A Decentralized Reactive Approach to Online Task Offloading in Mobile Edge Computing Environments

Qinglan Peng[1], Yunni Xia[1(✉)], Yan Wang[2], Chunrong Wu[1], Xin Luo[3(✉)], and Jia Lee[1]

[1] Software Theory and Technology Chongqing Key Lab, Chongqing University, Chongqing, China
xiayunni@hotmail.com
[2] Department of Computing, Macquarie University, Sydney, NSW 2109, Australia
[3] Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing, China
luoxin21@cigit.ac.cn

Abstract. In mobile edge computing (MEC) environments, the task offloading towards nearby edge servers usually occurs when local resources are inadequate for computation-intensive applications. While the MEC servers benefit from the close proximity to the end-users to provide services at reduced latency and lower energy costs, they suffer from limitations in computational and radio resources, which calls for smart, timely, and efficient offloading methods and strategies. In this paper, we consider an arbitrary request arrival pattern and formulate the MEC-oriented task offloading problem as an online multi-dimensional integer linear programming. We propose a decentralized reactive approach by adopting a dynamic-learning mechanism to yield online offloading decisions upon request arrivals. Experiments based on real-world MEC environment datasets show that our method outperforms state-of-the-art ones in terms of offloading responsiveness and efficiency.

Keywords: Mobile edge computing · Task offloading · Reactive scheduling · Decentralized scheduling

1 Introduction

Due to restricted battery power, storage, and computational capacity, mobile devices face challenges in executing delay-sensitive and resource-hungry mobile applications such as augmented reality and online gaming [6]. As a newly emerging computing paradigm, edge computing shows great capability in supporting and boosting such computation-intensive mobile applications. In the mobile edge computing (MEC) environments, the mobile edge is enhanced with computation resources and storage capabilities, possibly by the dense deployment of computational servers or by strengthening the already-deployed edge entities such as small-cell base stations. Consequently, mobile devices are able to offload their

E. Kafeza et al. (Eds.): ICSOC 2020, LNCS 12571, pp. 232–247, 2020.
https://doi.org/10.1007/978-3-030-65310-1_18

computationally intensive tasks to the edge servers to alleviate the insufficiency of local computing power and capacity [4].

In practice, MEC-oriented offloading requests can be dynamically submitted in real-time by end-users and specified with timing constraints. They require both logical and temporal correctness of computations. Due to the dynamic nature in MEC environments and the heterogeneity of both real-time tasks and edge servers, traditional centralized offloading strategies, where an offloading decision for a batch of requests is made at the centralized scheduler and performed simultaneously, can lead to bad user-perceived Quality-of-Service (QoS) and long waiting time of end-users. Intuitively, such long waiting time and low system responsiveness are usually caused by the fact that asynchronous requests have to gather at the scheduler first before the synchronous offloading decisions for a batch of such requests are made and carried out.

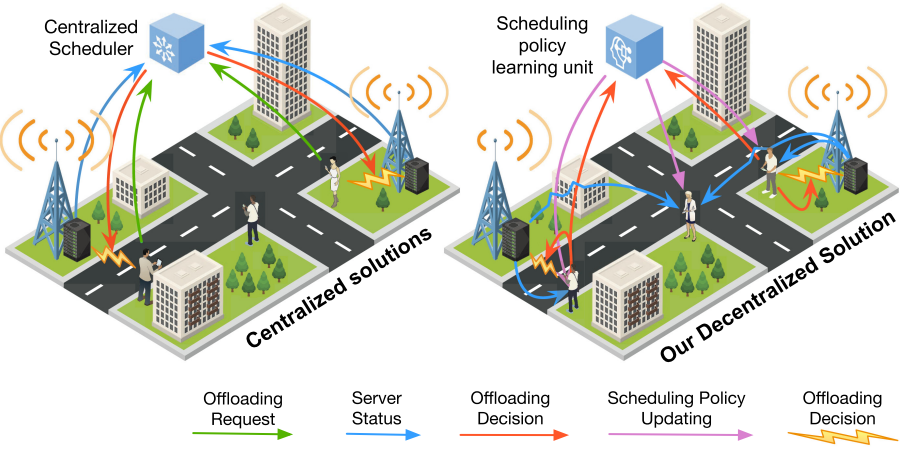

Fig. 1. System architecture comparison between ours and traditional ones.

In this paper, instead of considering centralized and batch-processing-based offloading decision-making, we propose a decentralized reactive approach to yield online offloading decisions upon request arrivals. Figure 1 shows the differences between ours and the traditional ones. It can be seen in the traditional solution that a centralized scheduler requires a global view of system status and all offloading requests are forwarded to it before offloading decisions are made and performed. Consequently, in addition to low system responsiveness, the centralized scheduling pattern can be susceptible to single-point-of-failures (SPOFs) [16] that affect all server nodes when the central one crashes and cause high communication overhead among the central scheduler and edge servers. There are also some studies, e.g., [10,17,23], that proposed distributed solutions by utilizing the parallelism feature of game-theory or alternating direction method of multipliers (ADMM). However, they still need a centralized coordinator to exchange sub-decisions to reach the final decision. In contrast, our approach is

completely decentralized, where the online offloading decisions are locally made by and applied to individual mobile users themselves instead of a centralized scheduler. Moreover, in our work, a significant difference exists from the traditional system architectures, namely, our proposed architecture is empowered by a learning unit which is responsible of continuously learning scheduling policies and pushing them to end-users, the dynamicity (e.g., the fluctuation of offloading request arrival rates and uneven geographical distribution of requests) of MEC environments is thus captured and tackled.

