Mobility-aware Tasks Offloading in Mobile Edge Computing Environment

Chunrong Wu, Qinglan Peng*, Yunni Xia*, and Jia Lee*
Software Theory and Technology Chongqing Key Lab
Chongqing University, Chongqing, China
{*qinglan.peng, xiayunni*}*@hotmail.com, lijia@cqu.edu.cn*

***Abstract*—In the mobile edge computing (MEC) paradigm, the underlying business processes of mobile applications can be modeled as application graphs, and mobile users are allowed to offload mobile tasks in these graphs to nearby edge servers to speed up their mobile applications. Nevertheless, various challenges, especially the quality of such a mobile task offloading in edge computing environment, are yet to be properly tackled. Most studies and related offloading strategies based on the assumption that mobile users are fully stationary when they are offloading tasks to edge servers. However, this is not realistic in real-world where edge users are keeping moving, which has a great impact on the success of task offloading and further affects the response time of mobile applications. In this paper, we take the mobility of edge users into consideration and employ a deep learning method to predict the future trajectories of mobile users. Then we develop an online method to solve the offloading problem based on the quality-of-service (QoS) and the predicted user trajectories in real-time. Experiments based on real-world user trajectories and edge server dataset show that our proposed approach can achieve lower offloading failure count and shorter response time than traditional ones.**

***Keywords*-Mobile Edge Computing; Mobile Task offloading; User Mobility; Mobile Application Graph;**

I. Introduction

Recent years have witnessed the prosperity of mobile devices and mobile applications. Nowadays, we are surrounded by different kinds of mobile devices including smartphones, tablets, wearable devices, *etc.,* and these pervasive devices have changed our daily lives a lot. For example, Augmented Reality (AR) and Virtual Reality (VR) applications have changed the way we interact with the real-world and cyberspace, and Artificial Intelligence (AI) applications make things much easier than before. However, it is true that mobile applications are becoming more and more heavy and complex, which poses a great challenge to the limited mobile resources, such as computing capability, memory, and storage, battery power, etc [1]. To overcome such a dilemma, traditional solutions, *e.g.,* [2], [3] and [4], try to offload the computation-intensive or memory-intensive mobile tasks into remote cloud to speed up the mobile applications. Nevertheless, even cloud computing features with unlimited resources that are capable of processing these offloaded mobile tasks vary quickly, the resulting high latency and additional communication overhead have a great impact on user-perceived quality-of-service (QoS).

Fortunately, recent advances in mobile communication, especially the 5th generation mobile networks (5G), prompt the emergence of mobile edge computing (MEC) and sheds light on the aforementioned problems. In MEC environment, edge infrastructures, *e.g.,* base stations, smart gateways, WiFi access points, or onboard computers, *etc.,* are equipped with a certain amount of computing capabilities. The underlying business processes of mobile applications can be modeled as application graphs, and the mobile users are thus allowed to offload the tasks in these graphs to nearby edge servers to boost the running of mobile applications [5].

However, mobile users are often with high mobility, which brings a great threat to the success of task offloading. An offloaded task will fail when its invoker moves out the sensor range of the edge server during the task execution. Therefore, how to capture the mobility of users and perform a smart task offloading strategy to avoid such failures to reduce the response time and offloading overhead has become the major problem. In this paper, we consider the mobility of end-users to make the offloading decisions, the user moving trajectories are predicted by employing a deep-learning-based method to acquire higher user-perceived QoS. We also consider a mixture edge cloud resource environment, where mobile tasks are allowed to offloaded into both edge servers or cloud servers, and develop an online method to make offloading decisions in real-time. We conduct a series of case studies based on real-world user trajectories and edge servers dataset, and the experimental results show that our approach is capable of dynamically capturing the mobility of mobile users and achieving lower offloading failures and shorter response times than traditional ones.

II. Related Work

Task offloading refers to the transfer of resource-intensive computational tasks to an external platform, it has been widely investigated for decades in many fields, such as offloading tasks to grid, cloud, ad hoc cloudlet, etc. As a new emerged computing paradigm, the core idea of MEC is that the computation should be performed near to data source instead of remote cloud [6]. It can also comply well

*Qinglan Peng, Yunni Xia, and Jia Lee are the corresponding authors of this work. This work is supported by the Fundamental Research Funds for the Central Universities under Grant 2019CDXYJSJ0022.

