}^{\infty}\sum_{i=1}^{N}[w_1c_d(i,t)+w_2c_e(i,t)]\\ &= \lim_{t\to\infty}\mathbb{E}[R_t|s_0, a_{0:t-1}, \pi]\\ &= C \end{aligned} \tag{14}$$

The return G in an average reward system is:

$$G_t = R_{t+1}-\overline{R}+R_{t+2}-\overline{R}+R_{t+3}-\overline{R}+\ldots \tag{15}$$

According to bellman expectation equation:

$$\begin{aligned}q(s,a) &= \mathbb{E}[G_t|S_t=s, A_t=a]\\ &= \mathbb{E}[R_{t+1}-\overline{R}+q(s',a')]\end{aligned} \tag{16}$$

To update q value, TD learning method is proposed:

$$q(s,a) = q(s,a)+\alpha[R_{t+1}-\overline{R}+q(s',a')-q(s,a)] \tag{17}$$

TD error is denoted as:

$$\delta = R_{t+1}-\overline{R}+q(s',a')-q(s,a) \tag{18}$$

$$\overline{R} = \overline{R}+\beta\delta \tag{19}$$

where $\overline{R}$ is average reward which is the same with target C. G_t is the return under average reward setting. We update the average reward according to equation (19). In TD error, $q(s', a)$ is generated by $\epsilon - greedy$. In this setting, we would get the optimal target C if we can find the optimal q value for each action and state. The proposed MDP problem can be solved as long as we find the $q_*(s, a)$ in each step.

We have mentioned above, the state consists of an $(N\times 1)+(N\times N)+(2\times N)$ matrix. Action set is a $1\times N$ matrix which direct the tasks allocation in a time slot. As a parameterized representation of action-value, the combination of two above matrix Action and State are applied. We also give an parameter matrix ω which has the same dimension. We use linear action value function approximation method for TD learning and we use Stochastic Gradient Descent (SGD) to update the networks. Two step size parameters are set as α and β while α is set to update the TD errors for networks, β is set to update the average reward. We don't set a decay factor γ because we have set average reward in this setting. The explore rate is set to ϵ. And in each step, we output the optimal Q value network. In SGD, the target is to find a parameter ω with the minimal mean-squared error between approximate value function and true value function.

In the initialization, a Q value parameters network is set with arbitrarily, for example we set all parameters as zero. There is no end in this system and the agent interacts with the environment continuously. In the beginning of a time slot, a state is observed by agent and it takes an action from action set with $\epsilon - greedy$ strategy. The selected action which represents the allocation of tasks to each ES is executed and it transfers to a new state $s(t+1)$ and the immediately reward for state $s(t)$ and action $a(t)$ pair is R. Then under state $s(t+1)$, it choose a new action $a(t+1)$ according to $\epsilon-greedy$ strategy. After that we update parameters in the system. TD error is calculated by $\delta \leftarrow R-\overline{R}+\overline{q}(s(t), a(t), \omega)$ and we also update the average reward $\overline{R} \leftarrow \overline{R}+\beta\delta$. Finally the parameter ω is updated according to $\omega \leftarrow \omega+\alpha\delta\omega(s(t), a(t))$. The algorithm flow is shown in Algorithm 1.

```
Algorithm 1 TD Learning for OODRP-MDP
Require:
    A differentiable action-value function parameterization
    $\overline{q}(S, A, \omega) \in N\times(1+N+2)$
    Algorithm parameters: step sizes $\alpha$, $\beta$
    Initialize value-function weights $\omega \in N\times(1+N+2)$
    arbitrarily ($e.g.$, $\omega = 0$)
    Initialize average reward estimate $\overline{R}$ arbitrarily ($e.g.$,
    $\omega = 0$)
    Initialize state $s(0)$, and action $a(0)$
    LOOP for each time slot
1:  In state s(t), take action a(t)
2:  Obvious reward $R$, $s(t+1)$
3:  Choose $a(t+1)(\epsilon - greedy)$
4:  Update TD error $\delta \leftarrow R-\overline{R}+\overline{q}(s(t), a(t), \omega)$
5:  Update average reward $\overline{R} \leftarrow \overline{R}+\beta\delta$
6:  $\omega \leftarrow \omega+\alpha\delta\omega(s(t), a(t))$
7:  $s(t) \leftarrow s(t+1)$, $a(t) \leftarrow a(t+1)$
```

VI. IMPLEMENTATION AND EVALUATION

In this section, we investigate the performance of the proposed online optimization mechanism for edge servers workload allocation and compare it with other mechanisms.

