Reinforcement Learning Based Offloading for Realtime Applications in Mobile Edge Computing

Hui Huang*, Qiang Ye*, Hongwei Du†
*Faculty of Computer Science, Dalhousie University, Halifax, Canada
†Department of Computer Science and Technology
Harbin Institute of Technology (Shenzhen), Shenzhen, China
Email: {huihuang, qye}@cs.dal.ca, hongwei.du@ieee.org

***Abstract*—Energy consumption is one of the most important issues for mobile devices such as smartphones and laptops. For mobile devices that execute multiple computation-intensive or delay-sensitive applications simultaneously, Mobile Edge Computing (MEC) based offloading provides a promising solution to the energy problem. However, blindly offloading all tasks to MEC servers is not the best choice because transferring a simple task to a MEC server via wireless networks might consume more energy than processing the task locally. In addition, Dynamic Voltage and Frequency Scaling (DVFS) could be utilized to reduce the energy consumption associated with locally processed tasks by appropriately lowering CPU frequency. In this paper, we propose a realtime reinforcement learning based offloading scheme, RRLO, which is based on both MEC-based offloading and DVFS-based energy consumption reduction. Technically, RRLO jointly learns the optimal offloading policy and DVFS-based scheduling method. Depending on the workload and network condition, RRLO not only determines whether a task should be offloaded to a MEC server, but also selects the best DVFS method used to schedule local tasks. Our simulation results indicate that RRLO outperforms the existing MEC-based offloading schemes.**

***Index Terms*—Offloading, Mobile Edge Computing, Reinforcement Learning, Realtime Applications**

I. Introduction

With the development of wireless networks such as LTE/5G and WiFi, mobile devices have been playing a key role in personal and business computing. Consequently, mobile devices often need to execute a number of applications simultaneously. Compared with desktops or servers, mobile devices tend to have limited computation capability. In addition, since mobile devices are normally powered by built-in batteries, their lifetime depends on the computation intensity of the executed applications. When running computation-intensive or delay-sensitive applications such as online gaming and virtual reality [1], mobile devices may easily become the bottleneck, seriously affecting user experience.

To address this problem, a variety of different computation offloading schemes have been proposed [2][3][4]. These schemes take advantage of the availability of communication networks and selectively offload part of computation-intensive tasks to resource-rich servers in close proximity, which leads to a new computing paradigm, Mobile Edge Computing (MEC). With MEC, the computation-intensive or delay-sensitive applications could be completed on time and the lifetime of mobile devices could be significantly extended. Although MEC-based offloading is highly promising, whether a specific task should be offloaded remains to be a challenging problem. For example, offloading a task may lead to higher power consumption because transferring the task from a mobile device to an MEC server via wireless networks could consume a large amount of energy. In this scenario, running the task on the local mobile device is a better option. Another problem with the existing offloading schemes is their computation complexity. Most of the existing schemes attempt to determine which tasks to offload by solving a Mixed Integer Programming (MIP) problem, which itself is a computation-intensive task that consumes much energy. Over the past years, Reinforcement Learning (RL) based offloading has been proposed to solve the energy consumption problem with the traditional offloading schemes [5]. With this approach, mobile devices learn the best offloading policy by interacting with the system environment. Namely, instead of solving a complex integer programming problem, mobile devices adjust the offloading policy continuously in a trial-and-error manner till the best policy is found. Another advantage of RL-based offloading is that its model training phase does not require the prior knowledge, which is often unavailable in practice.