The main contributions of this work are as follows: 1) we propose a completely decentralized architecture for MEC-oriented online task offloading problem by devolving the offloading decision-making power from a centralized scheduler to end-users themselves; 2) instead of assuming the request arrival rates are pregiven or follow certain distributions, we consider an arbitrary request arrival pattern, which is more in line with the practical application scenarios; 3) we present an efficient reactive dynamic-learning-based [1] approach which is capable of continuously learning and updating scheduling policies. We conduct a series of case studies based on two well-known real-world edge environment datasets and the experimental results have demonstrated the proposed approach clearly outperforms traditional ones in terms of average end-to-end offloading delay and decision-making time.

2 Related Work

Mobile task offloading aims at executing the computation-intensive and latency-sensitive mobile tasks with the help of external resources [16]. Recently the MEC-oriented mobile task offloading problem has attracted a lot of research interests and extensive efforts are devoted to it.

Some studies focus on the static optimization of multiple-server multiple-user task offloading problems during a request arrival batch window. For example, Chen *et al.* [4] formulate the task offloading problem at a certain time point as a mixed-integer-programming (MIP) problem, then they decompose and convert it into two convex optimization ones and develop a Lagrange-multiplier-based algorithm to solve them. Alameddine *et al.* [2] take the offloading request admission rate as the target and propose a Logic-Based-Benders-Decomposition-based method to reduce the searching space of the MIP problem. Yang *et al.* [23] consider both energy consumption and latency as offloading targets, they formulate the offloading problem as a potential game and present a distributed game-theoretical approach to solving it. Dai *et al.* [5] propose a two-tier computation offloading framework to cope with the heterogeneous networks and multiple mutually dependent mobile tasks, then the semi-definite-program and linear-convex-program are employed to yield user association plan and offloading decisions, respectively. Yang *et al.* [22] formulate the offloading decision problem as a classification one and propose a feedforward neural network model to optimize the offloading decisions.

There are also some studies considering the long-term offloading QoS as target and present the corresponding online solutions. For example, Liu *et al.* [14] and Du *et al.* [7] assume the task arrivals are i.i.d. over time and their average arrival rates

are pre-given. Chen *et al.* [3] and Xu *et al.* [21] consider that task arrivals follow a Poisson process and they both use random arrival rates to capture the temporal variation of task arrival pattern at different time. While Zhao *et al.* [24] assume the offloading demands of mobile users follow a binomial distribution with a pre-given probability. Based on these assumptions, they all build a local-edge two-tier queue model and utilize a Lyapunov optimization technique to maintain the stability of the system to achieve an online optimization. Huang *et al.* [11] present a deep-reinforcement-learning-based method to get online offloading decisions. However, their approach is a centralized one and they only consider one edge server in their system model.

A careful investigation into the aforementioned studies shows that they are still limited in three ways: 1) most of the existing studies base themselves upon centralized architecture that needs a central scheduler to yield offloading solutions, which may suffer from SPOFs and low system responsiveness; 2) the batch-processing scheduling mode of some static studies, e.g., [4,10], may result in additional waiting time, which may lead to bad user-perceived QoS and long waiting time; 3) some online studies, e.g., [3,14,24], are based on the assumptions that the offloading request arrival rates are pre-given or follow certain distributions. However, these assumptions are unrealistic in real-world application scenarios where offloading requests could arrive at an arbitrary time and their arrival rates fluctuate over time. Therefore, a decentralized reactive approach that adapts to a more general application scenario is in high demand.

3 Preliminary

In this section, we present our system model and give the problem formulation of online task offloading in MEC environments. Table 1 lists the notations used in our system model.

Table 1. List of notations

Notation	Description	Notation	Description
a_j	Arrival time of task t_j.	q_i	Maximum size of the task queue of edge server e_i.
b_i	Maximum total size of tasks that server e_i can handle in period P.	q_{ji}	Instantaneous task queue size of server e_i when task t_j arrivals.
c_i, c_j	Computing capability of edge server e_i, requester of task t_j.	r_i	Radius of edge server e_i' signal coverage.
d_{ji}	Distance between server e_i and user who invokes task t_j.	r_{ji}	Uplink data rate between edge server e_i and the requester of task t_j.
e_i	The i-th edge server in E.	$\mathbb{R}_{ji}$	Response time of offloading t_j to e_i.
E	The set of edge servers.	s_j	Data content size of t_j.
f_{ji}	Transmission delay of uploading the data of task t_j to server e_i.	t_j	The j-th task in T.
h_{ji}	Execution time of task t_j on the e_i.	T	The set of offloading requests.
l_i, l_j	Real-time geographic location of server e_i, requester of task t_j.	u_i	Utility value of server e_i that has been learned.
m	The count of edge servers in E.	v_j	Earliest start time of task t_j on server e_i.
n	The count of Offloading requests in T.	x_{ji}	An indicator to identify whether task t_j is scheduled to server e_i.
o_j	The count operations in task t_j.	ϵ	Decision repository length factor

3.1 System Model

As shown in Fig. 1, in the MEC environments, base stations are equipped with a certain amount of computing resources (i.e., edge servers) to fulfill users' offloading demands. Suppose that there are m edge servers in total in the MEC environments, and we use a set $E = \{e_1, e_2, ..., e_m\}$ to represent them. Each edge server can be described by a 4-tuple (l_i, r_i, c_i, q_i), where l_i is the geographic location of server e_i, r_i its radius of signal coverage, c_i its computing capability (e.g., CPU cycles per second [2,14]), and q_i its task queue length.

