Online Joint Scheduling of Delay-Sensitive and Computation-Oriented Tasks in Edge Computing

Fuming Zhang*†, Zhiqing Tang*†, Jiong Lou*†, Weijia Jia†*
*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
†State Key Lab of Internet of Things for Smart City, FST, University of Macau, SAR Macau 999078, China
{zhangfuming-alex, domain, lj1994}@sjtu.edu.cn, jiawj@um.edu.mo

***Abstract*—In the context of Edge Computing (EC) and Internet of Things (IoT), numerous tasks are offloaded from mobile users and sensor devices to edge nodes for further processing to reduce delay and solve the problem of insufficient local computation resources. These tasks can be mainly divided into delay-sensitive and computation-oriented tasks. The former tasks depend on the service provided by the container, while the latter tasks are submitted as a batch with task dependencies. Considering the heterogeneity of edge nodes, joint task scheduling can effectively improve resource utilization. However, relatively few researches consider the different characteristics of tasks like container constraints and task dependencies in joint task scheduling in EC. In order to fill in this gap, we propose a deep deterministic policy gradient (DDPG) based online joint task scheduling (OJTS) algorithm. Specifically, 1) We first model the problem of joint scheduling of delay-sensitive and computation-oriented tasks in resource-constrained EC scenario with the goals of maximizing system utility and minimizing system cost (weighted sum of the number and duration of unfinished tasks). 2) Then, we propose a deep reinforcement learning (DRL) algorithm to solve the above problem and make appropriate adjustments to the original network structure according to the scheduling decision. 3) Through validation on real-world trace, OJTS can improve the system utility by 26.0% and overall reward by 51.2% compared with baselines and meet real-time decision-making requirements.**

I. Introduction

The development of Internet of Things (IoT) and Edge Computing (EC) technology and the advent of 5G era have brought broader application prospects for the industry, transportation, security, and other fields. Compared with traditional cloud computing, computing nodes are deployed on the edge of the network in EC, which is closer to mobile users and sensor devices [1]. Mobile devices can offload tasks that are limited by insufficient computation resources or battery life to edge nodes to reduce the delay and transmission cost significantly.

In EC, the tasks submitted by mobile users and sensor devices are mainly divided into delay-sensitive and computation-oriented tasks [2]. As for delay-sensitive tasks, they have strict requirements for low delay, such as virtual reality (VR) [3] proposed by mobile users and object tracking proposed by sensor devices like surveillance cameras [4]. These tasks are offloaded to the nearby edge node in the form of service requests, which are processed by a limited number of running container instances of different types on the node.

In terms of computation-oriented tasks, they are mainly multi-modal tasks, such as behavior navigation [5] and trace-ability analysis [6], which are delay-tolerant and consume plenty of computation resources when processing and analyzing various types of sensor data. Unlike the delay-sensitive tasks, these tasks are submitted in the batch form and described using the "job-task-instance" model, i.e., a batch of tasks is considered as a job and each task consists of multiple instances. Besides, there may be dependencies between tasks, which means that the subsequent task can be executed only when all instances of the previous task have been completed.

Joint scheduling of these two tasks, i.e., processing as many computation-oriented tasks as possible while ensuring delay-sensitive tasks are completed within the expected time, can effectively improve the resource utilization of the system. Therefore, such a joint scheduling algorithm is urgently needed. However, to the best of our knowledge, there are relatively few researches on joint task scheduling in EC. Li et al. [7] focus on joint optimization of data placement and task scheduling and aim to enhance the user experience. However, more attention is paid to data block management without distinguishing tasks with different characteristics. Mao et al. [8] propose the joint management of radio and computation resources to minimize energy consumption in mobile edge computing scenarios. A tradeoff between delay and energy consumption is made, but the delay-tolerant tasks are not considered. Shah-Mansouri et al. [9] study a joint optimal price and task scheduling in mobile cloud computing. Considering utility maximization, an optimal scheduling strategy for delay-sensitive and delay-tolerant applications is provided, but the core perspective is profit maximization through dynamic pricing. In addition, the container constraints (limitations on the type and number of container instances) and task dependencies are not considered in all task models of these related works.

