A New Mobile Edge Computing Simulation Model for Offloading Decision

Zijian Li, and Yan Wang

Abstract—Mobile Edge Computing (MEC) has been seen as a promising paradigm providing services of high quality to mobile users. In this paper, we propose a general single-server-multi-BS Mobile Edge Computing scheme that allows mobile devices to leverage edge servers just as soon as they encounter offloading tasks. And a simulation model concerning space and time is proposed, which allows mobile users to travel between different Base Station serving zones. Rather than treat each offloading request as from user equipment that only shows up one time, mobile devices in our model can keep the impact of previous offloading decisions. Our simulation model is dynamic and random, just like the real physical world. For example, offloading tasks are randomly generated with the probability continuously varying. And we model the space factor of MEC, allowing Users move between different service areas. Formulating the offloading problem as a partially observable Markov decision process, we test full local, full offloading, and a greedy decision policy, showing some characteristics of our simulation model.

Index Terms—Mobile edge computing, computation offloading, Quality of Service, Markov decision process

1 Introduction

In recent years, smartphones, laptops, and tablet devices have become unprecedentedly popular; computing-intensive application services such as virtual reality, natural language processing, and some complex interactive software have been used more and more widely; users' Quality of Service (QoS) requirements continue to increase [1].

Although electronic devices have been developing rapidly in the past for a long time as described by Moore's Law, nowadays people still may encounter situations where computing power cannot keep up with our demands. And generally speaking, the computing tasks that a user's mobile device can complete are also limited by its battery capacity quite often. This gave birth to the computing paradigm of cloud computing. Cloud computing not only allows user equipment (UE) to utilize a large number of storage resources of the remote central cloud, but also can offload part of the computing tasks from the local to the central cloud server for completion. By transmitting input data and algorithm programs to the cloud, using high-performance cloud servers to complete computing tasks, and then sending the results back to UE, local energy consumption and computation latency shall be reduced, thereby improving QoS [2]. However, because cloud servers are generally concentrated in one place or a few places, transmission and communication costs are high for many users to perform cloud computing. That is to say, the delay and energy consumption of offloading the calculation to the cloud and transmitting the results back is relatively considerable.

Edge computing is widely regarded as a new paradigm that complements cloud computing, and the earliest concept of edge computing (cloudlet) was proposed in 2009 [3]. The idea behind it is to deploy servers with high computing performance in the network close to UE. Users of mobile devices can offload computing tasks to an edge server nearby through radio access networks (RAN), with less delay when offloading the data compared to Cloud computing. Edge computing is gaining more and more attention recently. And many researchers believe that it will play an important role with the popularization of next-generation 5G mobile communication technology. Due to the limited computing power of mobile edge computing servers (MECS) and the complexity of wireless transmission channels, the offloading decision method has a decisive effect on whether adopting MEC can improve QoS, as inadvisable decisions may lead to longer waiting and more energy waste.

The offloading decision is one of the key technologies of edge computing. At present, the main research approach is still via modeling and simulation. Although there are a large number of studies accumulated in the past decade, they tend to ignore the continuous impact of one offloading decision to the corresponding UE, as they often consider each user device as just demanding this only one task for offloading decision. Moreover, one conspicuous element of mobile edge computing application scenario is that mobile users can move, which leads to the number of user devices in one MEC service area or Base station (BS) service area varying, and a user device that leaves one service area can enter its adjacent ones, thus the best offloading decision strategy may change. Unfortunately, until now seldom work have imported the geographic aspect of MEC or a changeable number of User devices [4]. Meanwhile, for every user, there is a roughly periodical fluctuation of the offloading demand, with the period of a day. For example, people generally get more computing tasks in the morning and afternoon while less at noon and midnight, which urges good offloading strategies to have the ability to handle a variable environment.

In this paper, we bring the edge computing research soci-

• *Zijian Li and Yan Wang are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, P.R.China. E-mail: lizijian@buaa.edu.cn and w-yan@buaa.edu.cn*

ety a novel MEC simulation model. It reproduces the MEC features in the real-world discussed above to our best efforts. And We formulate the offloading decision problem as a partially observable Markov decision process (POMDP), which can be handled by common decision algorithms [5], [6].

1.1 Related work

Generally speaking, the first thing to consider when making an offloading decision is whether or not to offload. Once you decide to offload, then you need to discuss how many computing tasks to offload and which computing tasks to offload [7]. There are two types of MEC offloading according to whether tasks are allowed to be split and executed in different devices or not, namely partial offloading and full offloading(sometimes referred to as integral offloading, binary offloading, or all offloading). Partial offloading could be a relatively complicated process affected by many factors, and even some tasks cannot be split into several parts that are executed locally and remotely in the server. It is necessary to consider the separability of tasks and the dependencies of the sub-tasks when using partial offloading. Most of the latest studies are only discussed at the level of "whether to offload", in other words, full offloading [2], so as our work.

When the UE decides whether to offload a certain computing task, it often compares the time and energy consumption required when it is offloaded or not, in order to provide users with higher QoS. Some of the former researches aim to achieve lower delay solely while meeting the power constraints [8]. And there are others attempting to minimize energy consumption with the time constraints given [9]. Moreover, jointly optimizing time and energy is another option, e.g., in [10], a weighted total cost of energy, computation, and delay is optimized.