Mobile users in MEC environments who act in this region are allowed to offload their computational tasks to nearby available edge servers, and we use a set $T = \{t_1, t_2, ..., t_n\}$ to represent the set of n tasks offloaded by users in a period P. Each task can be described by a 5-tuple $(a_j, l_j, o_j, s_j, c_j)$, where a_j is the arrival time of task t_j, l_j the location of the user who offloaded t_j, o_j the computation amount (i.e., the CPU cycles needed to complete the task) of t_j, s_j the size of data contents of t_j, and c_j the computing capability of the requester of t_j itself.

In this paper, we consider that mobile users' computational tasks can be executed by both signal reachable edge servers [10,12,13] and local mobile devices [2,5,23]. We use d_{ji} to identify the distance between server e_i and the requester of task t_j. Like various well-known data processing and AI service engines (e.g., Apache Spark[1], Flink[2], and Tensorflow Serving [8]) do, edge servers in our model have multiple computing units and handle incoming tasks on a First-Come-First-Service (FCFS) basis as shown in Fig. 2. Therefore, given the status of server e_i at time a_j, the end-to-end delay of offloading task t_j to server e_i can be estimated as:

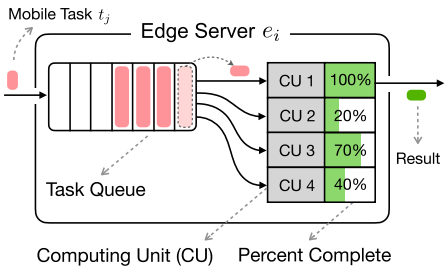

Fig. 2. Offloading request processing flow at edge server.

$$\mathbb{R}_{ji} = \begin{cases} f_{ji} + (v_j - a_j) + h_{ji}, & d_{ji} \leq r_i, \quad i = 1, 2, ..., m \\ h_{ji}, & i = 0 \end{cases} \tag{1}$$

where f_{ji} is the data transmission delay, h_{ji} is the task execution delay, v_j is the earliest start time of t_j on e_i, and thus $(v_j - a_j)$ is the waiting time of t_j at e_i's task queue. If there are no available edge servers around the requester of t_j, the offloading request t_j is declined and it will be executed by users' local devices, we use $i = 0$ to represent this situation. It can be seen that the end-to-end delay of offloading a mobile task to an edge server (i.e., $i \neq 0$) consists of three parts:

[1] https://spark.apache.org/docs/latest/job-scheduling.html.
[2] https://ci.apache.org/projects/flink/flink-docs-stable/internals/job_scheduling.html.

task uploading delay f_{ji}, waiting time $(v_j - a_j)$, and the execution delay h_{ji}. The task execution delay h_{ji} can be estimated as [2,4,22]:

$$h_{ji} = \begin{cases} o_j/c_i, & d_{ji} \leq r_i, \quad i = 1, 2, ..., m \\ o_j/c_j, & i = 0 \end{cases} \tag{2}$$

And the data transmission delay f_{ji} can be estimated as $f_{ji} = s_j/r_{ji}$, where r_{ji} is the user's uplink data rate and it can be calculated as [4]:

$$r_{ji} = \beta \log_2 \left(1 + \frac{p_j g_{ji}}{\delta^2}\right), \tag{3}$$

where β is the channel bandwidth, p_j the transmission power of the requester of task t_j, g_{ji} the channel gain between the server e_i and the requester of t_j, and δ^2 the background noise power.

3.2 Problem Formulation

Based on the above system model, we have great interest to know: for a given region where m edge servers are deployed with heterogeneous resource configurations and n offloading requests asynchronously raised in a period P, how to appropriately respond and take offloading actions with optimized offloading efficiency in terms of end-to-end offloading delay. The problem can be formulated as follows:

$$Min: \quad \frac{1}{n}\Big(\sum_{j=1}^{n}\sum_{i=0}^{m}\mathbb{R}_{ji}x_{ji}\Big) \tag{4a}$$

$$s.t: \quad d_{ji} \leq r_i, \qquad \forall x_{ji} \neq 0 \tag{4b}$$

$$s_j + q_{ji} \leq q_i, \qquad \forall j = 1, 2, ..., n, \quad \forall i = 1, 2, ..., m \tag{4c}$$

$$\sum\nolimits_{j=1}^{n} o_j x_{ji} \leq b_i, \qquad \forall i = 0, 1, ..., m$$

$$\sum\nolimits_{i=1}^{m} 0 \leq x_{ji} \leq 1, \qquad \forall j = 1, 2, ..., n$$

where $x_{ji} \in \{0, 1\}$ is the offloading decision that indicates whether task t_j is going to be scheduled to server e_i (e_0 represents local device), which is made in real-time upon t_j arrivals. b_i is the maximum computation amount that e_i can process within the period P, which can be estimated as $c_i \times P$, and q_{ji} the instantaneous task queue size of e_i when t_j arrives.

As shown in (4a), the objective is to minimize the average end-to-end delays of offloaded tasks arriving in a period P. (4b) and (4c) are the constraints of offloading distance and task queue capacity. (4d) indicates that an edge server is feasible to an offloading requester only if it has enough computation amount.

Note that in this paper, we consider users' offloading requests could arrive at an arbitrary time and we aim at yielding online offloading decision in a reactive manner upon request arrivals. The specifications of individual offloading

requests are unknown before being raised and thus the problem can be formulated as an Online-Multidimensional-Integer-Linear-Programming with no optimal online solutions [1,9].