In order to fill this vacancy, the online joint task scheduling (OJTS) algorithm based on deep reinforcement learning (DRL) is proposed to solve the problem, aiming to fulfill the different demands of tasks, maximize the utility and minimize the cost of the system. Our algorithm is based on the deep deterministic policy gradient (DDPG) [10] method with stable performance and fast convergence speed. According to the discrete action space in our case, its original architecture is adjusted appropriately. Specifically, the activation function used in the last layer of the policy network is changed to Softmax and the candidate action is obtained by sampling the probability distribution output. In addition, to verify the effectiveness of

978-1-7281-5212-7/19/$31.00 ©2019 IEEE
DOI 10.1109/MSN48538.2019.00064

our algorithm, we implement an EC simulation environment based on SimPy. Compared with the baseline algorithms, our OJTS algorithm achieves 26.0% system utility improvement and 51.2% overall reward improvement and enables real-time decision-making by validating on a real-world data set [11].

To summarize, we propose an online joint task scheduling algorithm for delay-sensitive and computation-oriented tasks based on DRL. Main contributions of this paper are as follows:

1) We are the first to jointly schedule delay-sensitive tasks and computation-oriented tasks with container constraints and task dependencies in EC. The heterogeneity of edge nodes, limitations on the type and the maximum number of container instances that each node can run are considered.
2) A DDPG-based algorithm is adopted to solve the joint scheduling problem. The activation function of the last layer of the policy network is modified to fit the discrete action space.
3) To add container constraints and task dependencies to the experiment, an EC simulation environment is implemented. Compared with the traditional heuristic algorithm, the proposed OJTS algorithm achieves a 26.0% improvement in system utility by verifying on real-world data set.

The remainder of this paper is organized as follows. In Section II, we introduce the system model and formulate the joint task scheduling problem. The DDPG-based online joint task scheduling algorithm is proposed in Section III. Then the simulation platform, experimental data, settings, and results are illustrated in Section IV. Finally, we conclude the paper and introduce future work in Section V.

II. Modeling and Problem Formulation

In this section, we first introduce the system model of EC in II-A. Then the problem of joint scheduling of delay-sensitive and computation-oriented tasks is formulated in II-B.

A. System Model

Under the general EC architecture, we mainly focus on the interaction between edge nodes N, mobile and sensor devices D. In this scenario, edge nodes accelerate the processing of task requests that are offloaded due to insufficient computation resources of mobile and sensor devices. To approach the real situation, the heterogeneity of nodes is fully considered, i.e., the computation resources of each node are different. For each edge node $n_i \in N$, CPU, memory, and image capacity are the primary resources to be considered. The unit of CPU resource is the core number, while the memory resource is normalized. The image is a lightweight executable package of software that includes everything needed to run a container.

Two types of tasks, delay-sensitive and computation-oriented tasks, are processed on edge nodes. A delay-sensitive task a_j is handled by containers running on edge nodes in the form of service request, and these running containers depend on the image resource. As shown in Table I, four delay-sensitive tasks are considered [12]. Accordingly, there are four types of image resources. The 'Resp' column in the table represents the expected response time of the task, and the task completed within that time will receive the highest utility score. Besides, 'Dur' and 'Vol' represent task duration and data size, respectively. It is assumed that every single image resource can support only one container instance of the corresponding type, while each container instance can handle multiple service requests simultaneously. However, as the number of service requests processed simultaneously increases, the resources consumed to process each request will increase as shown in Table II. When the number of requests exceeds 10, the container instance is overloaded, consuming twice the standard computation resources.

TABLE I
Delay-sensitive Task Specifications

Type	CPU	Mem	Resp(s)	Dur(s)	Vol(MB)
Face Recognition	3	2	3	0.5 - 1	0.2 - 0.6
Object Detection	3	2	5	1 - 2	0.5 - 1.5
Image Processing	5	4	10	2 - 4	2 - 5
Basic Search	1	2	2	0.5 - 1.2	0.1 - 0.2

TABLE II
Container Load and Resource Consumption

Load Status	Number of Requests	Resource Consumption Factor
Idle	0	0.5
Balance	1 - 5	1
High	6 - 10	1.5
Over	11 - ...	2

In terms of computation-oriented tasks, they are mostly multi-modal analysis tasks based on sensor data, e.g., the crowd analysis based on infrared, ambient light and channel state information (CSI). Unlike delay-sensitive tasks, computation-oriented tasks can be described using the "job-task-instance" model and each batch of tasks b_k is considered as a job. In such a model, the dependencies between tasks are described by the Directed Acyclic Graph (DAG). Each task has a certain number of instances, and only when all its instances have been executed can subsequent tasks begin to execute.

