Dynamic Service Migration with Partially Observable Information in Mobile Edge Computing

Xiaoqian Li[†], Yakun Zhou[*†], Yao Sun[‡], Siyu Chen[†], Jienan Chen[†], Gang Feng[†]
[†]*National Key Laboratory of Science and Technology on Communications*
University of Electronic Science and Technology of China, China
[‡]*James Watt School of Engineering, University of Glasgow, U.K*
Email: yakun-zhou@outlook.com

***Abstract*—Service migration, determining when, where and how to migrate the ongoing service, is of paramount importance in mobile edge computing (MEC) for provisioning high quality of service to mobile users. With respect to high network dynamics and stringent delay requirements, service migration is a rather challenging issue in MEC. In this paper, we formulate service migration as a partially observable Markov decision process (POMDP) based on the fact that an edge server can only obtain partial users' information, or the information of its own serving users. A learning-based intelligent service migration algorithm, named iSMA, is proposed to minimize the long-term service delay of all users. iSMA consists of two function modules, a latent space model and a cross-entropy planning algorithm, where the latent space model is used to infer the full state of the environment based on the partial information observed, and the cross-entropy planning algorithm is used to search the best service migration strategy. Numerical results show that our proposed iSMA reduces the service delay by about 58% when compared with a well-known deep learning-based solution.**

***Index Terms*—Service migration, mobile edge computing, partially observable Markov decision process.**

I. Introduction

Mobile edge computing (MEC), enabling cloud computing capabilities at network edge, is shown to be an effective way to enhance network performance as well as user experience [1]. In MEC, edge servers are distributed across the network and closely connected to edge nodes such as cellular base stations and wireless access points. As a result, mobile users are able to access the cloud services directly via radio access network (RAN), which significantly reduces service delay and network trunk traffic.

To maintain the above benefits of MEC, one crucial issue that needs to be carefully addressed is dynamic service migration [2]. Because of the limited coverage of a single edge server, some ongoing service may need to be migrated to adapt to user mobility. The problem is when, where and how to migrate the service such that the service delay can be minimized. It is challenging to find an optimal migration policy due to the high uncertainty in user mobility and service request patterns [3].

The service migration problem in MEC has attracted intensive research attentions in recent years [3]–[8]. In [3]–[5], by formulating the problem as a sequential decision-making process, the service migration strategy is designed based on markov decision process (MDP). Deep learning methods are used in [6] [7] for making efficient service migration decisions in the long-term process. In [8], a central controller is used to guide how and where to migrate the ongoing service under the software defined networking (SDN) framework.

However, the statistical model for capturing users' and servers' information in most of the relevant works is unrealistic. For example, only a single user or server is considered in [6] [7]. Although multiple users and servers are considered in [3]–[5], the authors assume that the information of all the users (in terms of locations and/or service requests) served by different edge servers is available. Note that obtaining such global information in a dynamic network is impractical/infeasible, as significantly large signaling overhead is incurred. Although the global information can be obtained if the SDN controller is deployed [8], maintaining such a controller introduces additional overhead and security threats [9]. In practice, an edge server may obtain only partial users information, or the information of its current serving users.

In this paper, we consider a practical scenario with multiple users and edge servers, and each edge server can obtain the information only of its own serving users. We study a general service migration problem (where computation offloading involved in the service are also considered), and formulate the problem as a partially observable markov decision process (POMDP). To minimize the long-term average service delay of all users, an intelligent algorithm (named iSMA) is proposed to find the best service migration and computation offloading strategies. Simulation results show that iSMA outperforms the existing DQN algorithm [6] by about 58%. Our main contributions are summarized as follows.

- We formulate a joint optimization problem of service migration and computation offloading. An intelligent algorithm iSMA is proposed, for simultaneously determining whether/how to migrate the service and whether/how to conduct the computation offloading, under the availability of partial users' information.
- A latent space model is designed to infer the hidden state such that the full state of the environment can be obtained. In particular, by taking both stochastic and deterministic transitions into consideration, we establish the latent space

This work is supported by Artificial Intelligence Key Laboratory of Sichuan Province NO. 2020RZJ02. *(Corresponding author: Yakun Zhou.)*

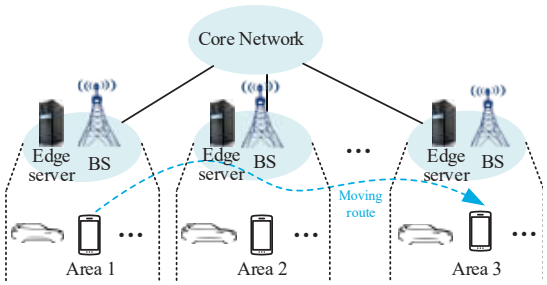

Fig. 1. Illustration of an MEC network

based on a recurrent transition model.