Another effective approach to extending the lifetime of mobile devices is Dynamic Voltage and Frequency Scaling (DVFS). DVFS is based on the observation that the Actual Execution Time (AET) of a task is often shorter than its Worst Case Execution Time (WCET) [6]. When the AET is shorter than the WCET, the time slack is utilized to reduce power consumption. Technically, with DVFS, when a mobile device does not have a full workload (i.e. the time slack exists), the frequency of the CPU is appropriately reduced so that energy consumption is seriously lowered while realtime tasks are completed on time. The most widely-used DVFS techniques are Cycle Conserving (CC), Look Ahead (LA), Dynamic Reclaiming Algorithm (DRA) and Aggressive Speed Reduction (AGR) [7]. Each of them could outperform others in certain scenarios. Huang $et\ al.$ proposed a RL-based DVFS scheme, which selects the most appropriate DVFS dynamically so that the total energy consumption is minimized [8].

In our research, we attempt to take advantage of both MEC-based offloading and RL-based DVFS in order to minimize the power consumption of mobile devices. Specifically, both the operation of task offloading and that of CPU frequency reduc-

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

tion are considered when a mobile device needs to execute a series of computation-intensive/delay-sensitive tasks. Technically, in this paper, we present a Realtime Reinforcement Learning based Offloading (RRLO) scheme to process realtime applications on mobile devices. Our goal is to optimize power consumption while satisfying the time requirements of realtime tasks. Specifically, the optimal offloading policy is learned via the feedback from the environment. At each decision point, RRLO scheduler checks three system states (system utilization, dynamic slack, and channel gain) before taking an action to affect the environment. At the end of the cycle, the energy consumed in the cycle is returned to the scheduler to indicate how efficient the selected action is. Overall, the major contributions of this paper can be summarized as follows:

- A new learning method, Double Q-learning with Intervention, is proposed. This method can effectively eliminate overestimation in the learning phase while satisfying a set of constraints of realtime applications.
- A new computation paradigm based on task offloading and frequency adjustment is devised to take advantage of both of these methods.
- A realtime learning-based offloading scheme that jointly optimizes task scheduling and offloading, RRLO, is proposed. Instead of executing all tasks locally, RRLO could offload part of computational tasks to MEC severs. For the tasks that are processed locally, the proper DVFS scheme is dynamically selected. Ultimately, the overall energy consumption of the system is minimized.

The rest of this paper is organized as follows. The related work on learning-based DVFS and learning-based offloading is described in Section II. In Section III, we present the models adopted in the system under investigation. The proposed realtime reinforcement learning based offloading scheme is discussed in Section IV. Our experimental results are summarized in Section V. In Section VI, we present our conclusions.

II. Related Work

In this section, we present the existing studies related to learning-based DVFS and learning-based offloading.

A. Learning-based DVFS

Generally, the energy efficiency of DVFS technique highly depends on the adaptability of varying system states. To reduce energy consumption, learning-based methods have been introduced to power management to learn how to switch DVFS dynamically to adapt to different states. There are extensive researches in the literature. Bhatti et al. presented an online learning-based power management approach, which combines Dynamic Voltage and Frequency (DVFS) with Dynamic Power Management(DPM) and learns the best strategy based on a sort of trained experts [9]. Islam and Lin proposed a reinforcement learning-based DVFS selection approach that integrates a set of DVFS technique and Q-learning to reduce system-level energy consumption [10]. In [11], Zhang et al. presented a deep reinforcement learning-based scheme to adaptively choose the proper DVFS technique. This scheme learns the best policy by using a Double-Q deep network, which can eliminate the overestimation and facilitate the convergence in the training period. However, as the expansion demand on computation and more restricted time constraints in mobile devices, it is imperative to offload some of computational tasks to the edge servers.

B. Learning-based Offloading

Recently, many learning-based offloading approaches have been proposed for energy optimization in edge computing. Li et al. proposed a reinforcement learning offloading approach to jointly optimize the offloading decision and resource allocation strategy. To be specific, a combination of decisions that consist of CPU frequency and offloading policy, are made based on local system usage and the remaining computation capacity in edge servers [12]. Liu et al. proposed a RL-based vehicle-assisted offloading approach that utilizes both Q-learning and Deep Reinforcement Learning (DRL) to approximate the global optimal policies in vehicle networks [13]. In [14], the authors proposed another DRL-based offloading scheme to lower energy consumption by solving two split sub-problems. In detail, this approach first exploits deep neural network to represent the state of channel gain and further proposes an adaptive procedure by adjusting parameters automatically to reduce the complexity. However, these schemes focus on the non-realtime tasks. In addition, they do not consider DVFS technique in local computing, leading to more energy that is consumed on the fly.