4 Proposed Dynamic-Learning-Based Approach

For the problem formulated above, in this section, we adopt a dynamic-learning mechanism [1] and propose a decentralized reactive approach, shorts for DRA, to yield reactive and online offloading decisions. Figure 3 shows the process of the proposed approach.

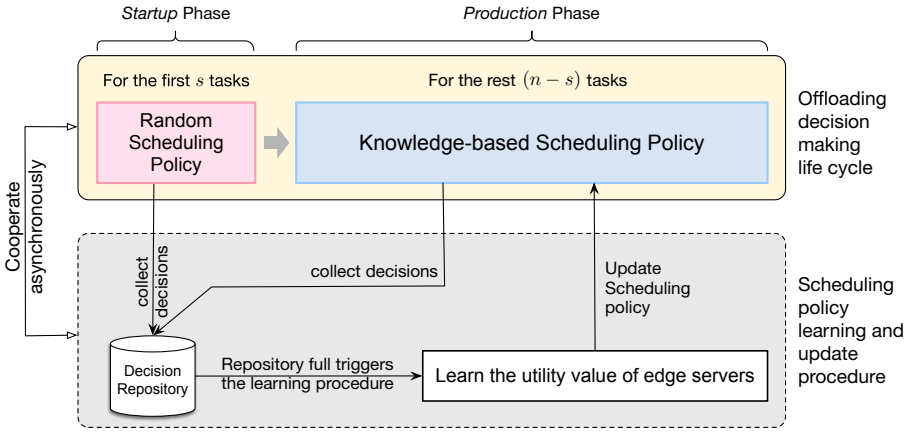

Fig. 3. The process of proposed DRA approach.

It can be seen that at the *startup* phase, a random scheduling policy is performed for the first s requests to collect training data to start the dynamic-learning procedure. The initial length of the decision repository is set to s and a dynamic-learning procedure is triggered to learn the scheduling policies when the repository is full. After that, our approach moves forward into the *production* phase, where the offloading decisions are made by the learned knowledge-based scheduling policies. Note that, due to the high dynamicity of the MEC environments, scheduling policies in this phase are evolvable and our learning procedure keeps updating and pushing them to end-users.

The knowledge-based scheduling policies are learned by reviewing the effect of past decisions. We use σ_j to denote the reward (i.e, end-to-end delay) of the j-th decision, and $y_j \in \{0, 1\}$ to determine the correctness of the j-th decision from the reviewing perspective. The above determination problem can be formulated to a linear programming (LP) as shown in (5).

Primal-LP

$$
\begin{aligned}
Min: \quad & \frac{1}{n}\Big(\sum_{j=1}^{s}\sigma_j y_j\Big) \\
s.t: \quad & \sum_{j=1}^{s} o_j y_j \le (1-\epsilon)\frac{s}{n}b_i, && \forall i \\
& 0 \le y_j \le 1, && \forall j \\
& i = 1,2,...,m, \quad j = 1,2,...,n
\end{aligned}
$$

Dual-LP

$$
\begin{aligned}
Max: \quad & \sum_{i=1}^{m}(1-\epsilon)\frac{s}{n}b_i u_i + \sum_{j=1}^{s} z_j \\
s.t: \quad & \sum_{i=1}^{m} o_j u_i + z_j \ge \sigma_j, && \forall j \\
& u_i, z_j \ge 0, && \forall i,j \\
& i = 1,2,...,m, \quad j = 1,2,...,n
\end{aligned}
$$

We also present its dual problem in (6), let $(\hat{\mathbf{u}}, \hat{\mathbf{z}})$ be its optimal solution, where vector $\hat{\mathbf{u}}$ indicates the utility value for all edge servers and it can be used to evaluate the gains of offloading decisions. We use a function $G(\hat{\mathbf{u}}, t_j)$ to represent the knowledge-based scheduling policy in the *production* phase:

$$
G(\hat{\mathbf{u}}, t_j) = \begin{cases} 0, & \text{if } (\mathbb{R}_{ji} - \sum_{i=1}^{m} u_i o_j) \le 0 \\ \underset{i}{\operatorname{argmin}}(\mathbb{R}_{ji} - \sum_{i=1}^{m} u_i o_j), & \text{otherwise} \end{cases} \tag{5}
$$

where $i = 1, 2, ..., m$ and $d_{ji} \le r_i$.

Algorithm 1 shows the specific steps of the proposed approach. It can be seen that DRA starts with initializing the decision repository R with $\varnothing$ and setting its length to $s = n\epsilon$ (as shown in lines 1–2), where $\epsilon \in (0, 1)$. Then, it performs a random scheduling policy to schedule the first s tasks and collects the corresponding decisions to R (as shown in lines 3–12). If the length of R equals s, it begins to learn the utility value of each edge server by solving the dual problem defined in (6). After that, for the rest incoming tasks, a knowledge-based scheduling policy is performed to reactively yield online offloading decisions (as shown in lines 13–25). Note that, once the decision repository is full, the learning procedure is triggered and the length of the repository is doubled to collect more decisions to realize dynamic learning (as shown in lines 15–17).