2379-1896/19/$31.00 ©2019 IEEE
DOI 10.1109/CANDAR.2019.00034

with preexisting computing infrastructures such as cloud computing or fog computing in a flexible and on-demanding way to fill the resource gap between mobile devices and remote cloud. According to the latest Cisco global cloud index [7], by 2021, 75% of data produced by humans, machines, or things will be stored or processed with the help of edge or fog computing.

Therefore, as one of the most promising computing paradigm, MEC has attracted a lot of attention, and more and more studies are focusing on the emergent problems, especially the mobile task offloading one. For example, Yang *et al.* [8] proposed an online method for the problem of allocating edge resources to execute the tasks offloaded by mobile users. They considered both the resource allocation and the computation partitioning problem together and proposed an online algorithm, which guarantees the real-time capability, to solve it, however, they do not consider the mobility of edge users. Da Silva Veith *et al.* [9] studied the problem of operator placement for stream application in edge computing. They considered a mixture edge cloud resource environment where operators are allowed to be placed in both edge servers or cloud servers. A heuristic method was presented to find the application patterns (fork, parallel, and join) and improve the response time of applications. However, they assume mobile users can access edge sources anytime and anywhere, which is unrealistic in the real world. Cai *et al.* [10] proposed an online method for the deployment of steam-based application operators on the edge computing environment. They first proposed a queue-theory based model to predict the response time of applications, then they formulated the operators' placement problem as an optimization problem and developed a response time-aware heuristic to yield the near-optimal solution. Like [9] and [8], they did not consider the mobility of users too. Deng *et al.* [3] proposed a mobile workflow task offloading method for mobile environment, they user Rand-Way Point (RWP) model to model the mobility of mobile users and design a Genetic Algorithm (GA)-based offloading method. However, their method is considered too time-consuming to be applied in a practical system.

A careful investigation into the aforementioned studies shows that the main limitation of existing work lies in the neglect of the mobility of mobile users. Existing works usually assume that the mobile users are stationary when they are offloading mobile tasks to edge servers and the communication between mobile users and edge servers are always reliable. However, this is unrealistic in real-world application scenarios, where mobile users are with universal mobility. Once mobile users move out of the transmission range of edge servers, the offloaded tasks will fail, and such failures will extend the response time of mobile applications and lead to a waste of edge computing resources. Therefore, mobility-aware tasks offloading approaches that can capture the mobility of edge users and make smart offloading decisions are in high need.

III. System Model and Problem Formulation

A. System Model

In the MEC environment, edge users are allowed to offload their computation-intensive tasks to edge server or cloud server to alleviate the limitation of mobile hardware resources. A mobile application usually consists of several functional modules, which work together in a business process way to achieve complex functionalities [8]. We use application graphs which orchestrated by functional modules to describe mobile applications, and they can be denoted by a 2-tuple $G = (T, D)$, where $T = \{t_1, t_1, ..., t_n\}$ is a set of tasks, a task is ready to be executed when all of its predecessor tasks are complete. We use function $S(t_i)$ to identify the size of computation amount of tasks, for example, $S(t_i) = 130m$ means there are 130 million instructions for task t_i. $S = \{d_{(i,j)} | i \neq j \wedge i, j \in [1, n]\}$ is the set data dependencies between two tasks, where $d_{(i,j)}$ represents that the output of task t_i is the input of task t_j. And the data transfer time can be evaluate as $d_{(i,j)}/\beta_{(u,e)}$, where $\beta_{(u,e)}$ is the bandwidth of communication channel between end user u and edge server e.

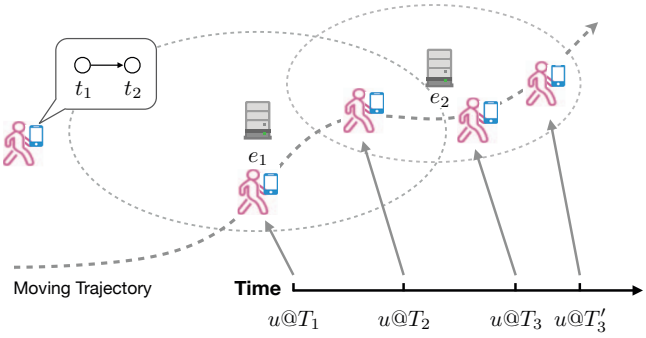


Figure 1: An example of task offloading in MEC environment.