In addition, with the increase in the number of mobile users, it will reduce more overall workload for decisions using decentralized computation offloading strategy and improve the scalability of the MEC system. This decentralized computation offloading approach, handling the decision-making work to each UE rather than all by the central MECS, is attracting more and more research attention nowadays [11], [12], [13], [14]. Zhao Chen et al. leave the task of offloading decision-making for edge computing to each UE itself [13]. In the single-server single-base station scenario they have studied, each UE makes its offloading decision independently based on its observation of the system state variable that the UE can get access to, including the task received, Signal to Interference plus Noise Ratio (SINR), and Channel State Information (CSI). The UE's decision output is a two-tuple, representing the calculation performed locally or in the MECS. It can be known from the previous research that the decentralized offloading decision can improve the scalability of the system. But if each UE makes offloading decisions independently, without awareness of the edge server usage and other UEs offloading requirements, it is unlikely to achieve joint optimization of the whole system.

Traditionally, when we use MDP to model an edge computing offloading decision process, the decision moment is fixed no matter when the task is coming [15]. This fixed decision periods modeling suffers from additional latency due to the waiting time between task coming and decision making. And most of the works assume that the decision epoch is long enough to finish executing a task, so that the offloading decision in the past will not influence future states [16]. In a resource-constrained MEC scenario with computationally intensive offloading tasks, the assumption may be violated.

1.2 Contribution

The first contribution of this work is that the simulation model built is practical due to its randomness and resources limitations. The simulation model takes into account limitations of multiple resources, including the local computing power and communication capabilities of mobile devices, MECS computing power, and limited base stations; And many parameters in the model are randomly generated mostly from uniform distributions. The task density of each user varies with time, in a way that "overcomes the law of large numbers", which will be explained in detail later.

Secondly, we introduce the concept of BSs' service zones and the movement of user devices in our simulation model. Mobile users are allowed to move between the service zones of each base station. And each time a user moves to an area that cannot be covered by any base stations, its feature parameters are reset. This somehow functions the same effect of generating new users and deleting old ones, which gives the model more variety and Variability.

Thirdly, we propose the offloading problem as a standard partially observable Markov decision process with variable decision cycles. And then some modifications of the standard Bellman equation to fit our problem are discussed. Moreover, we test our model in three basic offloading naïve policies: full offloading, full local, and greedy decision methods, with their experiment results illustrated.

The rest of the paper is organized as follows. We introduce our single-server multi-BS MEC model in Section 2, where the composition of task delay and energy consumption in both local and offloading modes is discussed in detail. In Section 3, the offloading decision-making problem is formalized as a partially observable Markov decision process, and its state, action and reward are given, so that the decision-making problem faced can be solved in a standard manner. At last, numerical results and some analysis are shown in 4.

2 System modeling

In a mobile edge computing scenario, for randomly arriving computing tasks, the decision may result in them being executed locally or offloaded to an edge server. Both these two situations will be discussed in this section, along with the introduction of the communication model and the computation model.

Our model is aimed at modeling the offloading problem of edge computing with a single server and multiple base stations in the scenario of a single cell with multiple users. The computing power of mobile devices is set much smaller than that of edge servers, where the term computing power in this paper is synonymous with CPU cycles per second that indicate computing speed.

Generally speaking, the purpose of offloading computing tasks to a MECS is to pursue lower latency and energy consumption than relying solely on local computing resources. Every computing task, e.g. the one denoted T_n, can be completely described by two attributes the amount of calculation c_n and the amount of data d_n. The amount of calculation representing the CPU cycles required to complete this task is generally not device-sensitive. The amount of data d_n represents the summed size of both the input and the program of this task that the UE needs to transmit to the MECS in the up-link when the task is offloaded.

Mobile users are allowed to move randomly between the service areas of the base stations. Each time the user moves to an area that cannot be covered by all base stations, its characteristic parameters are reset. The task density of each user in a way that "overcomes the law of large numbers" changes with time, and the simulation model has greater randomness and variability.

2.1 Communication model

In our proposed simulation model, there is only one MECS. And its service region is divided into several different zones with varied access to Base stations. When a task is offloaded to the MECS, it has to be transmitted via a BS. As for the communication network of the system, it is assumed that the frequency division multiple access method is used. For simplification, every BS is assumed to have an identical set of channels with different channel gains. Each channel can only be used for data transmission of offloading tasks for one mobile user at a time. Further, there are multiple small base stations (SBS) with different distances from the MECS and one Macro Base Station (MBS) in the same place with the MECS in the network. Fig. 1 shows a mobile device offloading its computation tasks in the proposed MEC communication network structure, where each BS has two channels.

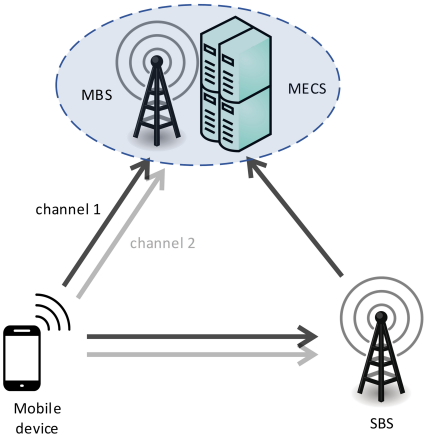

Fig. 1. Mobile edge computing offloading

Trying to take the concept of areas and UE movements into consideration in our simulation model, each user device is given an attribute to tell which SBS service zone it is located in. And we suppose each that zone is covered by only that single SBS, without overlap. An example with three SBS to illustrate this is shown in Fig. 2, where there are four possible zones that a device can be in. Our simulation is based on this four zones model without loss of generality. Every second, each UE shall move to other zones or not with the probability decided by a predefined matrix. This transition probability matrix of movements is set as in Section 4.1.