For the *startup* phase of our approach, the time complexity of getting nearby edge servers and filtering out unavailable ones (i.e., edge servers without enough computing power or full task queues) is $O(m)$, the time complexity of making a random offloading decision is $O(1)$. Thus, the total time complexity of making an offloading decision in the startup phase is $O(m)$. For the *production* phase, the time complexity of getting nearby available servers is also $O(m)$, but the time complexity of making an offloading decision by a knowledge-based scheduling policy, i.e., $G(\hat{\mathbf{u}}, t_j)$, is $O(m \log m)$. Thus, the total time complexity of making an offloaded decision in the production phase is $O(m \log m)$.

Competitive ratio analysis is a useful way to evaluate the optimality of online algorithms. An online algorithm is c-competitive if its expected performance could reach at least c factor of the optimal solution of the problem form the offline perspective. As proved by Agrawal *et al.* [1], the proposed DRA achieves a $1 - O(\sqrt{m \log n / B})$ competitiveness, where $B = min(b_i)$.

Algorithm 1: DRA

Input: Edge server count m; Offloading request count n; Edge server set E; Task set T; decision repository length base factor ϵ

Initialize offloading decision repository $R \leftarrow \varnothing$
Set the length of repository $s \leftarrow n\epsilon$
foreach t_j **in** the first s tasks of T **do**
    Initialize offloading decision $\boldsymbol{x_j} \leftarrow [0, 0, ..., 0]$
    $E^{'} \leftarrow$ Get all available edge servers around the invoker of t_j from E
    if $E^{'} = \varnothing$ **then**
        $x_{j0} \leftarrow 1$
    else
        $i \leftarrow$ Randomly select an edge server e_i in S and record i
        $x_{ji} \leftarrow 1$
    Schedule task t_j according to offloading decision $\boldsymbol{x_j}$
    Add $\boldsymbol{x_j}$ to decision repository R
foreach t_j **in** the rest tasks of T **do**
    Initialize decision $\boldsymbol{x_j} \leftarrow [0, 0, ..., 0]$
    if $length(R) = s$ **then**
        $(\hat{\mathbf{u}}, \hat{\mathbf{y}}) \leftarrow$ Solve the dual problem defined in (6)
        $s \leftarrow s \times 2$
    $E^{'} \leftarrow$ Get all available edge servers around the invoker of t_j from E
    if $E^{'} = \varnothing$ **then**
        $x_{j0} \leftarrow 1$
    else
        $i \leftarrow G(\hat{\mathbf{u}}, t_j)$
        $x_{ji} \leftarrow 1$
    Schedule task t_j according to offloading decision $\boldsymbol{x_j}$
    Add $\boldsymbol{x_j}$ to decision repository R

5 Experiments and Analysis

In this section, we conduct a series of case studies based on two real-world edge environment datasets to evaluate the performance of our proposed approach in terms of offloading responsiveness and efficiency.

5.1 Experiment Settings

EUA [12] and Telecom [18–20] are two well-known edge environment datasets, where EUA dataset[3] includes the geographic positions of edge servers and users in a CBD area in Melbourne, Australia, and Telecom dataset[4] contains the geographic positions of edge servers and the arrival time of edge users in the

[3] https://github.com/swinedge/eua-dataset.
[4] http://www.sguangwang.com/TelecomDataset.html.

Table 2. The details of two test cases in our experiment

Settings	Case	
	EUA [12]	Telecom [20]
Location	CBD area, Melbourne, AU	Central Urban area, Shanghai, CN
Upper-left coordinate	(−37.813134, 144.951300)	(31.233580, 121.423307)
Lower-right coordinate	(−37.815240, 144.974820)	(31.196371, 121.494572)
Area	2.04 km^2	32.67 km^2
Base station count	125	190
Edge user count	4000 ∼ 8000	10000 ∼ 20000
Experimental duration	2 h (7200 s)	2 h (7200 s)

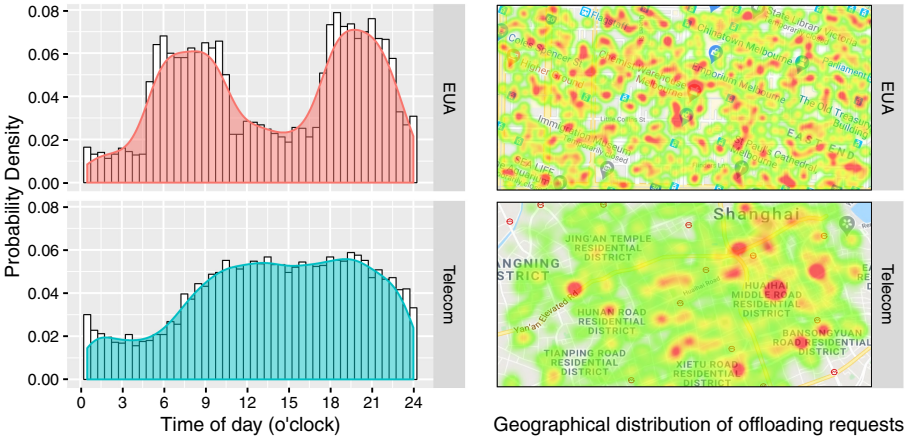

Fig. 4. Geographic distribution and arrival time probability density of offloading requests during a day.

urban area of Shanghai, China. In our experiments, we consider the whole region of the EUA dataset and the central region of the Telecom dataset as test cases, and Table 2 shows the details of them. As for the request arrival time data of the EUA case and user position data of the Telecom case, we use Uber request arrival time and Shanghai's taxi trajectories to make a supplement as illustrated in Fig. 4. It can be seen that users' offloading requests are distributed unevenly in different geographical areas and the arrival rates fluctuate over time. We choose the requests which arrive from 8 AM to 10 AM (i.e., P equals to 2 h) to conduct our case studies.