Besides, we also consider a mixture edge and cloud resource pool in this paper, where mobile tasks can be offloaded into both edge server or cloud server. Edge servers are distributed around the end-users, they could be base stations, wireless access points, smart gateways, or even on-board computers, etc. Edge servers are heterogeneous with different core numbers, computing capability, memory (queue length), and transmission range. While cloud resources are always available to all end users but users may suffer from higher transmission latency. We use $P = \{e_1, e_2, ..., e_m, c_1, c_2, ..., c_p\}$ to denote the mixture edge and cloud resource pool, where $\{e_1, e_2, ..., e_m\}$ represents the available edge servers, while $\{c_1, c_2, ..., c_p\}$ represents the cloud service provided by different cloud providers. Note that, users can only be served within their transmission range

[5], and we use R_i to identify the transmission range of e_i. We use M_i to identify the core number of the i-th edge server, E_i and C_i the computing capability of i-th edge server and cloud server respectively, which can be measured by million instructions per second (MIPS).

Fig. 1 shows such a mobile task offloading example, where user u want offload two tasks in its application graph to nearby edge servers. In this figure, the red dotted line is the moving trajectory of u and there are two edge servers nearby. Edge server e_1 is available for u at time T_1, server e_1 and e_2 are both available for u at time T_2 and only server e_2 is available at time T_3. At time T_1, only server e_1 is available for user u, therefor u decides to offload task t_1 to e_1. Suppose t_1 will complete at T_2 when u was still in the transmission range of e_1, the offloading of t_1 is success. Then, u decides to offload task t_2, at that time, *i.e.,* T_2, both edge servers e_1 and e_2 are available for user u. Suppose edge server e_1 is powerful than e_2, offloading task t_2 to e_1 and e_2 will complete at T_3 and T_3' respectively, and $T_3' > T_3$. As shown in the figure, if we offload task t_2 to server e_1, it will complete at time $T_3$when the user is out of the transmission range of edge e_1, which means offloading is failed and we have to re-offload t_2. In this way, not only response time is enlarged but also edge resources are wasted. Therefore, e_1 is powerful or more cost-effective than e_2, but once considering the universal mobility of end-users, e_2 is the best choice for task t_2 as shown in Fig. 2. And that is why we consider the mobility of the user and employ a machine learning method to capture it.

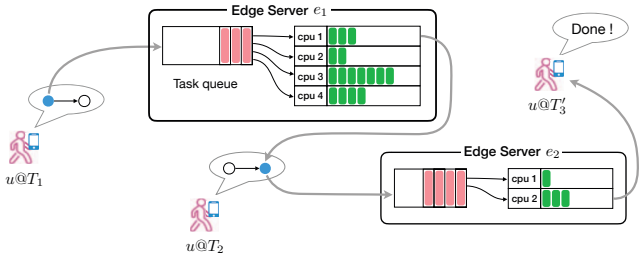


Figure 2: Offloading tasks to edge servers.

As shown in Fig. 2, each edge server has a limited task queue, the offloaded tasks which can not be processed immediately will be stored in the queue temporarily, we use Q_i to identify the size of tasks in the task queue of e_i, L_i the maximum capacity of task queue of e_i. And we consider that the edge servers are equipped with multi-core processors that can process multiple mobile tasks simultaneously, the tasks in the queue are served in an FCFS (first-come-first-served) way [11]. Therefore, the estimated response time of offloading task t_i to mixture edge cloud can be estimated as:

$$\tau(t_i) = \begin{cases} \frac{S(t_i)+Q_j}{E_j \times M_j} + \frac{d_{(i,k)}}{\beta_{(u,e_j)}} & l(t_i) = 1, e_j = f(t_i) \\ \frac{S(t_i)}{C_j} + \frac{d_{(i,k)}}{\beta_{(u,e_j)}} & l(t_i) = 0, c_j = f(t_i) \end{cases} \quad (1)$$

where $l(t_i) \in \{0, 1\}$ is a boolean function to identify whether t_i is going to be offloaded to edge server, and we use function $f(t_i)$ to identify the server (could be either edge or cloud) that t_i is going to be offloaded into.