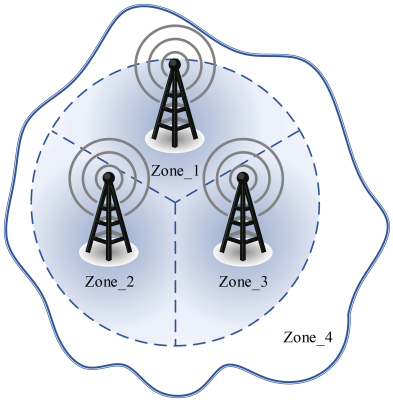

Fig. 2. Small Base Station service zones

For simplicity to illustrate the mathematical relations in the simulation model, let's suppose that one User device denoted as UE_n faced a task $T_n(d_n, c_n)$ for offloading decision now [17], [18], where d_n and c_n are described as in the beginning of this chapter. If UE_n selects offloading calculation to perform task T_n after the decision, the entire offloading computation process will be divided into three steps, that is, data being uploaded to MECS, MECS completing the offloaded task, and the backhaul of MECS execution results to UE_n. Given that the data returned by the calculation result is generally much smaller than the data uploaded, the time spent in the process of result backhaul is ignored here.

The process of uploading data to the MECS can be divided into two parts: from UE to BS and from BS to the MECS. From (1), the data uploading speed from UE_n to BS, r_n, can be derived [19], where W is the bandwidth of the wireless channel; P_n^{send} is the transmission power of UE_n when uploading data; g_n denotes the channel gain of the allocated channel. Here each base station has multiple channels with different gains to choose from. And σ^2 is the power of the background Gaussian white noise.

$$r_n = W \log_2(1 + \frac{P_n^{send} g_n}{\sigma^2}) \tag{1}$$

Then the time cost of uploading task data from UE_n to BS, $t_n^{o,t1}$, can be expressed as below.

$$t_n^{o,t1} = \frac{d_n}{r_n} \tag{2}$$

And the time it will take to transmit data from the base station to the server is shown in (3). Among it, φ_n is the parameter that measures the transmission rate of the relay communication base station to the MECS. It is related to the distance from the base station to the MECS in the network, and because the macro base station is deployed at the same place as the MECS in our model, if the base station is MBS, $\varphi_n = 0$.

$$t_n^{o,t2} = d_n \varphi_n \tag{3}$$

Let's discuss the energy consumption of UE_n during the offloading process. The first thing to notice is that uploading the task T_n to the base station will require UE to transmit a wireless signal. As we use P_n^{send} to indicate the power of the UE when sending data, the energy consumption of this step is as:

$$e_n^{o,t} = P_n^{send} t_n^{o,t1} \tag{4}$$

In the process of data transmission from the base station to the MECS and the process of performing task calculation in the MECS, the UE is in a standby state that also consumes a small amount of energy. Denoting the power of the standby state as P_n^l, the energy consumption of the UE introduced due to this waiting time is:

$$e_n^{o,p} = P_n^l (t_n^{o,t2} + t_n^{o,p}) \tag{5}$$

However, since UE will consume standby energy anyway whether or not it is waiting for the computation result from the MECS, only the energy required for communication is considered the energy used for MEC, namely:

$$e_n^o = e_n^{o,t} \tag{6}$$

To simulate the geographical movement of mobile users, a transition probability matrix is set up, according to which users randomly shift between different SBS service areas. In our simulation model, if currently a user is not performing offloading tasks, it is allowed to leave areas with edge computing service. At the same time, the simulation model also allows the arrival of new users to MEC service areas.

2.2 Computation model

In the MEC model, the entire computing power of the MECS can be shared by multiple offloaded tasks that are being executed at the same time, while the local computing power can only support the execution of one computing task.

The theorem of large numbers indicates that when the task arrival probability of each UE is random, the overall demand for tasks of all UE will be very stable; by setting the "task generation groups", each group includes several devices, and the task arrival probability of these devices are same;

Just as the performance of all mobile devices in the real world is not exactly the same, the actual application scenarios are full of variability and uncertainty. From the initialization of the environment to the parameters of randomly arrived tasks, they are all given a lot of randomness: Many parameters of the simulation model are uniformly distributed and randomly generated within a certain range.

The CPU frequency, energy density, data transmission power of each user equipment, the data volume of each randomly arrived task and the CPU cycles required to complete the task are all uniformly distributed within the reasonable value ranges of the corresponding parameters. The theorem of large numbers indicates that when the task arrival probability of each UE is random, the overall demand of tasks for all UEs will be very stable, given the condition that there are plenty of UEs. To introduce some change through time, we set several "task generation groups", and each group includes several UEs. Every UE is assigned to a task generation group, and the UEs assigned to the same group have the same task arriving probability. In this way, the UEs change their offloading demand in groups. It not only let the model take into account the differences in users' demand for edge computing, but also highlight the variability of the overall demand. There are multiple ways to vary the task arriving probability of such a group. In our case, there are equal chances to increase or decrease this value, and the way we vary it is by multiplying or dividing it by a factor decided by whether such change is making the value out of the expected one or not. The detailed process is shown in Algorithm 1, where the task arriving probability is denoted as x.

Algorithm 1 Changes of task arriving probability

```
if real(p) then
    increase x as following
    if x> X_0 then
        x=x×OUT
    else
        x=x×IN
    end if
else
    if real(p/(1 − p)) then
        decrease x as following
        if x< X_0 then
            x=x/OUT
        else
            x=x/IN
        end if
    end if
end if
if x>UP then
    x=UP
end if
if x<DOWN then
    x=DOWN
end if
```

$real(p)$ represents that a random event with the chance of p happens. UP, $DOWN$ is the upper and lower bound of x, which in our experiment is set to be $10X_0$ and $0.1X_0$. X_0 is a hyper-parameter reflecting task coming rate, which controls how UE task coming probability is changed. And IN, OUT are set to be 2 and 1.5, respectively.