In this paper, we consider both heterogeneous edge servers configurations and offloading requests. Table 3 shows the configurations of edge servers, users' mobile devices, and offloading requests in our experiments [4,5]. We scale the computing capability of edge servers from 75% to 125% to evaluate the performance of our approach and baselines under different edge resources. We also

Table 3. Experiment configuration

Parameters	Value	Parameters	Value
Computing capabilities of edge servers	2.2 ∼ 3.8 GHz	Computing units count of edge servers	4 ∼ 12
RAM size of edge servers	16 GB ∼ 2 TB	Computing capabilities of mobile devices (equivalent)	0.8 ∼ 1.2 GHz
Computation amount of offloading requests	50 ∼ 100 cycles/bit	Data amount of offloading requests	5 ∼ 100 MB
Communication channel gain	$127 + 30 \times \log d$	Transmission power	0.5 W
Bandwidth of mobile devices	20 MHz	Background noise power	2×10^{-13} W
Radius of edge servers' signal coverage	300 ∼ 600 m	Decision repository length factor ϵ	0.01

scale the number of offloading requests from 4000 to 8000 for EUA cases and 10000 to 20000 for Telecom cases to evaluate the performance of ours and its peers under different offloading request loads.

5.2 Baseline Approaches

We consider a conventional approach (OLA [9]) and three state-of-the-art ones (MobMig [15], GD [13], and SO [12]) as baselines, where OLA and MobMig are online approaches while GD and SO are static ones. OLA is an online method that is able to learn the scheduling policy like ours by adopting a one-time-learning mechanism, but its scheduling policy is non-renewable once determined; MobMig is also an online best-fit-decreasing-based method that always schedules the offloaded mobile tasks to their nearby available edge servers with the shortest expected end-to-end delay; while GD is a static method that employs a greedy heuristic to schedule the offloaded mobile tasks to their nearby available edge servers with the highest remaining computation capability; and SO is a static method that employs IBM CPLEX Optimizer to solve the integer linear programming (ILP) problem of multiple-user multiple-server allocation in a batch-processing way.

Our approach and baselines are all implemented by Matlab R2020b, and the LP and ILP problems in our approach and its peers are solved by the Matlab built-in *linprog* and *intlinprog* functions. The experiments are conducted on a personal computer with macOS Catalina, 3.6 GHz Quad-Core Intel Core i3 processor, 8 GB memory, and 256 GB storage.

5.3 Responsiveness Evaluation

Results: Figure 5 compares the average end-to-end delays of our approach and baselines under different edge resources and request loads. It can be seen that

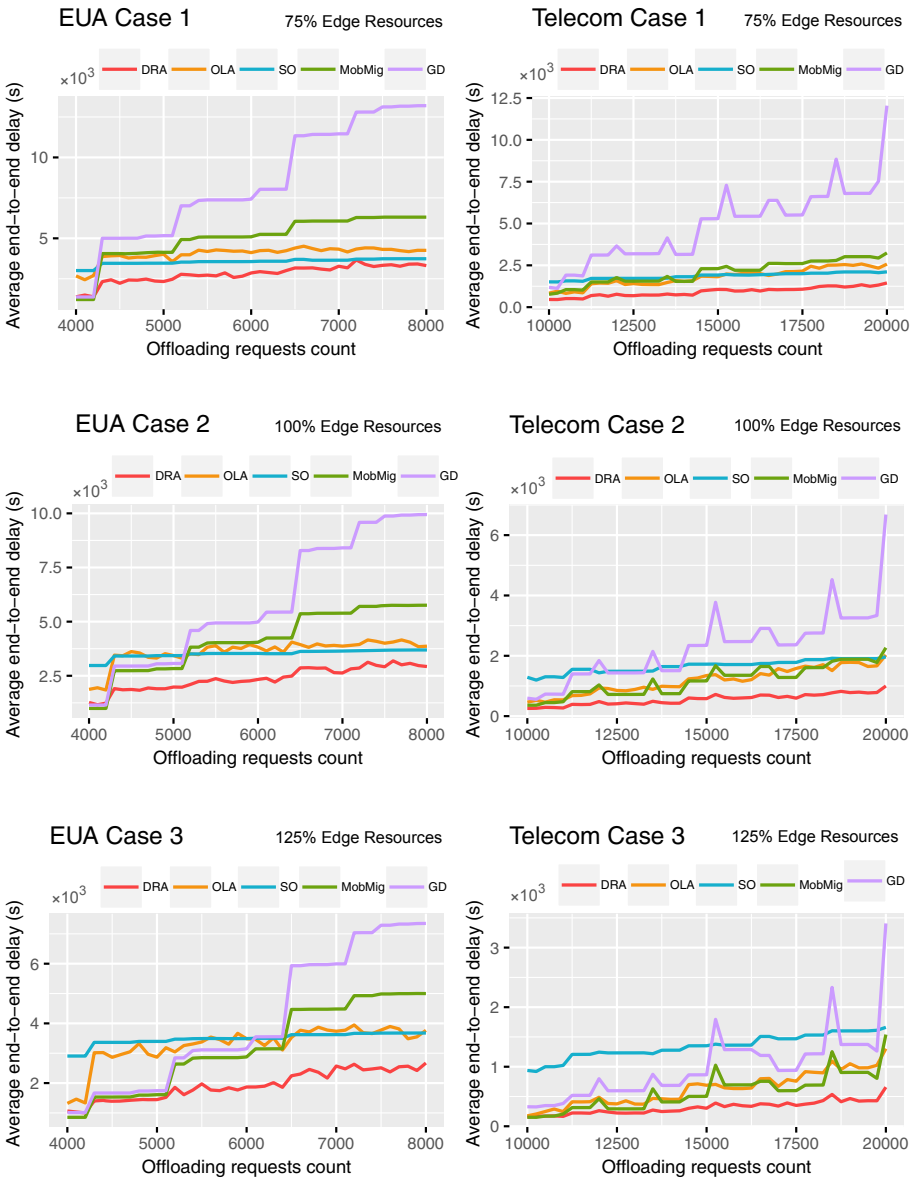

Fig. 5. Average end-to-end delay comparison.