B. Problem Formulation

Given a mobile user u who want to offload mobile tasks in their application graph G to a mixture edge and cloud resource pool P, we have the interest to know which task is suitable to be offloaded to which edge or cloud resource with the movement of end-user. Therefore, the aforementioned problem can be formulated as follow:

$$Min : h(G) \quad (2)$$

$$s.t : dist(u, e) < R_i \quad \forall\ t_i \in G.T \wedge l(t_i) = 1 \quad (3)$$

$$S(t_i) + Q_i \leq L_i \quad \forall\ t_i \in G.T \wedge l(t_i) = 1 \quad (4)$$

$$e = f(t_i)$$

$$i \in \{1, 2, ..., n\}$$

where $h(G)$ is the actual response time of graph G, $dist(u, e)$ the function to identify the distance between user and edge server, it can be obtained by aggregating all response time of tasks in a graph through a reduction method [12]. The offloading target of this paper is to reduce the response time of the total application graph as shown in E.q (2). E.qs (3) and (4) are two constraints, where E.q (3) guarantees that users can only offload tasks to edge servers which users are in their transmission range, while E.q (4) ensures that tasks are only allowed to be offloaded to those edge servers whose task queue is not full.

IV. Proposed Approach

In this section, we first introduce how we employ a machine learning method, *i.e.,* social generative adversarial networks (SGAN) [13], to get the predicted mobile user trajectories to estimate their corresponding available offloading time. Then we propose a mobility-aware online offloading method to select ideal resources for mobile tasks in real-time.

A. Social GAN for Mobile User Trajectory Prediction

Recent studies, *e.g.,* [14] and [15], have pointed that there is a potential up to 93% average predictability in user mobility. For example, Fig. 5 shows the collected human trajectories on a campus. We can see that most trajectories share the similarity and regularity patterns. And such similarity, periodicity, and regularity can be formally and properly described with novel methods. In this paper,

we employ SGAN to predict the trajectories of mobile users to achieve mobility-aware task offloading.

Based on generative adversarial networks (GAN), SGAN is a kind of deep-learning-based trajectories prediction method, it not only considers the target's historical trajectories as the basis of prediction but also exploits the moving activities of other humans around the target to further improve the accuracy of prediction. SGAN takes the historical historical trajectories of mobile users in a certain area as input and predicts their future trajectories. The historical trajectories can be described as $\mathbf{X} = \{X_1, X_2, ..., X_n\}$, where $X_i = (\overrightarrow{x_i}, \overrightarrow{y_i})^{t_h}$ denotes the i-th user' position vector in time steps $t_h = \{1, 2, ..., c\}$, c denotes the index of current time step. The predicted trajectories can be described as $\hat{\mathbf{Y}} = \{\hat{Y}_1, \hat{Y}_2, ..., \hat{Y}_n\}$, where $\hat{Y}_i = (\overrightarrow{\hat{x}_i}, \overrightarrow{\hat{y}_i})^{t_f}$ denotes the i-th user' predicted position vector in time steps $t_f = \{c, c+1, ..., p\}$, p denotes the length of prediction-step. The ground truth trajectories in time steps t_f is denoted as $\mathbf{Y} = \{Y_1, Y_2, ..., Y_n\}$.

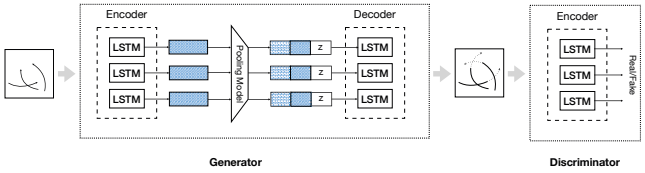


Figure 3: Structure of Social GAN.

Recent studies, *e.g.,* [14] and [15], have pointed that there is a potential up to 93% average predictability in user mobility. For example, Fig. 5 shows the collected human trajectories on a campus. We can see that most trajectories share the similarity and regularity patterns. And such similarity, periodicity, and regularity can be formally and properly described with novel methods. In this paper, we employ SGAN to predict the trajectories of mobile users to achieve mobility-aware task offloading.