III. System Model

In this section, we present the details of the task model, communication model, and energy consumption model adopted in the system under investigation.

A. Task Model

In mobile devices, a majority of realtime applications repeat periodically. Thus, we consider periodic realtime tasks in our scheme. A set of tasks is generally defined as follows:

$$T = \{t_1(p_1, w_1, b_1), t_2(p_2, w_2, b_2), ..., t_i(p_i, w_i, b_i)\}, \quad (1)$$

where t_i denotes i-th task and p_i is its period. w_i and b_i indicate the Worst-Case Execution Time (WCET) and the size of input data related to i-th task. The former is the longest execution time to complete the task, which can be approximated by particular estimation tools [10], while the latter is normally composed of program codes and the necessary parameters. Every task in the set has its utilization, which is calculated as the ratio of WCET to its period and defined as follow:

$$u_i = w_i/p_i \quad (2)$$

Thus, the total utilization of a task set T is calculated using Eq.(3).

$$U = \sum\nolimits_{\forall t_i \in T} {}^{w_i}\!/_{p_i}. \quad (3)$$

In each period, task t_i issues the j-th job that can be denoted as t_i^j, which has its own arrival time and execution time. The jobs are normally scheduled based on either static and dynamic priority. To ensure every scheduling periodic task being executed at least one time within a period, hyeperperiod is introduced in most realtime systems. The hyperperiod of a task set represents the Least Common Multiple (LCM) of all member tasks that can be calculated as follow:

$$H_p = LCM(p_1, p_2, ..., p_i). \tag{4}$$

B. Communication Model

In this section, we introduce the communication model in edge computing. Generally, the mobile device communicates with edge servers through the wireless access points. We assume that there are N ready tasks in one mobile device and one channel to connect the device and server. We denote o_i as offloading decision of task t_i, $i \in N$. That means, if o_i is equal to 0, task t_i is offloaded to an edge server. Otherwise, when o_i is equal to 1, the task is processed at the local device. Suppose that we have a decision profile of ready tasks $O = (o_1, o_2, ..., o_n)$, the data rate r_u of the uplink used to offload tasks from local device to a remote server can be calculated as follows [15]:

$$r_u = w \log_2 \left(1 + \frac{q_t c_g}{\sigma^2 + \sum_{i \in o_i = o_n} q_t c_g} \right), \tag{5}$$

where w denotes channel bandwidth and q_t is the transmission power that is determined by the power management method in edge server. Furthermore, c_g indicates the channel gain between local device and the connected server, and σ^2 means the Gaussian noise. In addition, we do not consider the downlink rate as the size of receiving data from the server is normally very much smaller than the uploading data size.

C. Energy Consumption Model

As the objective of the proposed approach is to optimize the total energy consumption in the mobile device, we only consider the local computation and communication energy consumption. The former typically consists of two parts, dynamic energy consumption and static energy consumption. The dynamic energy is consumed when executing instructions, while the static energy consumption is the leakage consumption that increases continuously even if the system becomes idle. The latter is due to the transmission of input data up to the remote servers.
1) Computation Energy Consumption: In modern processors, the dynamic energy consumption generally dominates the total energy consumption that can be computed as follow [7]:

$$P_d = c \cdot V^2 \cdot f, \tag{6}$$

where c is a coefficient, V and f denote the operating voltage and frequency, respectively. Alternatively, when the execution time of a task is too long, the accumulative static energy consumption cannot be omitted, which is calculated by Eq.(7) [13].