If choose to offload, in addition to (2), (3), the total delay will also include the part that MECS completing the offloaded task, which can be expressed as

$$t_n^{o,p} = \frac{c_n}{f_n} \tag{7}$$

Among the formula, f_n is the number of computing resources allocated to UE_n by the MECS, and its unit is the number of CPU cycles per second. The sum of the MEC computing resources allocated to all UEs shall not be greater than F, which is the maximum number of CPU cycles per second representing the computing power of MECS. The

process of transmitting the calculation result back to UE is omitted. And thus we can get the total offloading delay as

$$t_n^o = t_n^{o,t1} + t_n^{o,t2} + t_n^{o,p} \tag{8}$$

On the other side, if the user device UE_n chooses to perform its calculation task locally, the case would be different. Here we define t_n^l as its local execution delay, which is the execution time of the task with the corresponding local CPU. We let f_n^l be the number of local CPU cycles per second, then t_n^l could be expressed as

$$t_n^l = c_n / f_n^l \tag{9}$$

Then we use P_n^l to represent the consumed energy for local execution per second. And we let P_n^l be proportional to the square of CPU frequency, where the proportional coefficient, z_n^l, varies from device to device. Worth noting that $z_n^l f_n^l$ is the energy consumed to finish per CPU cycle amount of calculation. Then the energy consumption of locally executing T_n can be solved as [20]:

$$e_n^l = t_n^l P_n^l \tag{10}$$

where

$$P_n^l = z_n^l {f_n^l}^2 \tag{11}$$

This article assumes that the local CPU and up-link data communication of each UE can only serve one task at a certain time. This results in that when an earlier task is executed locally, new tasks waiting for offloading decision cannot be executed locally. If the decision algorithm allocates local computing power or up-link data communication resource, which has already been occupied, again, such a decision is invalid and won't be implemented.

In the section 2.1 above, we introduce how user devices move between SBS service zones. If a user moves to the place with none of BSs accessible, like "Zone_4" in the case illustrated in Fig. 2, its feature parameters are reset. And then the UE will execute all tasks in its task buffer locally in succession. The feature parameters of UE_n to be reset here include the transmission power P_n^{send}, the local computation speed f_n^l, local executing energy per second P_n^l, and the preset fixed parameters of the UE of the task generation group to which they belong; effect of resetting these parameters is similar to the generation of new user devices, but they differ in one critical aspect. After resetting feature parameters, the UE retains the queue of tasks to be executed in its buffer, which is why we say that it is "pseudo-generating" new users.

3 Solving the Multi-user Computation Offloading Game

As the mathematical relationship of variables has been given in both local and offloading calculation modes, it is now possible to express the offloading decision problem as a Markov decision process (MDP). Markov decision process is a mathematical framework for modeling discrete-time stochastic control process, where decisions are required sequentially given current observation [21]. MDP is based on two interactive objects, namely the agent and the environment, and its elements include state, action, strategy, and reward. Even some non-Markov processes can be modeled by MDP, and then the corresponding decision algorithm can be applied to get a fairly good solution.

State transitions in an MDP satisfy Markov property. Given the current state, future states do not depend on the past, that is, it satisfies:

$$\begin{aligned} P(s_{t+1}|s_t, s_{t-1}, s_{t-1},..., s_0, a_t, a_{t-1}, a_{t-1}, ..., a_0,) \\ = P(s_{t+1}|s_t, a_t) \end{aligned} \tag{12}$$

A partially observable Markov decision process is a generalized Markov decision process. The POMDP assumes that the system is determined by a latent MDP, but the agent cannot directly observe state variables that are sufficient enough to tell the state transaction probabilities. To represent the offloading decision problem as a POMDP, in this study, discrete-time slots are set up. And it is assumed that all events that occur in the same slot occur at the same moment. This approach is widely adopted.

We have adopted a variable period decision-making method, which will be explained in detail later. In previous related studies, although it is generally not clearly stated, there is another approach commonly adopted, which is to set fixed decision time slots for decision-making behaviors, and we make it as an integer multiple of the simulation time slot, named fixed period decision-making.

From the following two diagrams, it can be observed that the variable period approach of making decisions immediately after a task arrives has inherent advantages over the fixed period decision-making that faces possible waiting time after a task arrives.

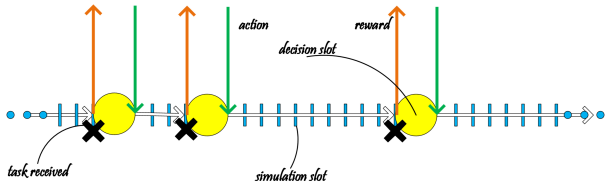

Fig. 3. Variable period decision

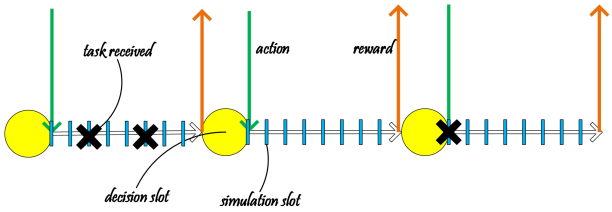

Fig. 4. Fixed period decision

In this paper, we only give you the MEC model and formulate the corresponding offloading decision problem as a standard POMDP, leaving the offloading strategy for future researchers. Yet we do implement a couple of vanilla methods, namely full offloading, full local, and the greedy one.