B. Problem Formulation

In this paper, our goal is to make full use of the computation resources of the resource-constrained EC system in order to ensure the completion of delay-sensitive tasks while completing computation-oriented tasks as many as possible, i.e., maximize the utility and minimize the cost.

In terms of system utility, each delay-sensitive task, based on the time it takes from submission to completion, is assigned with different utility evaluation a_j^u as follows:

$$a_j^u = \begin{cases} 2, & a_j^e - a_j^s \le a_j^r \\ 1, & a_j^r < a_j^e - a_j^s \le 2 \times a_j^r \\ -1, & a_j^e - a_j^s > 2 \times a_j^r \end{cases},$$

where a_j^e and a_j^s represent the completion time and submission time of a service request, respectively. a_j^r represents the

expected response time, which is shown in Table I. The shorter the task completion time, the higher the utility reward, and the task that exceeds the acceptable time limit is punished with a negative utility penalty.

As for computation-oriented tasks, when a job b_k is completed, its utility b_k^u is evaluated as follows:

$$b_k^u = \begin{cases} 300, & b_k^e - b_k^s \leq 300 \\ 0, & b_l^e - b_k^s > 300 \end{cases},$$

where b_k^e and b_k^s indicate the completion time and submission time of a job, respectively. Considering the number of delay-sensitive tasks is far more than that of computation-oriented tasks in the same period of time, we appropriately improve the utility evaluation of the latter tasks to balance the utility gap. Thus the system utility $U(t)$ at time t is defined as:

$$U(t) = \sum_{\{a_j | t-1 < a_j^e \leq t\}} a_j^u + \sum_{\{b_k | t-1 < b_k^e \leq t\}} b_k^u.$$

Regarding system cost $C(t)$, the weighted sum of the number and duration of currently unfinished delay-sensitive and computation-oriented tasks are calculated as follows:

$$C(t) = \frac{1}{2}\left[\frac{1}{300}\sum_{\{a_j | a_j^e > t\}}(1 + \frac{1}{a_j^d}) + \sum_{\{b_k | b_k^e > t\}}(1 + \frac{1}{b_k^d})\right],$$

where a_j^d and b_k^d indicate the duration of a service request or a job, separately. For these unfinished tasks, some of them will be accomplished on time with subsequent processing, while others will need to be offloaded to the cloud to complete on time, resulting in additional system cost. They are treated as system cost together since they cannot be distinguished at this point. When counting the duration of these unfinished tasks, the inverse of them is used. For normalization purposes, the cost of delay-sensitive tasks is divided by 300. When calculating the overall cost of the system, the sum of the cost of these tasks is divided by 2.

In summary, the joint task scheduling problem in EC can be defined as follows:

$$\max R = \sum_{t=0}^{T} U(t) - C(t), \tag{1}$$
$$s.t. \quad \alpha \times a_j.c \leq n_i.c, \quad \alpha \times a_j.m \leq n_i.m$$
$$b_k.c \leq n_i.c, \quad b_k.m \leq n_i.m \quad ,$$

where $a_j.c$ and $b_k.c$ represent CPU resource requirements for delay-sensitive and computation-oriented tasks, respectively. Correspondingly, $a_j.m$ and $b_k.m$ denote the memory resource requirements of tasks. Besides, $n_i.c$ and $n_i.m$ indicate the available CPU and memory resources of the node. α represents the resource consumption factor which depends on the number of service requests being processed by a specific type of container on the node as illustrated in Table II.

We aim to maximize the overall reward of the system, i.e., the difference between utility and cost while meeting the constraints. Eq. (1) expresses an advanced online bin-packing problem, which is NP-hard and usually relies on heuristic algorithms to solve it. However, the current heuristic algorithms are not stable in the EC scenario; on the other hand, they are time-consuming and difficult to meet the real-time decision-making requirements. Considering the memoryless property of the arrival tasks [13], we can adopt the method of DRL to solve the above problem.