A. EXPERIMENTAL SETTINGS

A prototype of the mechanism was implemented in a Java program. For these experiments, we conducted a numerical simulation to implement concurrent user requests arrival, communication among edge servers and costs under the proposed workload allocation mechanism. Some of the evaluation parameters are listed in Table 2.

TABLE 2. **Parameter settings.**

Parameter name (*notation*)	*Value and Unit*
Number of edge servers (N)	6
Length of time slot	1 unit
Channel frequency between ES (B)	4000 Bytes/unit
CPU frequency (F)	3000–5000/unit
CPU cycles required for a byte (Z)	1
Energy cost when CPU is free	0.1-0.25 J/unit
Energy cost when CPU is busy	0.9–2.5 J/unit
Energy cost when transmitting a bit of data	1 J/unit
Request's arrival rate ($Prob$)	0 – 1
K1 and K2	0 – 1
Tm	1 or 10

In this paper, there are 6 edge servers located in the edge networks. They communicate and exchange data through nearby base stations. To simulate a real-world scenario, edge servers are divided into three types with different data processing capabilities. F which represents the frequency of server processor is set to 3000, 4000 and 5000 per time unit, correspondingly. Energy costs when CPU is free and busy are set to 0.1 to 0.25 and 0.9 to 2.5 J per time unit. The energy cost of data transmission over channels is set to 1 J per time unit. The channel data transmission capacity among servers is set to 4000 bytes per time unit. We also set the length of time slot is a time unit. In order to verify the performance of the proposed mechanism in different scenarios, we apply different user request arrival rate in each time slot from 0.2 to 1. This means that a user task must arrive at the beginning of the time slot when *Prob* is set to 1.

In this setting, we explore the influence of different system parameters on proposed mechanism's efficiency by observing the system's long-term average return, delay, energy consumption and whether user requests are completed within their time constraints. These system parameters include frequency of incoming user requests, weights of energy consumption and delay and immediately reward parameter *Tm*.

B. EVALUATION RESULTS

1) IMPACT OF DIFFERENT TASKS ARRIVAL RATE

In first set of experimental results from Figure 4 to Figure 7, we compare the effects of different user request arrival rates on each metric. The value of $K1$ and $K2$ are equal to 0.5. *Prob* is set to 0.2, 0.4, 0.6, 0.8 and 1.

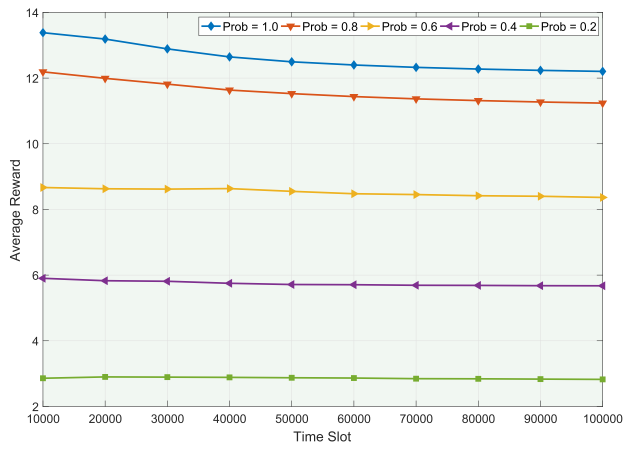


FIGURE 4. Runtime average reward composition.

Figure 4 shows the average system reward under different *Prob*. Since the reward directly reflects the decision value at each decision stage, according to the discussion in this paper, the immediate reward represents the weight sum of delay and energy consumption generated by the system when processing tasks at each time slot. So the smaller *Prob* is, the more efficient the system is. As time goes on, the curve changes significantly when *Prob* is greater than 0.6. This means that the longer the system runs, the more efficient the system will be in making decisions, which is also in line with the expectation of the mechanism proposed in this paper.