$$P_s = V \cdot I_s + |V_{bs}| \cdot I_j, \tag{7}$$

where I_s and V_{bs} denote the subthreshold current and the body bias voltage, and I_j is the reverse bias junction current. Thus, suppose the execution time of a task is ET_c, the computation energy consumption of this task can be given by

$$E_c = \lambda \cdot P_d \cdot ET_c + \rho \cdot P_s \cdot ET_c, \tag{8}$$

Here λ and ρ are two constants determined by specific hardware configuration.
2) Communication Energy Consumption: Generally, offloading tasks to edge server will lead to extra overhead like energy for data transmission. The amount of communication consumption highly depends on the size of input data and the transmission rate. Based on to Eq.(1) and Eq.(5), we can calculate this sort of consumption when delivering the input data of size b_n,

$$E_t = \frac{q_t b_n}{r_u} + L_{tail}, \tag{9}$$

where L_{tail} denotes the tail energy consumption [3]. It is incurred because the edge device still maintains the channel for a moment after the data transmission is completed. As the result, from Eq.(8) and Eq.(9), the total energy consumption of running a task (E_n) is calculated by the following equation,

$$E_n = E_c + E_t. \tag{10}$$

From the Eq.(10), we can observe that finding an appropriate offloading policy is critical for energy optimization in mobile devices.

IV. Reinforcement Learning Based Offloading

In this section, we first present the details of the proposed learning method, Reinforcement Learning with Intervention. Thereafter, we discuss the steps involved in the proposed offloading scheme, RRLO.

A. Double Q-learning with Intervention

Reinforcement learning is an autonomous learning process that learns the best policy through interactions with environment. In the standard RL framework, it has an agent, which makes decisions periodically and further improves the strategy gradually relying on the feedback until the optimal policy is achieved [16]. In a period, the agent first observes the environmental state s and then takes an action a on the environment. Before switching to the next period, either a reward r or penalty p is returned to the agent as the feedback. By repeating this process, the agent can learn the optimal policy π by maximizing the long-term accumulative reward.

Q-learning is one of the most popular model-free reinforcement learning method that has been applied to various applications [5], [10]. However, two limitations deter Q-learning to achieve better performance. First, many suboptimal decisions are made in the learning process as Q-learning only uses a single table to approximate Q values. Second, it is difficult to address the scenarios that have a set of constraints. In [17], van Hasselt proposed Double Q-learning as the solution of the former problem, while the latter problem is also tackled by providing certain intervention in each learning period [18].

To reduce suboptimal decisions while guaranteeing time constraints, we propose an improved version of Double Q-learning, Double Q-learning with Intervention, which considers a set of constraints in the learning process. Unlike Q-learning, the proposed method has two state-action tables and updates either of them in each decision point. Thus, the overestimation in Q values is largely mitigated.

As an improved Double Q-learning, the proposed method can conduct constraint checking in every period to avoid disastrous results when it is applied to constraint-critical applications. This improvement can be seen in Alg. 1 (specifically, Line 4 to Line 7) that checks whether a set of constraints are met or not in each individual period. Another benefit of the design is to facilitate learning convergence as the policies, which cannot meet the constraints, will be chosen with a low probability in the following learning process.

```
Algorithm 1: Double Q-learning with Intervention.
1  Initialize constraint set $\Gamma$, Q-table $Q_A$, Q-table $Q_B$, and state $s$;
2  Repeat;
3  Choose $a$, based on $Q_A$ and $Q_B$, and the next state $s'$;
4  Estimate penalty $p_e$;
5  if $(!\Gamma)$ then
6      $p \leftarrow MaximumValue$;
7  $p \leftarrow p_e$;
8  Choose to either update $Q_A$ or $Q_B$;
9  if Update $Q_A$ then
10     Define $a' = \arg\max_a Q_A(s', a)$;
11     $Q_A(s,a) \leftarrow Q_A(s,a) + \alpha(s,a)(p + \beta Q_A(s',a') - Q_A(s,a))$;
12 if Update $Q_B$ then
13     Define $a' = \arg\max_a Q_B(s', a)$;
14     $Q_B(s,a) \leftarrow Q_B(s,a) + \alpha(s,a)(p + \beta Q_B(s',a') - Q_B(s,a))$;
15 $s \leftarrow s'$;
```