III. Online Joint Task Scheduling Algorithm

In this section, we first briefly introduce reinforcement learning (RL) and the DDPG algorithm in III-A. Then our online joint task scheduling algorithm is presented in III-B.

A. Reinforcement Learning

In an RL algorithm, the system is mainly composed of environment and agent. At time t, the agent μ observes the environment to obtain the current state S_t and takes action A_t according to a particular strategy. Then the environment changes from state S_t to S_{t+1}, and the agent is rewarded with R_t based on the previous action. The above process will be repeated, and the agent needs to gradually learn to take appropriate actions in such a process, so as to maximize the expected reward $E[\sum_{t=1}^{T} R_t]$.

Traditional RL cannot deal with complex state and action space with high dimensions, so as a combination of deep learning (DL) and RL, DRL has emerged. Among many DRL algorithms, Deep Q Network (DQN) has the defect of overestimation, while the Actor-Critic (AC) method has the problem of slow convergence. To solve the above problems, DDPG algorithm is proposed [10]. Its deterministic policy can improve the efficiency and the combination with DQN makes the training process easy to converge. Therefore, our online joint task scheduling algorithm is based on DDPG. In DDPG, it mainly uses two parts: the Q network which maps state to action as an actor, and the deterministic policy network which gives a score to the aforementioned action as a critic. To increase the stability of the training process, the target network is proposed in [14].

B. Algorithm Design

Next, we will introduce the state space, action space, reward design, network structure, and online joint task scheduling algorithm in the following subsections.

1) State: The state is composed of the status of edge nodes and the information of the candidate task c_t. The candidate task is selected based on the strategy of choosing the first task from the pending task queue P that can be accommodated to any node. For each edge node, the current available CPU $n_i.c$, memory $n_i.m$, the number of delay-sensitive and computation-oriented tasks in progress $n_i.a$ and $n_i.b$ are extracted as its feature. For the candidate task, CPU $a_j.c$ ($b_k.c$), memory $a_j.m$ ($b_k.m$), and duration $a_j.d$ ($b_k.d$) are selected as features. In short, the system state at time t can be defined as:

$$S_t = (n_1.c, ..., n_{|N|}.b, a_j.c(b_k.c), ..., a_j.d(b_k.d)) \in \mathbb{S},$$

where $\mathbb{S}$ is the set of all states.

TABLE III
NETWORK STRUCTURE

Network	Layer	Shape	Activation Function
Q	L1	state_dim, 16	ReLu
	L2	16 + action_dim, 4	ReLu
	L3	4, 1	None
Policy	L1	state_dim, 16	ReLu
	L2	16 + action_dim, 4	ReLu
	L3	4, 1	None

2) Action: In our system implementation, the scheduler makes decisions at fixed scheduling intervals, and appropriate tasks are selected from the pending task queue and assigned to some edge node for processing according to the selected heuristic algorithm or the agent in DRL. Assuming that there are M pending tasks in the queue and $|N|$ available nodes at the current time, the number of available actions is $(|N|+1)^M$, where 1 represents a task is not scheduled to any node. Such a large action space will bring great difficulties to our training process. To reduce the action space, the agent is allowed to make multiple decisions during a single scheduling interval t_s, i.e., the system first selects a candidate task, and then the agent μ decides to assign it to one of the $|N|$ nodes. The action A_t at time t is defined as

$$A_t \in \{n_1, n_2, \ldots, n_{|N|}\} = \mathbb{A},$$

where $\mathbb{A}$ is the set of all actions and the action space is $|N|$.

3) Reward: The reward is set as a pre-estimate of the utility described in II-B. Specifically, when a delay-sensitive task is scheduled, we estimate its completion time and give it a utility reward of 2, 1, or - 1. When a computation-oriented task is scheduled, it is rewarded with 0.1 since it is difficult to assess whether the expected completion time is within 300 seconds or not based on complex task dependencies. Besides, if the candidate node c_n cannot accommodate the candidate task, a utility penalty of -1 is imposed.