As *Prob* gets smaller and smaller, the curve floats less and less over time. This is because the system is faced with less and less tasks in each time slot, so that the system can calmly deal with the arrival of tasks. The number of tasks processed by each edge server is small, so the cumulative delay of the system has little impact on subsequent tasks. Only when the task comes more and more frequently can the continuous optimization of the mechanism in this paper become more obvious.

According to Figure 5 and Figure 6, we can analyze the specific reasons that influence the trend of average return shown in Figure 4. It is clear that with the same weighting of delay and energy consumption, the system average delay directly affects the long-term average return. In Figure 6 different curves change little over time. However, the change trend of curves in Figure 5 is similar to that in Figure 4. Under a certain *Prob*, delay of the system processing task decreases continuously as time goes by. These two figures also show that the curve corresponding to a larger *Prob* is generally higher than the curve with a smaller *Prob* value, which means that when the system burden increases, results of decisions

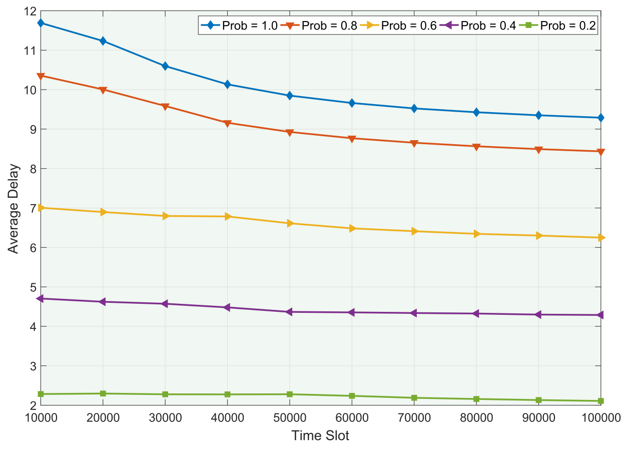


FIGURE 5. Runtime average delay composition.

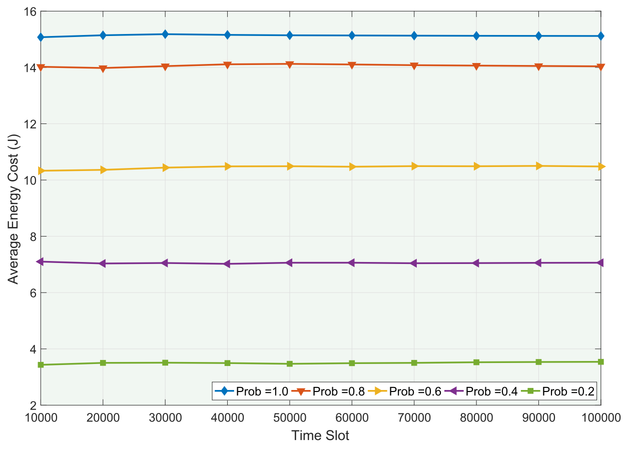


FIGURE 6. Runtime average energy cost composition.

made by the system are always worse, compared with a smaller system burden.

Figure 7 shows the proportion of user tasks completed within their time constraints. For this rate, it directly reflects users' satisfaction with the system, which is an extremely important indicator. For a user, request does not return within his expected time constraint means user's request is invalid, which is intolerable in many scenarios. On each curve, as time goes by, the rate of satisfaction of user requests increases significantly. Especially when *Prob* is small, at almost all times, user requests are satisfied. As *Prob* gets larger, satisfaction goes up as the system continues to make decisions, and all of curves reach over 98% after a certain period of time.

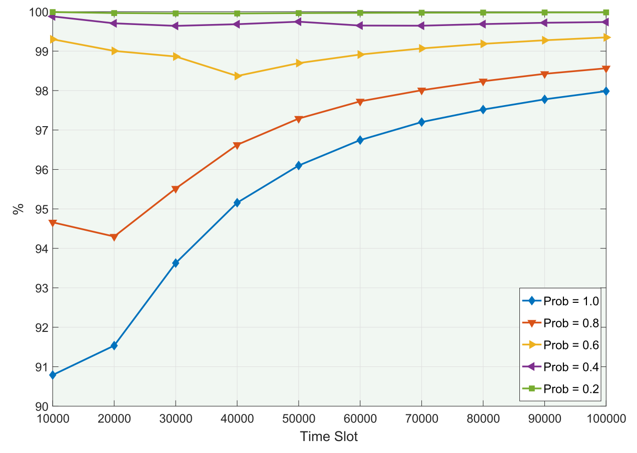


FIGURE 7. Tasks completion rate.