B. Details of RRLO

In our framework, there is a set of N realtime periodic tasks $T = (t_1, ..., t_n)$, a set of K DVFS techniques $D = (df_1, ..., df_k)$, and a profile of offloading decisions $O_r = (o_1, ..., o_n)$. Note that $O_r \in O$. In addition, O is a set of optional policies. The number of policies in this set (i.e. 2^N) depends on how many tasks are ready in the system. The objective of RRLO is to choose the joint optimal offloading policy and DVFS technique (O_r, df_k) at the beginning of each hyperperiod so that the total energy consumption is minimized while all realtime tasks do not miss the deadlines. Note that O_r is the optimal offloading policy corresponding to the ready tasks (i.e. N tasks) while the df_k is the selected DVFS technique.

Reinforcement learning-based method takes a state as input and one action as the output. Thus, the state space in the

TABLE I
STATE-ACTION TABLE

State(SU, DS, CG)	Action(O_1, df_1)	...	(O_r, df_k)
(0.1,0.1,0.1)	...	...	...
(0.1,0.2,0.1)	...	...	...
(0.1,0.2,0.2)	...	...	...
...	...	...	...

scheme can be defined as $S = (s_1, ..., s_i)$. Each state s_i involves three parameters (SU, DS, CG), which denote system utilization, dynamic slack, and channel gain, respectively. The reason to choose the parameters is that DVFS technologies normally conduct frequency scaling based on system usage and slack time while offloading policy greatly depends on the channel conditions. The system utilization of T can be calculated as the following Eq.(11):

$$SU = \sum_{i=1}^{n} \frac{\omega_i}{p_i}. \tag{11}$$

Another parameter of dynamic slack, which is the time slack between AET and WCET, is given by

$$DS = 1 - \frac{Et_{H_p}}{\sum_{i=1}^{n}\left(\frac{H_p}{p_i} \times \omega_i\right)}, \tag{12}$$

where Et_{H_p} is the total actual execution time in a hyperperiod H_p that can be computed by using Eq.(13).

$$Et_{H_p} = \sum_{i=1}^{n} \sum_{j \in H_p} AET\left(t_i^j\right), \tag{13}$$

Here t_i^j denotes j-th job that is generated by i-th task in specific hyperperiod.

The channel gain CG that represents the network condition at time t can be calculated by Eq.(14) [14].

$$CG = \alpha_t A_d \left(\frac{\xi}{4\pi f_c d_i}\right)^{d_e}, \tag{14}$$

where α_t is the Rayleigh fading factor and ξ is a constant. A_d, f_c and d_e denote antenna gain, carrier frequency, and path loss exponent, respectively. Our RRLO method is illustrated in Alg. 2, which works as follows:

Step 1: The scheme initializes a set of variables before entering a learning loop. At the beginning of each hyperperiod H_p, the scheduler of the scheme first observes current system state $s(SU, DS, CG)$. Based on the state s, an action $a(O_r, df_k)$, which owns the minimum Q-value in both Q tables (Q_A and Q_B), is selected accordingly. One example of the quantized state-action table is shown in Table I.

Step 2: According to the selected offloading policy O_r, the task set T is divided into two subsets T_o and T_l. The tasks in subset T_o are delivered to the edge server for computation, while T_l is scheduled in the local device. Meanwhile, the chosen DVFS technique df_k is leveraged to schedule the local task set T_l.