3.1 POMDP action

We define MECS's response to the offloading request of UE_n as

$$y_n = (R_n, B_n, g_n, f_n) \in \mathcal{Y} = \mathcal{N} \times \mathcal{G} \times (0, F_n] \tag{13}$$

This is the action that should be decided to take by the offloading decision policy in our case. The following explains the meaning of variables in each dimension of the offloading action vector y_n one by one. The variable on the first dimension, R_n, takes a value in $\{0, 1\}$ to indicate whether UE_n is allowed to perform offloading execution. When R_n = 0, it means that T_n is executed locally in UE_n. R_n = 1 means that the offloading decision is offloading T_n to the MECS for execution at the edge server. When R_n = 0, the task is completed locally, thus no MEC service is needed, and the value of B_n, g_n, f_n is meaningless. Here $\mathcal{N}$ is the number of all available communication base stations, indexed from one to $\mathcal{N}$. When B_n is selected from one to $\mathcal{N}$, it means that this calculation task will be offloaded to the MECS through the base station numbered B_n. The variable g_n in the third dimension represents that pass-band of the base station assigned to the task with a gain of g_n. The variable f_n in the fourth dimension represents the size of MECS computing resources allocated to this computing task, with the unit of CPU cycles per second. The POMDP action should take into account the occupation condition of the MECS resources, communication resources, and some other aspects.

3.2 POMDP state

The offloading decision algorithm makes decisions based on the observed state vector, and the state vector is defined as

$$x_n = (d_n, c_n, f_n^l, z_n^l, P_n^{send}, Z_n, C_n^s, C_n^l, O_{N,M}, F^{left}) \in \mathcal{X}^l \tag{14}$$

Among it, d_n, c_n, f_n^l, z_n^l, P_n^{send}, Z_n these quantities have been discussed in the local computing model. It is noteworthy that f_n^l, z_n^l, P_n^{send} will not change over time for the same UE, but for different UEs these values may be different.

$$C_n^s = \mathbf{1}(transmitting\ power\ occupied) \tag{15}$$

$$C_n^l = \mathbf{1}(local\ computing\ power\ occupied) \tag{16}$$

Here $\mathbf{1}(.)$ is the truth value function. Its value is one when the event in its brackets is true, otherwise zero. Use F^{left} to represent the vacant computing resources of the MECS currently. The matrix $O_{N,M}$ represents the situation of the channels' occupancy, where its amount of rows N is the number of base stations, indexed from 0 to N-1. And the M columns represent the M channels that each base station has. Any element of $O_{N,M}$, the channel occupancy matrix, has two possible values, 0 and 1. And for the element (i, j) of the $O_{N,M}$, its value should be $\mathbf{1}(\texttt{the j}^{\texttt{th}}\ \texttt{channel of the i}^{\texttt{th}}\ \texttt{BS is busy})$. $O_{N,M}$ is stretched into one dimension in the state vector x_n. Z_n represents the zone number where the UE is located, corresponding to the coverage area of SBS discussed in the last chapter.

3.3 POMDP reward

Our goal in this MEC model is trying to satisfy the QoS demand of all users in general. And thus the total reward function of transition from x_n to x_{n+1} is defined as the sum of the QoS indexes of all tasks in the system, multiplied by a modifying factor, which is exponentially dependent on the time length between x_n is observed and when $Gain_n^i$ is introduced, denoted as $time(from\ x_n\ to\ T_n^i)$ in (17) below.

$$Gain_n^{all} = \sum_i Gain_n^i \gamma^{time(from\ x_n\ to\ T_n^i)} \tag{17}$$

This $Gain_n^{all}$ will be the immediate reward received at the moment x_{n+1} is observed. And for $Gain_n^i$ in (17), the gain introduced by T_n^i when it is completed or failed, is defined as

$$\begin{aligned} Gain_n^i = \mathbf{1}(T_i\ finished\ successfully)(I^t t_i + \\ I^e e_i + I^{finish}) + I^{fail}\mathbf{1}(T_i\ failed)) \end{aligned} \tag{18}$$

$\mathbf{1}(.)$ has the same meaning as when it appears above, which is a truth function. I^t, I^e, I^{finish}, I^{fail} are weight coefficients chosen manually, and they also play the rule of unifying units for summing up gains introduced by different factors. The reward after each action is the sum of all Gains during the period between this action is taken and the next one. For those offloading decisions that make an invalid allocation, the corresponding task will still maintain in the buffer. If a task in the buffer exceeds the maximum allowable delay, it will be cleared and judged as a failure.

Noting we raise a variable period decision POMDP, which means that the interval between states transitions here may be different. Considering this point, to use the famous Bellman equation, the discount factor part should be relevant to the length between two transitions of states, just like $\gamma^{time(from\ x_n\ to\ T_n^i)}$ in (17). The Bellman equation is rewritten as follows:

$$Q(x, y) = E_{p,x'}[p(x, y) + \gamma^{time(from\ x\ to\ x')} Q(x', \mu(x'))] \tag{19}$$

Here the Q value $Q(x, y)$ represents the quality of action y given the state x, the higher the better. $E_{p,x'}$ denotes expectation, where the payoff $p(x, y)$ and the next state x' is uncertain. And in our paper, the payoff should be the expectation of the corresponding $Gain_n^{all}$ defined in (17).