4) Network Structure: The structure of the Q network Q and policy network μ is shown in Table III. Correspondingly, the structures of the target networks Q' and μ' are the same as the above two. The system state consists of node status and candidate task information, so state_dim equals to $(|N|+1)\times 4$, where $|N|$ and 1 represent the number of nodes and candidate tasks respectively, and 4 is the feature dimension. For the task feature, it has one dimension less than the node feature, which is padded with 0. As for the action dimension, it is set to the number of nodes $|N|$. In order to adapt to our discrete action space, the activation function of the last layer of the policy network is changed from Tanh to Softmax. This can avoid that when the original outputs of the last layer of the network are negative, they will become zero after the non-linear transformation, which makes it impossible for us to get the index of the candidate node according to the probability distribution sampling later.

5) Online Joint Task Scheduling Algorithm: The proposed online joint task scheduling (OJTS) algorithm based on DDPG is shown in Algorithm 1. As shown from line 3 to 25, the scheduling process is awakened at certain intervals and continuously schedule tasks until there is no suitable candidate node or task. Specifically, in line 6, the candidate task is selected based on the strategy of choosing the first task from the pending task queue that can be accommodated to any node. In line 7, the agent selects an action node based on current state. Next, from line 8 to 17, it is determined whether the action node selected by the agent can accommodate the candidate task. If so, $c_n = A_t$. Otherwise, the first node that can accommodate the candidate task is chosen as the candidate node. After the scheduling operation, in line 21, the current state, action, calculated reward and observed next state are pushed into the replay buffer B, for subsequent network training.

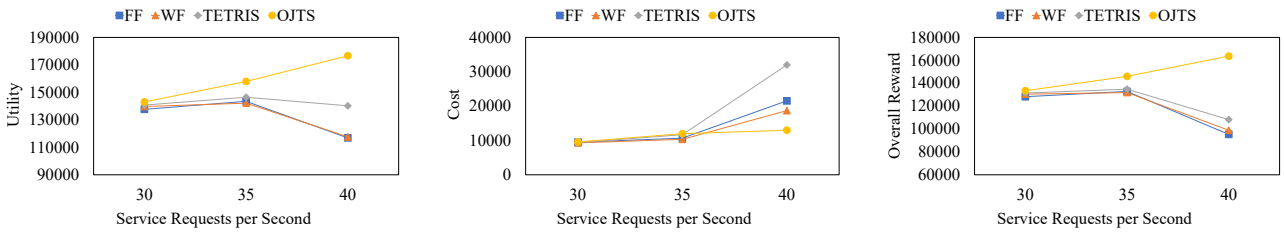
From line 26 to 28, the parameters of the neural network θ are updated at each training interval t_t. The main process is as follows. First, we sample a random minibatch with K transitions (S_i, A_i, R_i, S_{i+1}) from B. The updated Q value is set to

$$Y_i = R_i + \gamma Q'(S_{i+1}, \mu'(S_{i+1}|\theta^{\mu'})|\theta^{Q'}), \quad (2)$$

Algorithm 1 Online Joint Task Scheduling

Input: μ, N, P, t_s, t_t

1: Initialize replay buffer B
2: Set $t_c = 0$
3: **while** $p \neq \emptyset$ **do**
4: $t_c = t_c + t_s$
5: **while** True **do**
6: Select c_t from P and set $c_n = \emptyset$
7: Observe S_t and get $A_t = \mu(S_t)$
8: **if** A_t accommodate c_t **then**
9: $c_n = A_t$
10: **else**
11: **for** n_i in N **do**
12: **if** n_i accommodate c_t **then**
13: $c_n = n_i$
14: **break**
15: **end if**
16: **end for**
17: **end if**
18: **if** $c_n \neq \emptyset$ and $c_t \neq \emptyset$ **then**
19: Schedule c_t to c_n
20: Calculate R_t and observe S_{t+1}
21: Push (S_t, A_t, R_t, S_{t+1}) to B
22: **else**
23: **break**
24: **end if**
25: **end while**
26: **if** $t_c \ \% \ t_t = 0$ **then**
27: Update network parameters according to Eq. (2) - (5)
28: **end if**
29: Sleep for t_s
30: **end while**
31: **end**



(a) Utility (b) Cost (c) Reward

Fig. 1. Performance of Different Methods

where γ is a discount coefficient.