our proposed DRA approach gets the lowest average end-to-end delay in the most of the cases of two datasets. More specifically, the proposed DRA can achieve a 35.72%, 32.89%, 42.26%, and 59.79% lower end-to-end delay on average compared with OLA, SO, MobMig, and GD respectively in EUA cases, and it can also achieve a 50.43%, 64.41%, 51.12%, 74.86% lower end-to-end delay on average than them in Telecom cases.

Analysis: DRA beats MobMig and GD because it is a learning-based approach that is capable of learning scheduling strategies continuously from past decisions, while the scheduling strategies of MobMig and GD are invariable regardless of the changes of the MEC environments. Though OLA is also a learning-based method, its one-time-learning mechanism can not capture the changes in the utility value of edge servers when faced with a highly dynamic MEC environments. While in our approach, the scheduling policies are continuously updated by a dynamic learning mechanism, the dynamicity of MEC environments are thus captured and that is why DRA outperforms OLA. SO method also shows an advantage over some baselines (e.g., MobMig and GD) in some cases, that is because it is a static method that always gets the optimal offloading decisions from the global view at the end of a batch window. However, its batch-processing mode may result in additional waiting time and it suffers from the decision time explosion problem when faced with an increasing number of requests.

5.4 Efficiency Evaluation

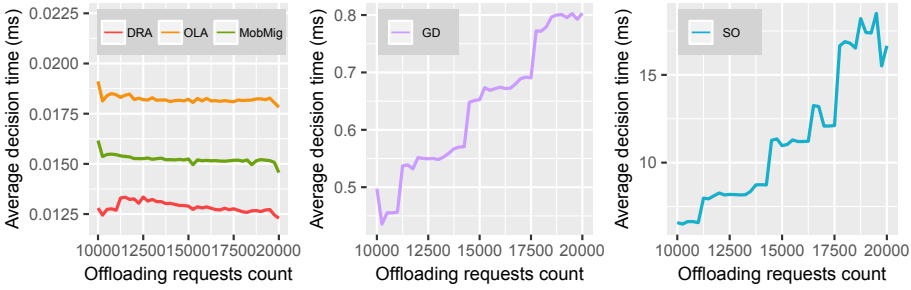

Fig. 6. Offloading decision time comparison.

Results: We also evaluate the efficiency of our approach and baselines by comparing their average decision time of each request. As shown in Fig. 6, DRA achieves the lowest decision time than other methods (on average, 9.59% lower than MobMig; 26.98% lower than OLA; 96.83% lower than GD; and 99.88% lower than SO). It can be seen that the average decision time of three online methods (i.e., DRA, OLA, and MobMig) keeps steady with the increasing of request count, while that figures for GD and SO methods show a rapidly increasing trend. Besides, the average decision time of all online methods is around 0.015 ms, while that figures for GD and SO methods are around 0.7 ms and 15 ms, which are two and three orders of magnitude higher than online ones, respectively.

Analysis: The time complexity of making an offloading decision of DRA, OLA, and MobMig methods are all $O(m \log m)$, where m is the number of edge servers, thus they are not sensitive to the increasing request count. The reason why the DRA method achieves a lower decision-making time than other online methods

lies in that its a decentralized approach where offloading decisions are made by end-users themselves locally instead of a centralized scheduler, the request forwarding delays are thus avoided and offloading decision time is shortened. The time complexity of GD is $O(n'm \log m)$, where n' is the number of requests that arrive in the same batch window, thus its decision time shows a growth trend with the increasing of request count. Similarly, the SO method also shows such a trend, but its global optimization strategy consumes more time.

6 Conclusion and Further Work

This paper targets at the online mobile task offloading in MEC environments. We consider the arbitrary arrival pattern of offloading requests and formulate the online MEC-oriented task offloading problem as a multi-dimension online integer linear programming problem. We employ a dynamic-learning mechanism and propose a decentralized reactive approach to solve it, the proposed approach is capable of continuously learning and updating scheduling policy to cope with the highly dynamic MEC environments. Case studies based on two real-world edge datasets have verified the effectiveness and efficiency of the proposed approach.

In our further studies, the following concerns will be addressed: 1) some time-series prediction and trajectories prediction methods could be utilized to predict the future request arrival rates and the request densities in different areas, which can be used to perform a load-balance and task-migration in advance to serve more users; 2) we only consider CPU tasks in this paper, GPU and CPU-GPU hybrid task offloading problem will be investigated in our future works; 3) more QoS metrics such as offloading monetary cost, reliability, and energy consumption of mobile devices will be considered, evaluated, and added to our system model.