4 Numerical Results

4.1 General setup

In the experiment part, we test three intuitive offloading approaches, aiming to illustrate some basic features of our simulation model. The three naïve offloading strategies are full local, full offloading, and greedy policy respectively. If given a POMDP state x_n in 14, we stipulate the corresponding action decided by each of these strategies as follows. Full local strategy always yield the POMDP action $(0, 0, 0, 0)$. Full local strategy will give action $(1, BS, channel, \alpha F^{left})$,

where α is a preset parameter defaulted as $2/3$. We have tested $\alpha = 1/2$ and And BS is Z_n if not all channels of its micro BS are occupied, otherwise BS shall be 0. $channel$ shall be the non-occupied one with the highest gain of the chosen BS. The greedy approach we implement uses a very basic logic. It will make a local computing decision if the communication of UE is busy, and decides to offload the computation if there is already a task running locally. In the case that both offloading and local approach is possible, our greedy strategy shall compare the expected weighted sum of energy consumption and time latency in both cases, and choose one accordingly.

The transition probability matrix of UE movement discussed around Fig. 2 is as below. We choose this to make the probability of leaving zone 4, the zone without MEC services, a little bit greater than moving into zone 4.

$$T = \begin{bmatrix} * & 0.001 & 0.001 & 0.001 \\ 0.001 & * & 0.001 & 0.001 \\ 0.001 & 0.001 & * & 0.001 \\ 0.004 & 0.004 & 0.004 & * \end{bmatrix}$$

where the star sign is set to be the value that could make sums of each line to be 1, which is the probability of keeping its zone unchanged for each UE in each movement period. Other important parameters used in the simulation experiment are set as in Table 1.

4.2 Experimental results

Fig. 5 to Fig. 9 illustrate the effect of different task coming rates on the MEC offloading QoS.

And Fig. 5 shows that the greedy policy chooses to locally execute more tasks if the task density gets higher. With the task coming rate increases, a large percent of computationally powerful MECS resources will have been already allocated, so as the superior BSs and channels. This makes the new coming tasks can only adopt to be executed locally. Note that the offloading rate of the full offloading strategy is not 100%. This is because in the cases that a UE is out of MEC service, no offloading decision shall be made and such UE shall adopt local computing automatically.

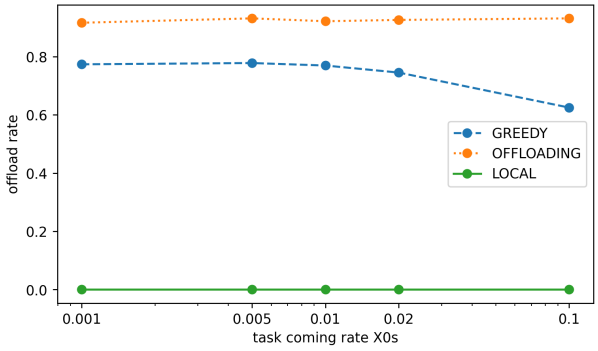

Fig. 5. Offloading rate versus task coming rate

Fig. 6 shows that the average time latency increases as the task coming rate increases regardless of which offloading strategy is adopted. Another thing to be noticed is that the latency is not proportional to the task coming rate. It maintains stable if the task coming rate is low.

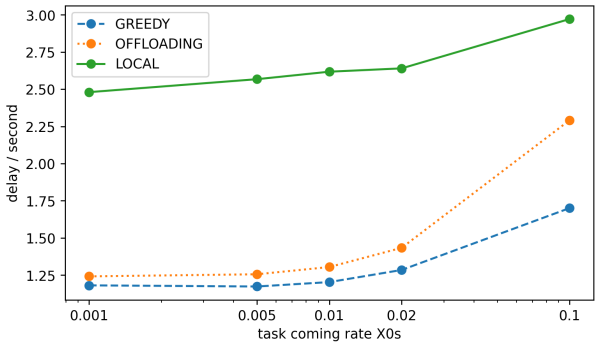

Fig. 6. Average delay versus task coming rate

As in Fig. 7, we see the average energy cost per task in both full offloading and full local are basically invariant to task coming rate. And offloading is more energy-efficient than local computation. The greedy strategy can achieve a similar energy consumption result with the full offloading approach if the task density is relatively low, and it increases if the task density gets higher, in coherence with the increase of offloading rate.

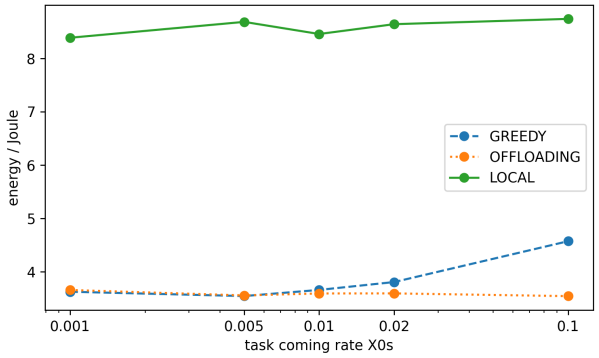

Fig. 7. Average energy versus task coming rate

Fig. 8 demonstrate that with a higher workload, both full offloading and full local have almost identical bad fail rate results. They are affected by task coming rate in similar degrees.