Based on generative adversarial networks (GAN), SGAN is a kind of deep-learning-based trajectories prediction method, it not only considers the target's historical trajectories as the basis of prediction but also exploits the moving activities of other humans around the target to further improve the accuracy of prediction. SGAN takes the historical trajectories of mobile users in a certain area as input and predicts their future trajectories. The historical trajectories can be described as $\mathbf{X} = \{X_1, X_2, ..., X_n\}$, where $X_i = (\overrightarrow{x_i}, \overrightarrow{y_i})^{t_h}$ denotes the i-th user' position vector in time steps $t_h = \{1, 2, ..., c\}$, c denotes the index of current time step. The predicted trajectories can be described as $\hat{\mathbf{Y}} = \{\hat{Y}_1, \hat{Y}_2, ..., \hat{Y}_n\}$, where $\hat{Y}_i = (\overrightarrow{\hat{x}_i}, \overrightarrow{\hat{y}_i})^{t_f}$ denotes the i-th user' predicted position vector in time steps $t_f = \{c, c+1, ..., p\}$, p denotes the length of prediction-step. The ground truth trajectories in time steps t_f is denoted as $\mathbf{Y} = \{Y_1, Y_2, ..., Y_n\}$.

B. Mobility-aware Offloading Method

As discussed early, in the MEC environment, edge users are usually with universal mobility. Therefore, edge users' resource pool and the QoS (*e.g.,* workload, expected response time, transmission distance, and bandwidth, etc.) of edge or cloud resources in the pool are changing all the time, which pose a great challenge to the real-time capability of offloading methods. To this end, in this paper, we propose an online offloading strategy whose offloading decisions are made on the fly to cope with such a highly dynamic and time-critical MEC environment.

Algorithm 1: Online task offloading procedure

Input: User u; Mobile application graph $G = (T, D)$

1 $R \leftarrow$ get all tasks in G without precursors ;
2 $E \leftarrow \emptyset$;
3 **while** $|T| \neq 0$ **do**
4 **foreach** $t \in R$ **do**
5 $r \leftarrow$ RTPred(u, t) ;
6 schedule task t to resource r ;
7 $R \leftarrow R - \{t\}$;
8 $E \leftarrow E \cup \{t\}$;
9 **foreach** $t \in E$ **do**
10 **if** t *is finished* **then**
11 $E \leftarrow E - \{t\}$;
12 $T \leftarrow T - \{t\}$;
13 $ts \leftarrow$ get all tasks in G without precursors ;
14 $R \leftarrow R \cup ts - R \cap ts$;
15 **else if** t *is timeout* **then**
16 $r \leftarrow$ RTPred(u, t) ;
17 reschedule task t to resource r ;

The procedure of our approach is shown in Algorithm 1, where R represents the set of tasks that are ready to be offloaded, E the set of tasks that have been already offloaded but are yet to finish. The procedure starts with finding the tasks which are ready to be offloaded in an application graph G, and we put them into R (as shown in line 1). In the beginning, there are no tasks under executing, therefore we initialize E with $\emptyset$ (as shown in line 2). Then, it enters the main loop (as shown in lines 3-17) until all tasks in the graph are complete. There are two main part in the main loop, the first is scheduling tasks in R to suitable resources (as shown in lines 4-8), in this step, the mobility-aware offloading strategy we proposed is invoked to get the most suitable resource for task t. The other is checking whether there are tasks under executing are finished (as shown in lines 9-17). Note that, if tasks in E complete successfully, they will be removed from E and T, and the process will find its successor tasks which are ready to be

offloaded and put them into R. If the tasks in E do not receive a response in expected response time, we regard these offloading are fail we and will re-offload them. The re-offloading of tasks will extend the response time and waste computing resources, therefore, smarter offloading strategies will exploit available information as much as possible to avoid rescheduling to achieve lower response time and better user-perceived experience.

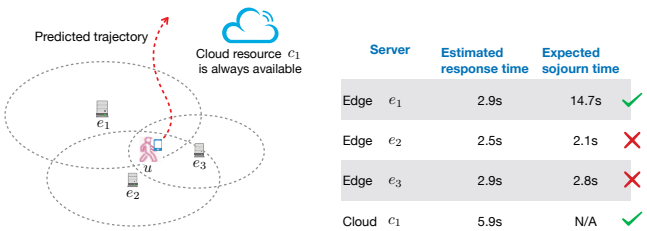
Algorithm 2: RTPred

Input: User u; Mobile task t to be offloaded
Output: resource r for task t

$P \leftarrow$ perceive all available edge servers and cloud services;
foreach $e \in P$ **do**
  $e.t \leftarrow$ evaluate the expect response time for task t executed on resource e according to E.q (1) ;
  if *e is edge resource* **then**
    $e.s \leftarrow$ evaluate the expected sojourn time for mobile user u in the sensor range of edge server e according to E.q (6) ;
    if $e.t \leq e.s$ **then**
      $C \leftarrow C \cup \{e\}$;
  else
    $C \leftarrow C \cup \{e\}$;
$r \leftarrow$ find the resource in C with the lowest response time ;
return r ;