Through the above simulation results it can be seen that in the proposed reinforcement learning-based mechanism, with the growing of time, decision-making efficiency of the system is higher and higher. Average delay of the system is decreasing, especially after a certain time slot, almost all decisions made by system can make a task meet its time constraint.

2) IMPACT OF WEIGHTS AND TM

We then evaluate impact of weights and *Tm*. The weights of delay and energy cost in reward may cause the system to have different effects on the two factors when making decisions. For setting parameter *Tm*, we want to avoid poor decisions to the greatest extent. The difference between setting *Tm* to 1 and 10 mean that when *Tm* is equal to 10, reward value for decisions that might result in task timeouts increases significantly and is effectively eliminated. We also set $K1$ as 0.1, 0.5 and 0.9.

Figure 8 shows curves of average reward in different parameter combinations. Almost all curves show a downward trend as time slot increases. This trend is even more pronounced in all curves where *Tm* is equal to 10. Part of the reason is that there are fewer timeout decisions with larger rewards value, resulting in significant changes in average rewards. At the same time, the greater the weight of delay, the smaller the average reward of the system. It also indicates that the system decision has a greater impact on delay.

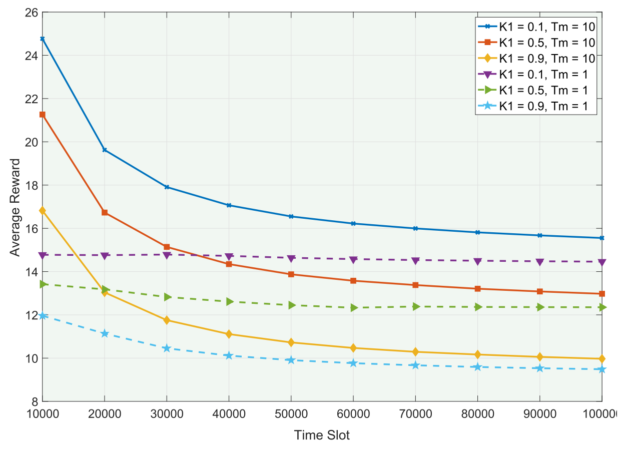


FIGURE 8. Runtime average reward composition.

In Figure 9, when $Tm = 10$, all curves are below the corresponding curve when $Tm = 1$, indicating that setting a large *Tm* does affect average delay of the system and significantly reduces delay caused by decisions. By observing the same curve group of *Tm*, we can know that the increase of the weight of delay makes the system more in the direction of decreasing delay when making decisions.

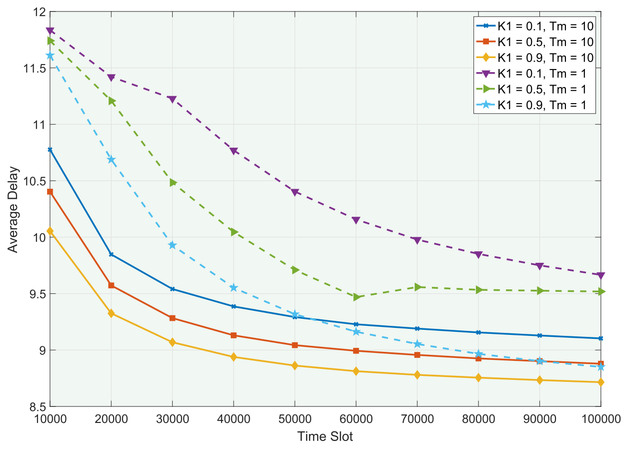


FIGURE 9. Runtime average delay composition.