TABLE II
TASK DESCRIPTIONS

Task No.	period	WCET	jobs	data
1	50	7	6000	300
2	30	2	6000	150
3	100	10	3000	500
4	60	3	5000	400

Step 3: At the end of each hyperperiod, the scheduler calculates the computation energy consumption E_c and transmitting energy consumption E_t. In addition, the scheduler checks whether there is deadline miss by scheduling with the action $a(O_r, df_k)$. If the time constraint cannot be met, the penalty p is assigned to a predefined large value. Otherwise, the sum of E_c and E_t is returned to the scheduler as the penalty.

Step 4: Based on the penalty p and next state s', the scheduler randomly updates either Q_A or Q_B. Next, it goes back to step 1 until all tasks are completed. Finally, the total energy consumption of T is calculated.

According to the Alg. 2, finding system state and the optimal action dominates the computational complexity in this scheme. The complexity of the former is $O(1)$, while the latter is $O(2^N)$ since it requires $2^N \times K$ comparisons, where N is the number of ready tasks in a hyperperiod and the constant K is the number of DVFS techniques. As for space complexity, it can be formulated as $(O(|SU| \times |DS| \times |CG| \times |2^N| \times |K|))$, which depends on the size of Q tables.

V. PERFORMANCE EVALUATION

In this section, we present the details of the experimental results, which clearly indicates the advantage of the proposed scheme.

A. Experiment Configuration

In the simulations, we evaluate the performance of RRLO by thoroughly comparing RRLO with Local Scheduling (i.e. all the jobs are processed locally). Note that RRLO is based on Double Q-learning [8]. To further understand the advantages of RRLO, we developed a variant of RRLO, which is denoted as RRLO-Q-Learning because it is based on Q-learning. Technically, RRLO-Q-Learning learns the optimal policies of offloading and task scheduling using Q-learning. The simulator, which integrates four DVFS techniques(CC, LA, DRA, ARG) and offloading candidates, is developed using C++. Four realtime tasks are used in the experiments. Each of them consists of four tuple descriptions (period, WCET, jobs, and data) that are generated randomly. The details are shown in Table II. For each task, the related AET of jobs follows the uniform distribution in the range [0, WCET]. According to Eq. (4), the hyperperiod of the task set is $LCM(50, 30, 100, 60) = 300$. For simplicity, we assume the transmission power is 50 mWatts for all tasks. In Double Q-learning, the learning rate and discount rate are 0.8 and 0.9, respectively.

Algorithm 2: Realtime Reinforcement Learning Offloading.

```
   Input: Task set $T$, Q-table $Q_A$, Q-table $Q_B$
   Output: $a(O_r, df_k), O_r \in O, df_k \in D$
 1 Function MainScheduling;
 2 Initialize Q tables $Q_A \leftarrow 0, Q_B \leftarrow 0$;
 3 for $H_p = 1, 2..., i$ do
 4     Observe the system utilization $SU$, dynamic slack
       $DS$, channel gain $CG$;
 5     Current state $s_i \leftarrow s(SU, DS, CG)$;
 6     Choose action $a_i \leftarrow a(O_r, df_k), a(O_r, df_k)$ with the
       minimum Q value in state $s_i$;
 7     Divide $T$ into $T_o$ and $T_l$ based on $O_r$;
 8     Offload $T_o$ to edge server;
 9     Schedule $T_l$ in local device;
10     Calculate penalty $p \leftarrow PenaltyCalculation()$;
11     Update either of tables $UpdateQtables(p, s_i, a_i)$;
12 Function $PenaltyCalculation()$;
13 Calculate $E_c$ using Eq.(8);
14 Calculate $E_t$ using Eq.(9);
15 for $j = 1, 2..., n$ do
16     if $MissDeadline(t_j)$ then
17         $p \leftarrow MaximumValue$;
18 $p \leftarrow E_c + E_t$;
19 Return $p$;
20 Function $UpdateQtables(p, s_i, a_i)$;
21 Observe next state $s'$;
22 $s \leftarrow s_i$, $a \leftarrow a_i$;
23 if Update $Q_A$ then
24     $a' = \arg\max_a Q_A(s', a)$;
25     $Q_A(s, a) \leftarrow$
       $Q_A(s, a) + \alpha(s, a)(p + \beta Q_B(s', a') - Q_A(s, a))$;
26 if Update $Q_B$ then
27     $a' = \arg\max_a Q_B(s', a)$;
28     $Q_B(s, a) \leftarrow$
       $Q_B(s, a) + \alpha(s, a)(p + \beta Q_B(s', a') - Q_B(s, a))$;
```