The procedure of our proposed mobility-aware offloading method, short for RTPred, is shown in Algorithm 2. It starts with constructing the mixture edge and cloud resource pool for user u at current time, which aims to guarantee the real-time capability of our method. C stands for the qualified candidate resource set for task t. For each edge server in P, we first estimate the response time of task t on it according to E.q (1) and the sojourn time of user u in its sensor range according to E.q (6). Then, only those edge servers whose estimated sojourn time of u in their range are longer than the response time of t offloaded into them are considered qualified edge servers and put into C. Due to the cloud resources are always available for every mobile users, we evaluate their corresponding response time and put them into C. And finally, resource with the lowest estimated response time will be selected and return as the final offloading destination for task t.

Fig. 4 shows such an example, it can be seen that there are three edge servers available for user u at current time, and u want to offload task t to edge server or cloud server. But which resource is the most suitable for t? The right table shows the estimated response time of t offloaded to different resources, and we can see that edge server e_2 and e_3 are



Server		Estimated response time	Expected sojourn time	
Edge	e_1	2.9s	14.7s	✓
Edge	e_2	2.5s	2.1s	✗
Edge	e_3	2.9s	2.8s	✗
Cloud	c_1	5.9s	N/A	✓

Figure 4: An example of resource selection.

excluded because the estimated response time of t on them exceed the sojourn time of u in their sensor range. It is of great possibility to offload fail for e_2 and e_3. On the contrary, edge server e_1 meets the requirement and cloud server c_1 always available for every users, therefore we put e_1 and c_1 into our candidate servers set. Finally, t will be assigned to the qualified server with lowest estimated response time, *i.e.,* e_1.

Suppose there are n edge and cloud resources in the resource pool, the time complex of estimating the QoS of available edge and cloud resources is $O(n)$, the time complex of finding the most suitable resource is $O(n \log n)$. Therefore the time complex of our RTPred method is $O(n \log n)$, which guarantees a good real-time capability to cope with the high dynamic MEC environment.

V. Experiments and Analysis

In this section, we conduct a series of case studies based on real-world user trajectories and edge server performance datasets to validate the effectiveness of our approach in terms of offloading failure count and total response time.

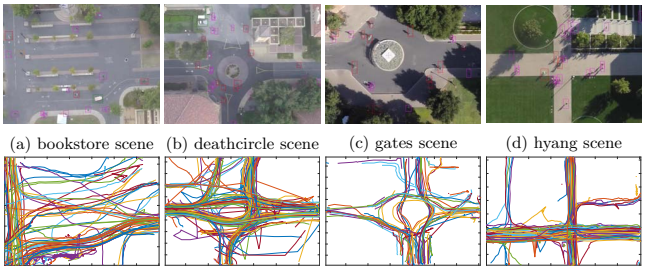
We consider Stanford Drone dataset[1] as the experimental scenes. In this dataset, all pedestrians' movement trajectories in a certain area on Stanford campus are recorded for consecutive periods. We choose *bookstore*, *gates*, *deathcicle*, and *hyang* these four scenes with varying crowd density to carry out our experiments. The aerial views and users' trajectories of these four scenes are shown in Fig. 4 and the details of them are shown in Table 1.

Table I: The details of experiment scenes.

Scene name	Scene area	Trajectory record length	User number	Edge server number
bookstore	2484.66m^2	7m25s	237	25
deathcicle	2693.29m^2	6m57s	872	50
gates	2351.42m^2	1m13s	111	50
hyang	2697.12m^2	6m49s	197	30

We consider the Advantech EIS-D210[2] (*i.e.,* 38,46 MIPS per core at 1.5GHz, 4 GB of RAM, and with IEEE 802.11

[1] http://cvgl.stanford.edu/projects/uav_data/
[2] http://www.advantech.com.cn/products/071c0784-5e9b-4bb8-bb07-2cb06da757a1/eis-d210/mod_5e343ae5-0935-4a82-8cf4-cd0a30ec7d5b



(a) bookstore scene (b) deathcircle scene (c) gates scene (d) hyang scene

(e) users' trajectories in bookstore scene (f) users' trajectories in deathcircle scene (g) users' trajectories in gates scene (h) users' trajectories in hyang scene

Figure 5: Experiment scenes.