In Figure 10, the curves of two groups of different *Tm* showed different trends. When $Tm = 1$, the change trend of curves basically conforms to that it rises first, then becomes stable and some curves even decline. When the weight of energy consumption becomes larger and larger, the average energy consumption curve of is lower than the other two curves and it decreases with the increase of time slot. When $Tm = 10$, curves show a relatively consistent downward trend. Meanwhile, as the weight of energy consumption increases, the corresponding curve does not show a gradually decreasing trend, but the overall error is within 0.3 J.

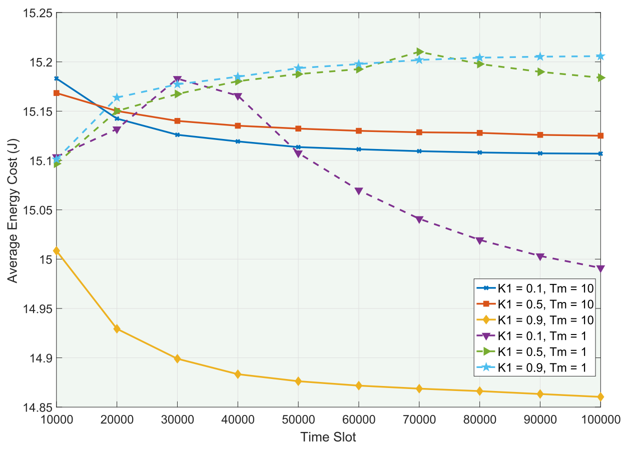


FIGURE 10. **Runtime average energy cost composition.**

As a factor that may directly affect user's request completion rate, in Figure 11, the curves group with larger Tm is significantly higher than the curves group with smaller Tm. In addition, in the set of curves with $Tm = 10$, the weight has little influence on task completion rate. In the other set of curves, different weights have a relatively large influence on completion rate, among which, when the weight of delay is larger, the completion rate of the curve is larger. At the same time, the commonality of these two groups of curves is that when $K1 = 0.9$, the completion rate is higher than the other two curves, while when $K1 = 0.5$ and 0.1, the completion rate of two curve is quite close.

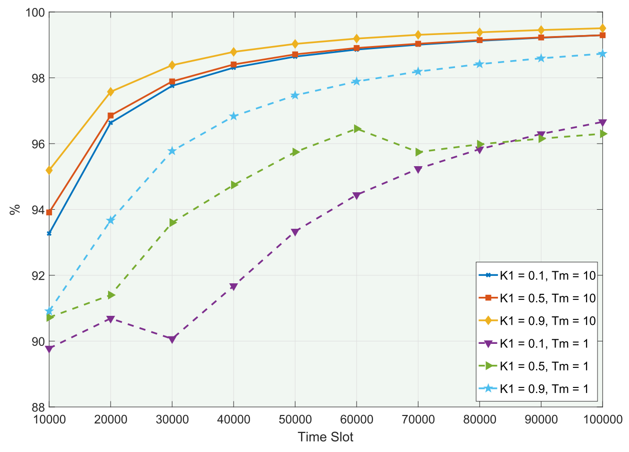


FIGURE 11. **Tasks completion rate.**

According to the results of this set of experiments, setting different weights and Tm values has a greater impact on system delay, but less impact on system energy consumption. At the same time, experimental results also show that by adjusting these parameters, the system decision can be made in the desired direction.

C. COMPARISON WITH OTHER MECHANISMS

We compared the proposed mechanism with Random strategy and Bottom-up game strategy mentioned in [35], which are represented by **Proposed**, **Random** and **Game**, respectively. We set that $K1 = 0.5$, $Tm = 1$ and it iterated 20000 times. In this setting, we explore the influence of different request arrival rate $Prob$ on those mechanisms by observing the system's long-term average return, delay, energy consumption, and whether user requests are completed within their time constraints.

Figure 12 shows the curves of average reward under different task arrival rate. Obviously, **Proposed** has advantages over other two mechanisms. With the increase of $Prob$, average return growth trend of **Proposed** slows down. When $Prob$ reaches 90 percents, its return remains stable. Average returns of the other two methods are similar. When $Prob$ increases, change trend of **Game** exceeds that of the **Random** method. When task arrival rate is low, the performance of each mechanism is similar, while it increases gradually, which reflects that the proposed mechanism can effectively adapt to the situation of heavy workload.