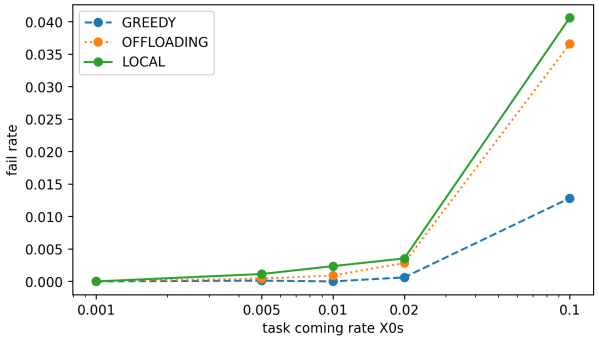

Fig. 8. Task fail rate versus task coming rate

The y-axis reward in Fig. 9 can be understood as a weighted sum of delay, energy and task fail rate. Here we set I^t = -0.5 sec^{-1}, I^e = -0.02 J^{-1}, I^{finish} = 10 and I^{fail}

TABLE 1
Simulation parameters

	name	type	value	measure
time	slot width	scalar	0.1	second
	total simulation time	scalar	48	hours
Base Station	BS2MECS transition rate	list	[0, 0.05, 0.1, 0.15]	second/MB
	channel gain	list	[-5, -10, -15]	dB
	bandwidth	scalar	5	MHz
	noise	scalar	4×10^{-8}	W
MECS	CPU computing ability	scalar	15	GHz
UE	CPU computing ability	range	0.5∼2	GHz
	energy coefficient	range	1∼4	W/GHz^{-2}
	transmission power	range	5∼15	W
	number (by default)	scalar	20	–
task	X_0 (by default)	scalar	0.01	sec^{-1}
	task generation groups	scalar	3	–
	data size	range	0.2∼10	MB
	computing cost	range	0.5∼5	G(CPU cycle)
	maximum accepted delay	scalar	10	sec

= -4. From that, we shall draw the following conclusion. Though all the three working modes tend to get bad results as the task coming rate increases, the greedy strategy curve maintains steady the most. Due to its feature that lets both local and MEC computing resources be utilized, the greedy approach can function well in conditions with higher workloads. And the offloading curve drops faster than the local curve. Using computation resources from both MECS and UE shall make our service more robust in extreme conditions.

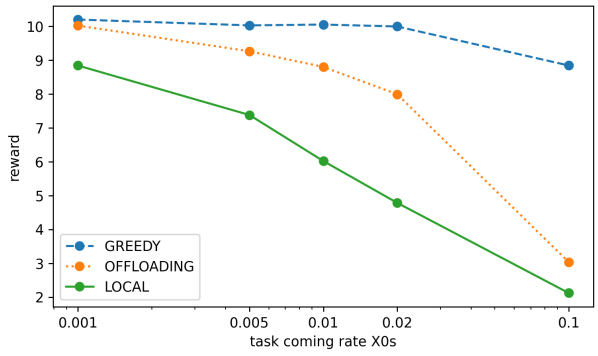

Fig. 9. Average reward versus task coming rate

The drop of the GREEDY curve in Fig. 10 indicates with UEs number increases, local computing shall be more dominant. Fig. 11 shows that as the number of UEs increases, the all offloading strategy tends to get longer latency. And we see that the all local approach is not affected by UE number that much, in Fig.11 to Fig. 13.

It is worth noticing that in Fig. 13, the task fail rate of our full local approach does increase with the UE number increasing. This may be because of the coarseness of our simulation slot limiting the number of decisions that could

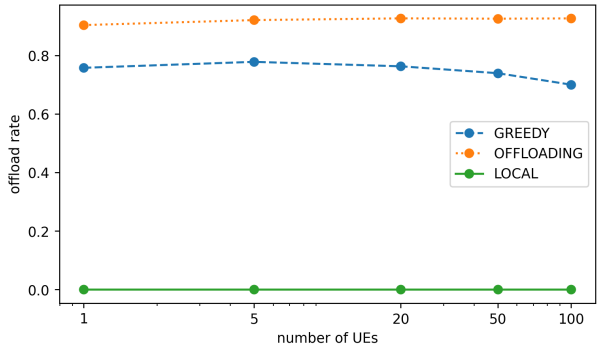

Fig. 10. Offloading rate versus UE number

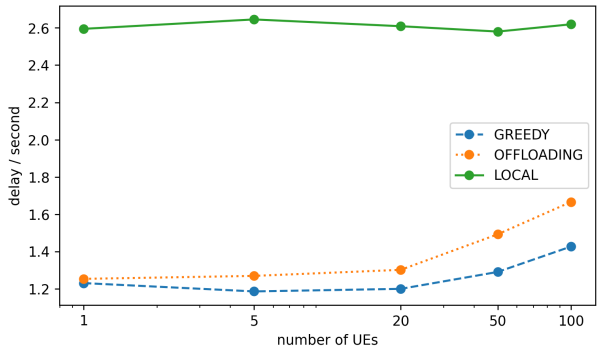

Fig. 11. Average delay versus UE number

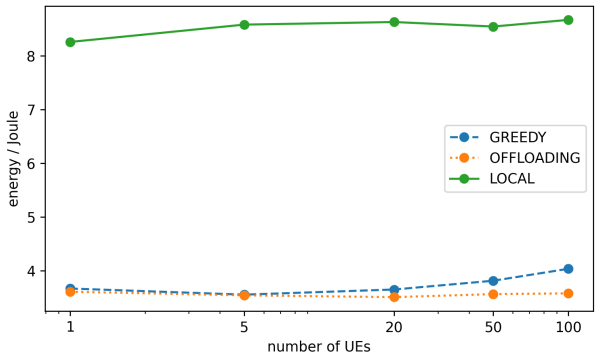

Fig. 12. Average energy versus UE number

be made during a certain amount of time. In Fig. 14, full offloading tend to get a lower reward as the number of UE increases. This is consistent with the theoretical analysis that the computing resource of MECS won't increase with UEs number increases.