WiFi standard) as the edge servers and the Del PowerEdge R840 (*i.e.,* 317,90 MIPS per core at 3.0GHz and 768 GB of RAM) as the cloud servers. Edge servers are randomly deployed in the above scenes with transmission ranges of 30-50m, and cloud servers are always available for every mobile user. The latency of the LAN communications between edge servers and users is obtained from a uniform distribution with 0.015 ms and 0.8 ms, and bandwidth is set to 100 Mbps. While the latency of the WAN communications between cloud servers and users is obtained uniformly between 65 ms and 85 ms, and its bandwidth is set to 1 Gbps [9].

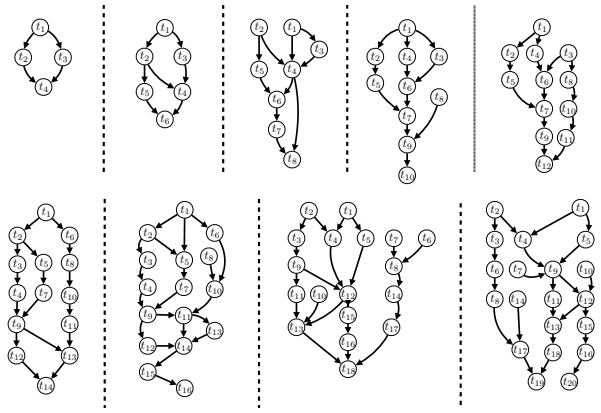


Figure 6: Application graphs.

As [9] did, we generate a set of application graphs using a Python library[3] as shown in Fig. 6. The task numbers of the application graph set are range from 4 to 20, the size of tasks is assigned by using a uniform distribution with a mean value of 4600m instructions, the size of data transmission for each task is assigned in the same way with mean value 750 kilobyte [8]. We consider Load Balance (LB) [16], Response-Time-Aware (RTA) [10], and Greedy offloading strategy (GD) as baselines:

- **LB** [16]: it evaluates the load of available edge servers at first, and then offloads tasks to serves whose load is relatively low to achieve load balance;
- **RTA** [10]: it is a kind of operator placement strategy in MEC environment, but can also be used in mobile task offloading. It first maps tasks to servers randomly, then use the proposed queue-theory-based model to estimate every path in the graph, and finally improve the longest path in the graph to reduce the response time.
- **GD**: it always offloads tasks to the nearest edge server to avoid offloading failure to save time.

In our experiments, edge servers are randomly distributed in the four different scenes, humans who move in these areas are seen as mobile users, and they are allowed to offloading tasks to the mixture edge and cloud resource pool simultaneously. We conduct each case for over 50 times and the average offloading failure count and the response time are recorded.

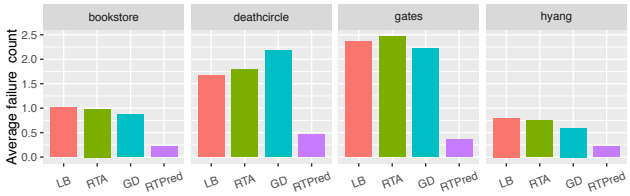
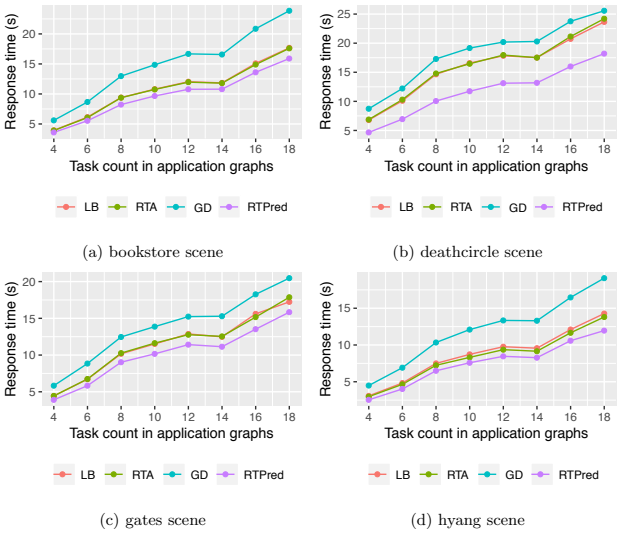


Figure 7: Average failure count evaluation.