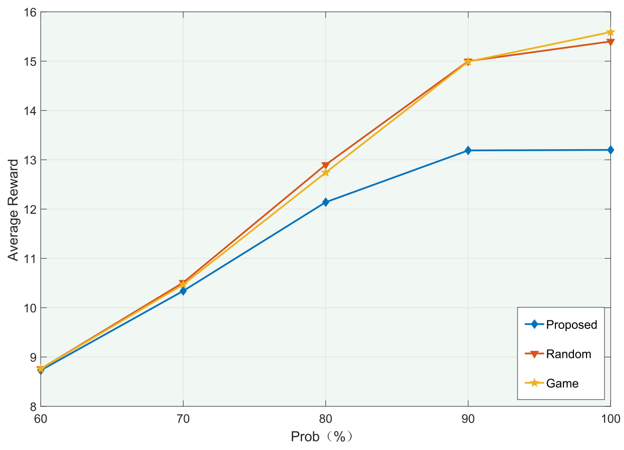


FIGURE 12. **Average reward composition.**

As an element of average return, Figure 13 and Figure 14 show the variation trend of delay and energy consumption respectively. Figure 13 shows a similar trend to Figure 12. The curves of **Game** and **Random** are close at different $Prob$,

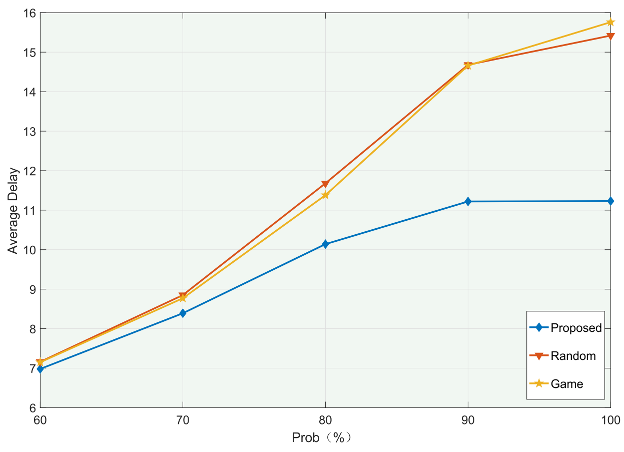


FIGURE 13. **Average delay composition.**

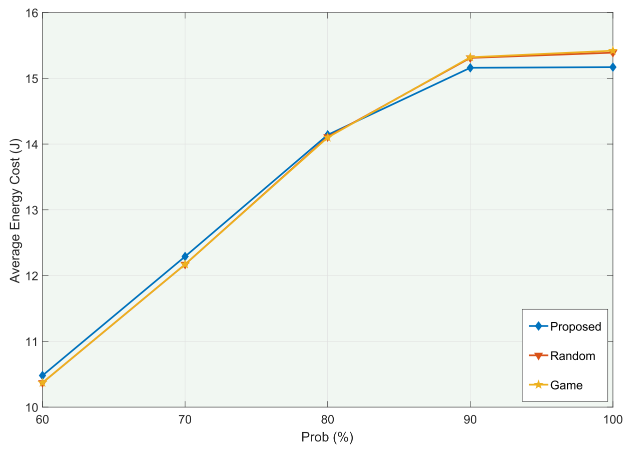


FIGURE 14. Average energy cost composition.

while delay of **Proposed** is lower as *Prob* increases. When *Prob* = 100, it means that each time slot will receive a new concurrent user request. While delay of **Proposed** remains stable, which indicates that the proposed mechanism can still make better decisions under long-term operation of the system with heavy load. In Figure 14, the variation trends of three curves are relatively similar and their values are close to each other, while **Proposed** curve is a little bit lower when *Prob* is close to 100 percents. This indicates that the main reason affecting the change of target function is delay. Because of many factors that affect delay, the delay caused by data transmission and processing can vary greatly in any system state when different actions are taken. Compared wit delay, the change of energy consumption is mainly affected by data size. So the performance of those mechanisms is similar in energy consumption variation. This also shows that the mechanism proposed in this paper can better adapt to task scheduling in a multi-factor environment.