B. Experimental Results

In our research, we first compared the proposed offloading scheme with the other two methods under investigation using three sets of realtime tasks. Every set is composed of three tasks that are randomly selected from Table II. In our simulations, the model is trained with a different task set, whose tasks involve a certain number of jobs with different AET. Furthermore, we used two different task sets to evaluate the performance of the scheme in terms of energy optimization.

From Fig. 1, we can arrive at two conclusions. First, compared to Local Scheduling that only processes tasks at local devices, the two learning-based offloading schemes have better performance in terms of energy reduction (at least 13%). We believe that the reason behind the results is that, within the same time period, the learning-based schemes could deliver a number of computational tasks to the remote edge servers that have more computation ability and resource, consequently resulting in less energy consumption locally. Second, RRLO outperforms RRLO-Q-learning because of the mitigation of Q-values overestimation in the learning process so that it can choose the optimal policy more precisely. Furthermore, as seen in Fig .2, when the task number grows up to four, the proposed scheme also has stable performance in terms of energy optimization, and works better than Local Scheduling and RRLO-Q-learning offloading. The reason is that the reclaim of a large amount of time slack allows RRLO to lower the local energy consumption while the tradeoff between computation energy consumption and offloading overhead is achieved.

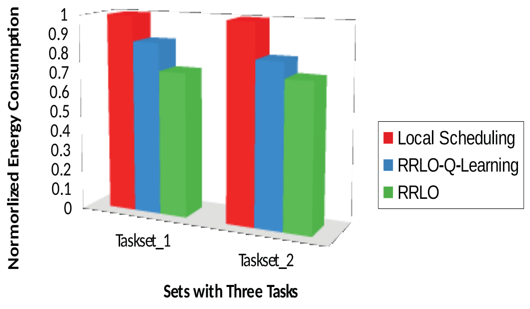

Fig. 1. Experimental Results Based on Three Tasks

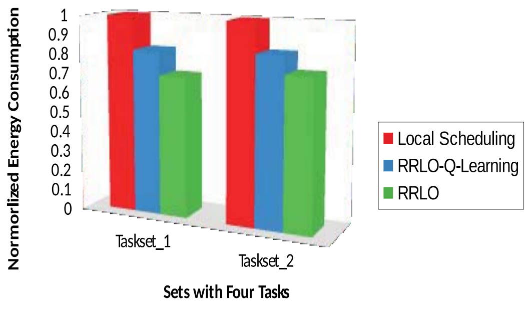

Fig. 2. Experimental Results Based on Four Tasks

VI. Conclusion

In this paper, we propose, RRLO, a learning-based offloading scheme for realtime applications on mobile devices in MEC. Specifically, an improved version of Double Q-learning, called Double Q-learning with Intervention, is proposed to guarantee that the deadlines of realtime tasks can be satisfied. Based on the proposed learning method, RRLO is capable of finding the optimal offloading policy and task scheduling method simultaneously. In practice, by interacting with system environment, RRLO could selectively offload part of computational tasks to MEC servers so that the total energy consumption is minimized. Our simulation results indicate that the proposed scheme outperforms Local Scheduling and RRLO-Q-learning.

Acknowledgment

This study was supported by National Natural Science Foundation of China (No. 61772154). It was also supported by Shenzhen Basic Research Program (No. JCYJ20190806143011274). In addition, it was supported by National Sciences and Engineering Research Council of Canada (No. RGPIN-2017-05853).