We first evaluate the average failure count of different methods in four scenes. In this experiment, Edge users' application graphs are randomly chosen from the above 9 graphs with different task count. As shown in Fig. 7, it can be seen that our approach can significantly reduce the offloading failures compared with its peers in all cases (*e.g.,* 78.16% failures reduced than LB on averages in four scenes; 78.68% failures reduced than RTA, and 78.23% failures reduced than Greedy). That is because our method considers the mobility of users and leverages a learning-based method to capture the moving trend of edge users. The predicted user trajectories can help us filter out those edge servers with less available time to avoid potential offloading failures, and the total response time of application graphs are thus reduced.

We also investigate the average response time of our method and its peers in terms of different application graphs at four scenes. As shown in Fig. 8, it can be seen that our method can achieve lower average response time in all four scenes (*e.g.,* 15.53% lower than LB on average; 14.81% lower than RTA on average; and 33.87% lower than Greedy on average). That is because the proposed mobility-aware approach can significantly reduce the offloading failures as shown in Fig. 7, and the reduced failures is capable of further reducing the total response of application graphs. Besides, the consideration of the mixture edge and cloud resource pool in our approach also helps a lot. If there are no suitable edge resources to accommodate edge users' tasks, offloading them to cloud, which may suffer from high latency and additional communication overhead, is always better than

[3] https://gist.github.com/bwbaugh/4602818



(a) bookstore scene (b) deathcircle scene

(c) gates scene (d) hyang scene

Figure 8: Average response time evaluation.

offloading them to the wrong edge servers, which not only impacts the user-perceived QoS but also wastes the limited edge computing resources.

VI. Conclusion and Further Work

In this paper, we target the mobile task offloading problem over the MEC environment. We take the mobility of edge users into consideration and employ a deep-learning-based method, *i.e.,* SGAN, to capture the moving trend and predict the future trajectories of users. We also consider mixture resource pool where users are allowed offload mobile tasks to both edge servers and cloud servers, features of both edge computing and cloud computing are exploited in this way. And finally, a mobility-aware online task offloading algorithm is proposed to find the most suitable resource for every task in an application graph in real-time.

As future works, the Service-Level-Agreement (SLA) constraints (both soft and hard ones) will be concerned and the corresponding task offloading methods or algorithms will be introduced to reduce the SLA-violation rate. Besides, more QoS metrics, including the reputation and reliability of edge servers, will be well investigated and considered in our system model.

References

[1] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, p. 111, 2019.

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

[3] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, 2015.

[4] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, 2015.

[5] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 230–245.

[6] 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.

[7] C. V. Networking Index, "Forecast and methodology, 2016-2021, white paper," *San Jose, CA, USA*, vol. 1, pp. 0–4, 2016.

[8] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang, "Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds," *IEEE Transactions on Services Computing*, 2019.

[9] A. da Silva Veith, M. D. de Assuncao, and L. Lefevre, "Latency-aware placement of data stream analytics on edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 215–229.

[10] X. Cai, H. Kuang, H. Hu, W. Song, and J. Lü, "Response time aware operator placement for complex event processing in edge computing," in *International Conference on Service-Oriented Computing*. Springer, 2018, pp. 264–278.

[11] C.-F. Liu, M. Bennis, and H. V. Poor, "Latency and reliability-aware task offloading and resource allocation for mobile edge computing," in *2017 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2017, pp. 1–7.

[12] Y. Xia, Q. Zhu, Y. Huang, and Z. Wang, "A novel reduction approach to analyzing qos of workflow processes," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 2, pp. 205–223, 2009.

[13] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2255–2264.

[14] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, "Limits of predictability in human mobility," *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010.

[15] Q. Peng, Q. He, Y. Xia, C. Wu, and S. Wang, "Collaborative workflow scheduling over manet, a user position prediction-based approach," in *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Springer, 2018, pp. 33–52.

[16] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.