- Based on the latent space model, a cross-entropy planning algorithm is then designed to search for the best service migration and computation offloading strategies, aiming at minimizing the average service delay of all users in the long-term process.

II. System Model And Problem Formulation

We consider a mobile edge system consisting of N users and L edge servers, as shown in Fig. 1, where each edge server is co-located with a base station such that it can be accessed by the users via RAN [3] [4]. Since the users' statistical information obtained by an edge server is time-varying and depends on user mobility, we suppose that each edge server updates its decisions on regular time basis. To this end, we assume time is discretized into slots, with the same time interval in each slot, and there is no need for the users and edge servers to precisely synchronize on individual time slot.

A. System Model

1) User Model

Let u_i denote the i-th $(i = 1, 2, \ldots, N)$ user among the N users, and b_j denote the base station serving u_i. Like [3], we consider the case where users (e.g. autonomous vehicles) continuously access the service hosted on edge servers and generate the data that needs to be processed continuously. We assume the data-oriented computation task generated by a user can be partitioned into a finite number of component tasks [10].

Without loss of generality, c_i is used to denote the computational workload (or data size) in the component task generated by user u_i. Task requests will be generated sequentially by u_i to request the edge server's computing resources for processing each component task. In particular, a task request will be generated immediately when the data (or computational workload) collected by user u_i reaches c_i.

Let $T_i(\tau)$ denote the task request from u_i at time slot τ. Since the task request is generated based on the data size c_i, $T_i(\tau)$ in a time slot τ can be NULL if the data collected is less than c_i. Otherwise, $T_i(\tau) = \{c_i, f_i^L, f_i^r, g_i, \delta_i^{max}\}$, where f_i^L is the computing capability of u_i (i.e., CPU clock frequency), f_i^r is the number of CPU cycles needed to complete this component task, g_i is the real-time channel gain between u_i and its base station, and δ_i^{max} is the maximum delay allowed to complete this component task. Each task is measured based on the number of CPU computing cycles required to complete it (or f_i^r) [11]. Note that when c_i is fixed for a specific service, the elements in $T_i(\tau)$ are also fixed expect g_i.

2) Edge Server Model

Let e_l denote the l-th edge server $(l = 1, 2, \ldots, L)$. Edge server e_l is assumed to periodically broadcast the information of its remaining computing resources to other servers such that they can determine whether to migrate the service (and component tasks) to e_l. When server e_l first receives a task request $T_i(\tau)$ from user u_i, it will immediately execute a decision-making algorithm (e.g. iSMA proposed in Section III) to determine the action for this request. (An action includes a service migration and computation offloading strategy.) Meanwhile, server e_l will update its decisions every D_l time slots, or e_l will execute the decision-making algorithm every D_l slots to update the actions for all the task requests received.

The decision-making algorithm will output an action for each task request input. Let a_i be the action for task request from u_i. In particular, action $a_i = \{a_i(IP), a_i(\eta), a_i(f^e)\}$, where $a_i(IP)$ is the address of the edge server serving u_i, $a_i(\eta)$ is the computation offloading ratio and $a_i(f^e)$ is the computing resource ratio allocated for this task. If $a_i(IP)$ is the edge server itself, the service is not migrated; otherwise, the service will be migrated to another edge server for better performance. The value of $a_i(\eta)$ determines the computation offloading strategy for the task, i.e., no computation offloading $(a_i(\eta) = 0)$, full offloading $(a_i(\eta) = 1)$ and partial offloading. Since it is impractical to partition each component task arbitrarily [10], we choose the value of $a_i(\eta)$ from set $\{0, 0.1, 0.2, ..., 0.9, 1\}$.