Figure 15 shows tasks completion rate under different *Prob*. All three curves declined with the increase of task arrival rate. The decline rate of **Proposed** is obviously slower and it keeps slowing down. When *Prob* is greater than 70 percents, decline rate of curve **Game** and curve **Random** becomes faster and when *Prob* reaches 90 percents, there is a trend of slowing down. But they are still far below than that of **Proposed**. It can be seen that when workload of the system increases, the mechanism proposed in this paper can make better decisions, while other two mechanisms both show a significant decline, leading to a significant decline in efficiency after long-term running of the system, which seriously affects the quality of experience of users.

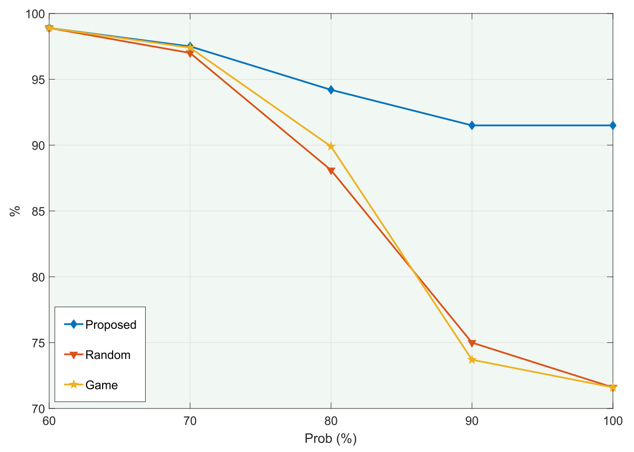


FIGURE 15. Tasks completion rate.

VII. CONCLUSION

This paper proposed an online task scheduling optimization mechanism based on reinforcement learning in the framework of edge server collaboration, in order to reduce the long-term delay and energy consumption of the system while satisfying internal temporal constraints of requests with complex structure. Specifically, an MDP is adopted to model the problem and TD learning with an average reward setting was applied to solve this problem. Due to the dimensionality curse caused by the large scale of system state and action set, a linear value function approximation is proposed, whose parameters are adjusted by gradient descent method. Extensive experiments are conducted, and the results show that, with the increase of the number of iterations, and the return of the system is increasing. Compared with other baseline algorithm and other optimal mechanism, our technique is promising in reducing long-term delay and energy consumption while achieving higher satisfaction rate of complex requests.

REFERENCES

[1] 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, Oct. 2016.

[2] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.

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

[4] T. Qiu, B. Li, X. Zhou, H. Song, I. Lee, and J. Lloret, "A novel shortcut addition algorithm with particle swarm for multisink Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 16, no. 5, pp. 3566–3577, May 2020.

[5] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, May 2016.

[6] 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, Oct. 2016.

[7] Y. Zhang, P. Du, J. Wang, T. Ba, R. Ding, and N. Xin, "Resource scheduling for delay minimization in multi-server cellular edge computing systems," *IEEE Access*, vol. 7, pp. 86265–86273, 2019.

[8] P. Pace, G. Aloi, R. Gravina, G. Caliciuri, G. Fortino, and A. Liotta, "An edge-based architecture to support efficient applications for healthcare industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 15, no. 1, pp. 481–489, Jan. 2019.

[9] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, and X. Shen, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11049–11061, Nov. 2018.

[10] S. Josilo and G. Dan, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 85–97, Feb. 2019.

[11] M. Sun, Z. Zhou, J. Wang, C. Du, and W. Gaaloul, "Energy-efficient IoT service composition for concurrent timed applications," *Future Gener. Comput. Syst.*, vol. 100, pp. 1017–1030, Nov. 2019.

[12] X. Xue, H. Han, S. Wang, and C.-Z. Qin, "Computational experiment-based evaluation on context-aware O2O service recommendation," *IEEE Trans. Services Comput.*, vol. 12, no. 6, pp. 910–924, Nov. 2019.

[13] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 708–719, Jul. 2018.

[14] F. Wang, H. Xing, and J. Xu, "Optimal resource allocation for wireless powered mobile edge computing with dynamic task arrivals," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[15] T. Qiu, B. Li, W. Qu, E. Ahmed, and X. Wang, "TOSG: A topology optimization scheme with global small world for industrial heterogeneous Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3174–3184, Jun. 2019.