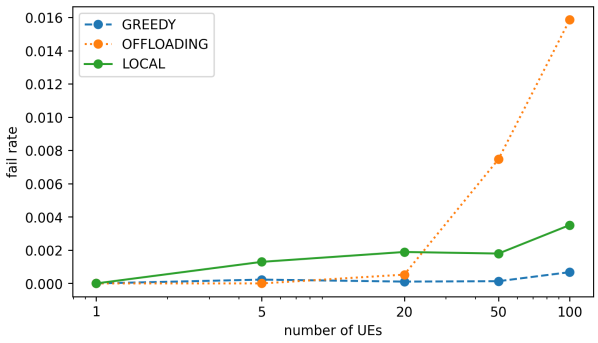

Fig. 13. Task fail rate versus UE number

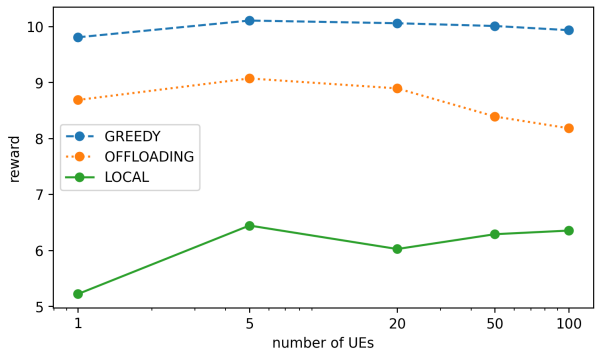

Fig. 14. Average reward versus UE number

α is the proportion of MECS spare computing resources to be used for a new task, in the case that it is executed in MECS. We have tested several α values in the default condition, showing the results in Table 2, and choose α to be $3/4$ in our other experiments.

5 Conclusion

In this paper, we bring the edge computing society a new MEC simulation model, with its offloading decision process formed as a POMDP. We argue our model considers the spacial movement of UEs, and can preserve long-term effects of one offloading decision. Moreover, our model has enough randomness in many aspects. We try to reflect the fluctuation of offloading demands in reality while still keeping randomness. Commonly it would be a conflict, considering "the law of large numbers". By designing "task generation groups", we successfully save ourselves from such a dilemma. The variable period decision feature of our POMDP offloading problem is more suitable for handling the scenario where new offloading demands come randomly. We also implement three naïve offloading strategies and compare their effects. We hope our model can be broadly applied in future works.

Acknowledgments

This work is supported by the National Key Research and Development Project of China [Grant numbers 2018YFB2003501] and the National Nature Science Foundation of China [Grant numbers 61320106010, 61573019, 61627810]. Other than that, we would like to thank Junyao Yang from SASEE of BUAA for helpful discussions about this research and thank Zehui Yang from IOA of CAS for helping us edit our draft paper.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *arXiv preprint arXiv:1701.01090*, 2017.

[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, 2017.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[4] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, p. 102781, 2020.

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

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *2012 IEEE 1st International Conference on Cloud Networking (CLOUDNET)*, pp. 80–86, IEEE, 2012.

[8] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 1451–1455, IEEE, 2016.

[9] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE access*, vol. 4, pp. 5896–5907, 2016.

[10] M.-H. Chen, B. Liang, and M. Dong, "A semidefinite relaxation approach to mobile cloud offloading with computing access point," in *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 186–190, IEEE, 2015.

[11] S. Jošilo and G. Dán, "Selfish decentralized computation offloading for mobile cloud computing in dense wireless networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 1, pp. 207–220, 2018.

[12] Q.-V. Pham, T. Leanh, N. H. Tran, B. J. Park, and C. S. Hong, "Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach," *IEEE Access*, vol. 6, pp. 75868–75885, 2018.

TABLE 2
Influence of different $alpha$

	α	reward	delay	energy	fail rate	offload rate
OFFLOAD	$1/4$	7.280	1.7871	3.557	0.0029	0.9249
	$1/2$	7.969	1.454	3.574	0.0011	0.9254
	$3/4$	7.859	1.468	3.523	0.0060	0.9255
	0.99999	6.802	1.154	3.570	0.1796	0.9129
GREEDY	$1/4$	9.694	1.6175	3.9761	0.0002	0.6920
	$1/2$	9.944	1.313	**3.743**	**0.0001**	0.7491
	$3/4$	9.921	**1.275**	3.809	0.0003	0.7404
	0.99999	**9.957**	1.347	4.470	0.0060	0.6442

[13] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, no. 1, pp. 1–21, 2020.

[14] B. Cho and Y. Xiao, "Learning-based decentralized offloading decision making in an adversarial environment," *arXiv preprint arXiv:2104.12827*, 2021.

[15] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[16] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Transactions on Mobile Computing*, vol. 14, no. 12, pp. 2516–2529, 2015.

[17] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, 2015.

[18] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

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

[20] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

Zijian Li joined Computational Intelligence Lab, NTU, Singapore as an intern in 2019, and received his Bachelor degree in the field of Automation Science from Beihang University in 2020. His research interests lie in the area of developing better Machine Learning Techniques, and Multimodality understanding.

Yan Wang received the Ph.D. degree in control theory and control engineering from Northeastern University, Shenyang, China, in 2003. From 2003 to 2005, she was a Postdoctoral Fellow with the National Key Laboratory of Intelligent Technology and Systems, Tsinghua University. She is currently an Associate Professor with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, China. Her research interests include networked control systems, intelligent systems, and computation.