3) Service Migration Mechanism

In our model, each user u_i stores the address of the edge server currently associated with. Each edge server e_l stores the set of its current serving users U_τ^l, the set of task requests received T_τ^l and the set of remaining computing resources at all edge servers F_τ. In other words, $U_\tau^l = \{u_i | u_i \ served \ by \ e_l \ at \ t = \tau\}$, $T_\tau^l = \{T_i(\tau) | u_i \in U_\tau^l\}$.

Suppose user u_i sends its first task request $T_i(\tau)$ to edge server e_l at time slot τ. Server e_l then recognizes that u_i is a newcomer, the user set U_τ^l and request set T_τ^l are then updated, and the decision-making algorithm will be immediately executed at server e_l with U_τ^l, T_τ^l and F_τ as input.

When the algorithm terminates, action $a_i = \{a_i(IP), a_i(\eta), a_i(f^e)\}$ is obtained and sent to user u_i. No service migration if $a_i(IP)$ is still server e_l itself; otherwise, the service needs to be migrated to another server for better performance and the corresponding edge server for serving u_i is then updated. Then u_i will immediately upload $a_i(\eta)$ portion of the collected data to its associated edge server and process the remaining $(1 - a_1(\eta))$ portion of data locally. Once the $a_i(\eta)$ portion of data is received, the target edge server will use $a_i(f^e)$ $(a_i(f^e) \leq 1)$ portion of its computing resources to process this data. The component task is completed when the local data processing (for the

$(1 - a_i(\eta))$ portion of data) is finished and the processing result (of the $a_i(\eta)$ portion of data) obtained by the edge server is returned to u_i.

In addition to immediately executing the decision-making algorithm when receiving the first request from a newcomer, edge server e_l also updates its decisions every D_l time slots for all the requests received. The new service migration strategy (determined by $a_i(IP)$) and computation offloading strategy (determined by $a_i(\eta)$) obtained will then be sent to the corresponding user. We call D_l as a decision period of edge server e_l. During the decision period of e_l, the same service migration and computation offloading strategies will be applied to the component tasks generated by e_l's current serving user. Obviously, the value of D_l determines how often the decision is updated, and its impact on system performance can be evaluated through simulation experiments.

B. Objective Function

Based on the mechanism described above, our objective is to design a decision-making algorithm for minimizing the long-term average component task completion time of the N users. Each component task completion time is mainly determined by 1) local data processing time ($(1 - a_i(\eta))$ portion of data), 2) data uploading time ($a_i(\eta)$ portion of data) and 3) data processing time at the edge server ($a_i(\eta)$ portion of data). The time required for the user to send the task request or the server to send back the action a_i is ignored since the size of control messages can be negligible as compared to the data size of each component task.

The local data processing time at user u_i is

$$t_i^L = \frac{f_i^r(1 - a_i(\eta))}{f_i^L}, \tag{1}$$

where f_i^L is the CPU clock frequency of user u_i, and f_i^r is the number of CPU cycles needed to complete the component task.

The data uploading time can be divided into two parts: the wireless transmission time from user u_i to its base station b_j and the transmission time from b_j to edge server e_l. With signal power p_i and channel gain g_i between u_i and b_j, the channel capacity is

$$R_i^w = Blog(1 + \frac{p_i g_i}{N_0}), \tag{2}$$

where B is the channel bandwidth, and N_0 is the noise power. With formula (2), the wireless transmission time is

$$t_i^{cw} = \frac{c_i a_i(\eta)}{R_i^w}, \tag{3}$$

where $c_i a_i(\eta)$ is the uploaded data size.

Suppose the average data transmission rate from base station b_j to edge server e_l is R_i^f. Then the transmission time from b_j to e_l is mainly determined by

$$t_i^{cf} = \frac{c_i a_i(\eta)}{R_i^f} d_{i,l}, \tag{4}$$

where $d_{i,l}$ is the number of hops needed from b_j to e_l via the Internet. The queuing delay is ignored by assuming the network is not heavily loaded.

Finally, the data processing time at server e_l is

$$t_i^e = \frac{a_i(\eta) f_i^r}{f_l^e a_i(f^e)}, \tag{5}$$

where f_l^e is the CPU clock frequency of server e_l and $a_i(f^e)$ is the computing resource ratio allocated for the task.