[16] C. Yi, J. Cai, and Z. Su, "A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications," *IEEE Trans. Mobile Comput.*, vol. 19, no. 1, pp. 29–43, Jan. 2020.

[17] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.

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

[19] W. Zhang, H. Flores, and P. Hui, "Towards collaborative multi-device computing," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2018, pp. 22–27.

[20] G. Calice, A. Mtibaa, R. Beraldi, and H. Alnuweiri, "Mobile-to-mobile opportunistic task splitting and offloading," in *Proc. IEEE 11th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2015, pp. 565–572.

[21] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2014, pp. 352–357.

[22] G. Shu, X. Zheng, H. Xu, and J. Li, "Cloudlet-assisted heuristic offloading for mobile interactive applications," in *Proc. 5th IEEE Int. Conf. Mobile Cloud Comput., Services, Eng. (MobileCloud)*, Apr. 2017, pp. 66–73.

[23] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

[24] L. Chen, P. Zhou, L. Gao, and J. Xu, "Adaptive fog configuration for the industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4656–4664, Oct. 2018.

[25] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6.

[26] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, early access, Jul. 24, 2019, doi: 10.1109/TMC.2019.2928811.

[27] F. Zhang, J. Ge, C. Wong, C. Li, X. Chen, S. Zhang, B. Luo, H. Zhang, and V. Chang, "Online learning offloading framework for heterogeneous mobile edge computing system," *J. Parallel Distrib. Comput.*, vol. 128, pp. 167–183, Jun. 2019.

[28] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[29] L. Lei, H. Xu, X. Xiong, K. Zheng, and W. Xiang, "Joint computation offloading and multiuser scheduling using approximate dynamic programming in NB-IoT edge computing system," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5345–5362, Jun. 2019.

[30] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-Q-network-based multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 39974–39982, 2019.

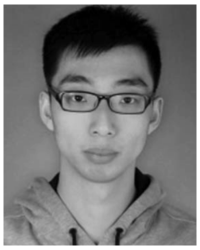
[31] Y. Zhang, L. Meng, X. Xue, Z. Zhou, and H. Tomiyama, "QoE-constrained concurrent request optimization through collaboration of edge servers," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9951–9962, Dec. 2019.

[32] T. Qiu, J. Liu, W. Si, and D. O. Wu, "Robustness optimization scheme with multi-population co-evolution for scale-free wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1028–1042, Jun. 2019.

[33] N. Chen, T. Qiu, X. Zhou, K. Li, and M. Atiquzzaman, "An intelligent robust networking mechanism for the Internet of Things," *IEEE Commun. Mag.*, vol. 57, no. 11, pp. 91–95, Nov. 2019.

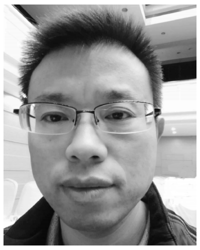
[34] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[35] D. Y. Zhang and D. Wang, "An integrated top-down and bottom-up task allocation approach in social sensing based edge computing systems," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 766–774.



YAQIANG ZHANG is currently pursuing the Ph.D. degree with the School of Information Engineering, China University of Geosciences (Beijing). His research interests include the Internet of Things and edge computing.

ZHANGBING ZHOU is currently a Professor with the China University of Geosciences (Beijing), China, and an Adjunct Professor with TELECOM SudParis, France. His research interests include the Internet of Things, services computing, and business process management.



ZHENSHENG SHI is currently a Senior Engineer with the Research Institute of Petroleum Exploration and Development, CNPC, China. His research interests include the Internet of Things, sedimentology, sequence stratigraphy, and reservoir geology.

LIN MENG is currently an Associate Professor with the Graduate School of Science and Engineering, Ritsumeikan University, Japan. His research interests include the IoT, high-performance computing, and image processing.

ZHENJIANG ZHANG is currently a Professor with the School of Software Engineering, Beijing Jiaotong University, China. His research interests include wireless sensor network techniques and wearable sensor network techniques.