Next, the Q network is updated by minimizing the loss:

$$L = \frac{1}{K}\sum_{i}(Y_i - Q(S_i, A_i|\theta^Q))^2. \tag{3}$$

After that, we update the policy network using the sampled policy gradient

$$\nabla_{\theta^\mu} J \approx \frac{1}{K}\sum_{i} \nabla_A Q(S, A|\theta^Q)|_{S=S_i, A=\mu(S_i)} \nabla_{\theta^\mu}\mu(S|\theta^\mu)|_{S_i}. \tag{4}$$

Finally, the target Q network and target policy network are updated according to the following equations

$$\theta^{Q'} = \tau\theta^Q + (1-\tau)\theta^{Q'}, \theta^{\mu'} = \tau\theta^\mu + (1-\tau)\theta^{\mu'}, \tag{5}$$

where τ is another discount parameter used to smooth the learning process.

IV. Performance Evaluation

In this section, we first introduce our simulation environment in IV-A, then describe the experimental data we use in IV-B. Next, we clarify the parameter settings in IV-C, and finally show and analyze the experimental results in IV-D.

A. Simulation Environment

Based on Python, the SimPy module, and the work of [15], we implement an EC simulation environment. The core components of this environment include cluster, broker, scheduler, and monitor. In each simulation, a cluster consists of several nodes with different computation resources. Two different brokers are used to submit the arrival delay-sensitive and computation-oriented tasks to the cluster. Besides, a universal joint scheduler is designed for these two different types of tasks. Different scheduling algorithms can be used by configuring the scheduler. The Monitor counts and records the status of the simulation environment according to a fixed time interval, so as to facilitate us to view and analyze the real-time information of the system.

B. Experimental Data

In the experiment, we use the real-world cluster trace from Alibaba [11]. The cluster_trace_v2018 is used, which includes records of about 4000 servers in a period of 8 days. The computation-oriented tasks are mainly extracted from the batch workload records in the data set, including the job name, task name, task type, instance number, CPU, memory, and duration. The DAG dependencies are obtained from task names and the duration is obtained by subtracting the end time from the start time in the task record. Besides, in order to make the data more consistent with the EC scenario, the upper limit of the number of instances per task is set to 20 and the duration is set to 60 seconds.

Although the above data set contains container-type data records, they are not detailed enough to meet our requirements. Therefore, we generate the data based on delay-sensitive task specifications. The duration and volume are randomly generated within the range as shown in Table I, and the minimum partition interval of the values is 0.1.

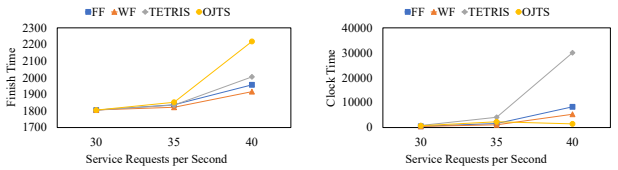
C. Parameter Settings

The duration of the experiment is set to 1800 seconds and the cluster has five nodes. Besides, a total of 120 computation-oriented tasks need to be processed. About two tasks of this type are submitted every 60 seconds. Regarding the delay-sensitive task, the number of service requests submitted to the cluster every second is varied to observe the performance of the algorithm under different task loads. In the scheduling process, the task scheduling interval is set to 0.2 seconds. Considering the long duration of computation-oriented tasks, we schedule these tasks once every five scheduling intervals. In addition, the parameters of our OJTS algorithm are set as follows: $\gamma = 0.99, \tau = 1e-2$ and the learning rates of Q network and policy network equal to 1e-3.

D. Experimental Result

In order to demonstrate the efficiency of the proposed algorithm, we compare our OJTS algorithm with three heuristic baseline algorithms, first-fit (FF), worst-fit (WF) and Tetris [16]. A brief introduction to them is as follows:

- **FF**: As the name implies, FF is a straightforward greedy approximation algorithm. It processes the tasks in arbitrary order and attempts to assign the task to the first node which can accommodate the task.
- **WF**: As for WF, it also processes tasks in arbitrary order and assigns them to the node with the most available resources.
- **Tetris**: Tetris maps task requirements and node available resources into Euclidean space and chooses the task node pair with the largest dot product as the decision result.



(a) Finish Time (b) Clock Time

Fig. 2. Time Consumption of Different Methods

This strategy tends to select tasks with high resource usage or tasks whose resource requirements are proportional to available node resources.