Acknowledgements. This work is supported in part by the Graduate Scientific Research and Innovation Foundation of Chongqing, China (Grant No. CYB20062 and CYS20066), and the Fundamental Research Funds for the Central Universities (China) under Project 2019CDXYJSJ0022. The author gratefully acknowledges the support of K.C.Wong Education Foundation, Hong Kong.

References

1. Agrawal, S., Wang, Z., Ye, Y.: A dynamic near-optimal algorithm for online linear programming. Oper. Res. **62**(4), 876–890 (2014)
2. Alameddine, H.A., Sharafeddine, S., Sebbah, S., Ayoubi, S.: Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing. IEEE J. Sele. Areas Commun. **37**(3), 668–682 (2019)
3. Chen, L., Zhou, S., Xu, J.: Computation peer offloading for energy-constrained mobile edge computing in small-cell networks. IEEE/ACM Trans. Netw. **26**(4), 1619–1632 (2018)
4. Chen, M., Hao, Y.: Task offloading for mobile edge computing in software defined ultra-dense network. IEEE J. Sel. Areas Commun. **36**(3), 587–597 (2018)

5. Dai, Y., Xu, D., Maharjan, S., Zhang, Y.: Joint computation offloading and user association in multi-task mobile edge computing. IEEE Trans. Veh. Technol. **67**(12), 12313–12325 (2018)
6. Deng, S., Wu, H., Yin, J.: Mobile service provisioning. Mobile Service Computing. ATSTC, vol. 58, pp. 279–329. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-5921-1_8
7. Du, W., et al.: Service capacity enhanced task offloading and resource allocation in multi-server edge computing environment. In: 2019 IEEE International Conference on Web Services (ICWS), pp. 83–90 (2019)
8. Fang, Z., Lin, J.H., Srivastava, M.B.: Multi-tenant mobile offloading systems for real-time computer vision applications. In: Proceedings of the 20th International Conference on Distributed Computing and Networking, pp. 21–30 (2019)
9. Feldman, J., Henzinger, M., Korula, N., Mirrokni, V.S., Stein, C.: Online stochastic packing applied to display ad allocation. In: de Berg, M., Meyer, U. (eds.) ESA 2010, Part I. LNCS, vol. 6346, pp. 182–194. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15775-2_16
10. He, Q., et al.: A game-theoretical approach for user allocation in edge computing environment. IEEE Trans. Parallel Distrib. Syst. **31**(3), 515–529 (2019)
11. Huang, L., Bi, S., Zhang, Y.J.: Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Trans. Mob. Comput. **19**(11), 2581–2593 (2020)
12. Lai, P., et al.: Optimal edge user allocation in edge computing with variable sized vector bin packing. In: Pahl, C., Vukovic, M., Yin, J., Yu, Q. (eds.) ICSOC 2018. LNCS, vol. 11236, pp. 230–245. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03596-9_15
13. Lai, P., et al.: Edge user allocation with dynamic quality of service. In: Yangui, S., Bouassida Rodriguez, I., Drira, K., Tari, Z. (eds.) ICSOC 2019. LNCS, vol. 11895, pp. 86–101. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33702-5_8
14. Liu, C.F., Bennis, M., Debbah, M., Poor, H.V.: Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. IEEE Trans. Commun. **67**(6), 4132–4150 (2019)
15. Peng, Q., et al.: Mobility-aware and migration-enabled online edge user allocation in mobile edge computing. In: 2019 IEEE International Conference on Web Services (ICWS), pp. 91–98. IEEE (2019)
16. Rafique, W., Qi, L., Yaqoob, I., Imran, M., u. Rasool, R., Dou, W.: Complementing IoT services through software defined networking and edge computing: a comprehensive survey. IEEE Commun. Surv. Tutor. **22**(3), 1761–1804 (2020)
17. Sun, M., Xu, X., Tao, X., Zhang, P.: Large-scale user-assisted multi-task online offloading for latency reduction in D2D-enabled heterogeneous networks. IEEE Trans. Netw. Sci. Eng. (2020). https://doi.org/10.1109/TNSE.2020.2979511
18. Wang, S., Guo, Y., Zhang, N., Yang, P., Zhou, A., Shen, X.S.: Delay-aware microservice coordination in mobile edge computing: a reinforcement learning approach. IEEE Trans. Mob. Computi. (2019). https://doi.org/10.1109/TMC.2019.2957804
19. Wang, S., Zhao, Y., Huang, L., Xu, J., Hsu, C.H.: Qos prediction for service recommendations in mobile edge computing. J. Parallel Distrib. Comput. **127**, 134–144 (2019)
20. Wang, S., Zhao, Y., Xu, J., Yuan, J.: Edge server placement in mobile edge computing. J. Parallel Distrib. Comput. **127**, 160–168 (2019)

21. Xu, J., Chen, L., Zhou, P.: Joint service caching and task offloading for mobile edge computing in dense networks. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 207–215. IEEE (2018)
22. Yang, B., Cao, X., Bassey, J., Li, X., Qian, L.: Computation offloading in multi-access edge computing: a multi-task learning approach. IEEE Trans. Mob. Comput. (2020). https://doi.org/10.1109/TMC.2020.2990630
23. Yang, L., Zhang, H., Li, X., Ji, H., Leung, V.C.: A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing. IEEE/ACM Trans. Netw. **26**(6), 2762–2773 (2018)
24. Zhao, H., Deng, S., Zhang, C., Du, W., He, Q., Yin, J.: A mobility-aware cross-edge computation offloading framework for partitionable applications. In: 2019 IEEE International Conference on Web Services (ICWS), pp. 193–200. IEEE (2019)