References

[1] X. F. Chen, H. G. Zhang, C. Wu, S. W. Mao, Y. S. Ji and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning", *IEEE Internet of Things Journal*, vol. 6, no.3, pp. 4005-4018, June 2019.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading", *IEEE Communications Surveys & Tutorials*, vol. 19, no.3, pp. 1628-1656, June 2017.

[3] S. T. Guo, J. D. Liu, Y. Y. Yang, B. Xiao and Z. T. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing", *IEEE Transactions on Mobile Computing*, vol. 18, no.2, pp. 319-333, Februry. 2019.

[4] Y. L. Geng, Y. Yang, and G. H. Cao, "Energy-efficient computation offloading for multicore-based mobile devices", in *Proceedings of IEEE Computer Communications Conference(INFOCOM)*, 2018, pp. 46-54.

[5] N. C. Luong, D. T. Hoang, S. M. Gong, D. Niyato, P. Wang, Y. C. Liang and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey", *IEEE Communications Surveys & Tutorials*, May, 2019, DOI: 10.1109/COMST.2019.2916583.

[6] H. Aydin, R. Melhem, D.Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks", *IEEE Transactions on Computers*, vol. 53, no.5, pp. 584-600, May, 2004.

[7] Q. C. Zhang, M. Lin, L. T. Yang, Z. K. Chen and P. Li, "Energy-efficient scheduling for real-time systems based on deep Q-learning model", *IEEE Transactions on Sustainable Computing*, vol.4, no. 1, pp. 132-141, March, 2019.

[8] H. Huang, M. Lin and Q. C. Zhang, "Double-Q Learning-Based DVFS for Multi-core Real-Time Systems", in *Proceedings of Green Computing and Communications*, 2017, pp. 522-529.

[9] M. Bhatti, C. Belleudy, and M. Auguin, "Hybrid power management in real time embedded systems: An interplay of DVFS and DPM techniques", *Real-Time Systems*, vol. 47, no. 2, pp. 143-162, Mar. 2011.

[10] F. Islam and M. Lin, "Hybrid DVFS scheduling for real-time systems based on reinforcement learning", *IEEE Systems Journal*, vol. 11, no. 2, pp. 931-940, June, 2017.

[11] Q. C. Zhang, M. Lin, L. T. Yang, Z. K. Chen, S. U. Khan and P. Li, "A double deep Q-learning model for energy-efficient edge scheduling", *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 739-749, October, 2019.

[12] J. Li, H. Gao, T. Lv and Y. M. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC", in *Proceedings of Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1-6.

[13] Y. Liu, H. M. Yu, S. L. Xie and Y. Zhang, "Deep Reinforcement Learning for Offloading and Resource Allocation in Vehicle Edge Computing and Networks", *IEEE Transactions on Vehicular Technology*, August, 2019, DOI: 10.1109/TVT.2019.2935450.

[14] L. Huang, S. Z. Bi and Y. J. Zhang, "Deep Reinforcement Learning for Online Computation Offloading in Wireless Powered Mobile-Edge Computing Networks", *IEEE Transactions on Mobile Computing*, July, 2019, DOI: 10.1109/TMC.2019.2928811.

[15] X. Chen, L. Jiao, W. Z. Li, and X. M. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing". *IEEE Transactions on Networking*, vol. 24, no. 5, pp. 2795-2808, October. 2016.

[16] L. Pinto, J. Davidson, R. Sukthankar and A. Gupta, "Robust adversarial reinforcement learning", In *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 2817-2826.

[17] H. van Hasselt, "Double Q-learning", In *Proceedings of Advances in Neural Information Processing Systems*, 2010, pp. 2613-2621.

[18] W. Saunders, G. Sastry, A. Stuhlmueller and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention", In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2018, pp. 2067-2069.