Fig. 1 represents the performance of different methods under different task loads. The system utility of these methods is shown in Fig. 1(a). It can be seen that the proposed OJTS algorithm can improve the system utility, especially when the number of requests per second is 40, compared with Tetris, our algorithm can achieve 26.0% improvement. In addition, with the increase of task load, the computation resources of the system are not enough to support all tasks, so that the system utility of the baseline algorithm decreases under the penalty of utility. From the system cost of the different methods shown in Fig. 1(b), under high workload, the system cost of the OJTS algorithm is only 69.5% of that of the WF method. Fig. 1(c) shows the overall reward of the system, which is the difference between utility and cost. The OJTS algorithm has always been ahead of other algorithms under different task loads, and when the number of requests per second equals to 40, the leading margin reaches the maximum of 51.2%.

As shown in Fig. 2, our algorithm also performs well in time consumption. Fig. 2(a) represents the completion time of the last task in the simulation process. Although FF, WF and Tetris excel in this respect, the low utility and high cost shown in Fig. 1 illustrate they fail to prioritize those delay-sensitive tasks. Fig. 2(b) shows that heuristic algorithms consume much time in the execution process, even far beyond the time used in the simulation process, especially in the case of high task load. However, the clock time (including the model training time) consumed by the OJTS algorithm is less than the simulation time, which means it meets the real-time decision-making requirements in EC.

In short, OJTS has an overall best performance than all baselines and outperforms the baseline approaches up to 26.0% on system utility and 51.2% on overall reward and meet the needs of real-time decision-making.

V. Conclusion and Future Work

In this paper, we first model the problem of joint scheduling of delay-sensitive and computation-oriented tasks in resource-constrained EC scenario. Then, to improve system utility and reduce system cost, we propose the OJTS algorithm based on DDPG. Finally, the validation on real-world data-trace shows that our algorithm improves by 26.0% and 51.2% on system utility and overall reward compared with baselines. Besides, it dramatically reduces the computation cost to meet real-time decision-making requirements. As for future work, we consider taking certain strategies to adjust the containers running on edge nodes according to the changes of service requests, so as to provide more efficient scheduling decisions.

Acknowledgment

This work is supported by State Key Lab of Internet of Things for Smart City, University of Macau; Chinese National Research Fund (NSFC) Key Project No. 61532013 and No. 61872239; 0007/2018/A1, 0060/2019/A1, DCT-MoST Joint-project No. 025/2015/AMJ of Science and Technology Development Fund, Macao S.A.R (FDCT), China, and University of Macau Grant Nos: MYRG2018-00237-RTO, CPG2019-00004-FST and SRG2018-00111-FST.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Mobile edge computing and networking for green and low-latency internet of things," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 39–45, 2018.

[3] 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.

[4] A. Basharat, A. Gritai, and M. Shah, "Learning object motion patterns for anomaly detection and improved object detection," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008, pp. 1–8.

[5] M. Sakamoto, T. Nakajima, and S. Akioka, "A methodology for gamifying smart cities: Navigating human behavior and attitude," in *International Conference on Distributed, Ambient, and Pervasive Interactions*. Springer, 2014, pp. 593–604.

[6] H. Mora-Mora, V. Gilart-Iglesias, D. Gil, and A. Sirvent-Llamas, "A computational architecture based on rfid sensors for traceability in smart cities," *Sensors*, vol. 15, no. 6, pp. 13 591–13 626, 2015.

[7] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *Journal of Parallel and Distributed Computing*, vol. 125, pp. 93–105, 2019.

[8] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[9] H. Shah-Mansouri, V. W. Wong, and R. Schober, "Joint optimal pricing and task scheduling in mobile cloud computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 5218–5232, 2017.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[11] Alibaba. (2019) Alibaba cluster trace program. [Online]. Available: https://github.com/alibaba/clusterdata

[12] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[13] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, 2018.

[14] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[15] F. Li and B. Hu, "Deepjs: Job scheduling based on deep reinforcement learning in cloud data center," in *Proceedings of the 2019 4th International Conference on Big Data and Computing*. ACM, 2019, pp. 48–53.

[16] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 455–466.