With equations (1)-(5), each component task completion time is then expressed as

$$t_i = max(t_i^L, t_i^{cw} + t_i^{cf} + t_i^e). \tag{6}$$

Under the constraint that t_i is no longer than a given threshold δ_i^{max}, we aim at designing the decision-making algorithm to minimize the long-term average t_i over the N users.

C. Partially Observable Markov Decision Process

Let $A_\tau = \{A_\tau^l, l = 1, 2, ..., L\}$ denote the actions for the N users at time slot τ, where $A_\tau^l = \{a_i | u_i \in U_\tau^l\}$ is the action set obtained by server e_l at time slot τ. Let $o_\tau^l = \{U_\tau^l, T_\tau^l, F_\tau\}$ be the observed state of e_l at time slot τ, which is a partial state of the environment. Then we formulate the problem as a POMDP:

- Transition model: $s_\tau^l \sim p(s_\tau^l | s_{\tau-1}^l, A_{\tau-1}^l)$
- Observation model: $o_\tau^l \sim p(o_\tau^l | s_\tau^l)$
- Reward model: $r_\tau^l \sim p(r_r^l | s_\tau^l, A_\tau^l)$
- Action: $A_\tau^l \sim \pi^l(A_\tau^l | o_{\leq\tau}^l, A_{<\tau}^l)$
- Encoder: $s_\tau^l \sim p(s_\tau^l | o_{\leq\tau}^l, A_{<\tau}^l)$

where s_τ^l denotes the hidden state (or unobserved state) of the environment inferred by server e_l at time slot τ, and r_τ^l is a reward obtained based on action set A_τ^l and hidden state s_τ^l. Based on the partial observed states $o_{\leq\tau}^l$ and previous actions $A_{<\tau}^l$, server e_l then decides whether and how to migrate the service to another server such that the long-term average component task completion time (or service delay) is minimized. In particular, the decision-making algorithm should be designed to find a sequence of actions $A_0^l, A_1^l, \ldots, A_\infty^l$ at each server from $t = 0$ to $t = \infty$ such that the actions $A_0, A_1, \ldots, A_\infty$ for all N users are optimal, as detailed below:

$$\begin{aligned}
\mathbf{P}: A_0, \ldots, A_\tau, \ldots, A_\infty &= \underset{A_0,\ldots,A_\tau,\ldots,A_\infty}{\arg\min} \{\sum_{\tau=0}^{\infty}(\frac{1}{N}\sum_{i=1}^{N} t_i)\} \\
&= \underset{A_0,\ldots,A_\tau,\ldots,A_\infty}{\arg\min} \{\sum_{\tau=0}^{\infty}(\frac{1}{N}\sum_{i=1}^{N} max(t_i^L, t_i^{cw} + t_i^{cf} + t_i^e))\} \\
&= \underset{A_0,\ldots,A_\tau,\ldots,A_\infty}{\arg\min} \{\sum_{\tau=0}^{\infty}(\frac{1}{N}\sum_{i=1}^{N} max(\frac{(1 - a_i(\eta)) f_i^r}{f_i^L}, \\
&\frac{c_i a_i(\eta)}{R_i^w} + \frac{c_i a_i(\eta)}{R_i^f} d_{i,l} + \frac{f_i^r a_i(\eta)}{f_l^e a_i(f^e)}\}
\end{aligned} \tag{7}$$

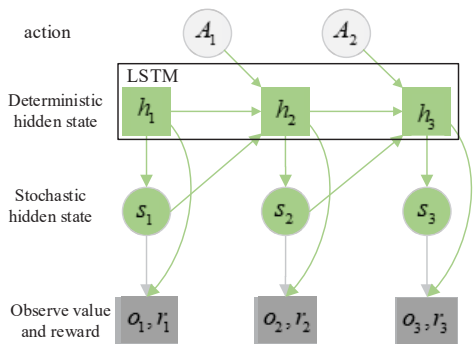

Fig. 2. Recurrent state transition model

III. Intelligent Service Migration Algorithm

An intelligent decision-making algorithm iSMA is designed in this section for jointly solving the problem of service migration and computation offloading. It consists of two function modules: a latent space model and a cross-entropy planning algorithm, where the latent space model is used to infer the hidden state of the environment, and the cross-entropy planning algorithm is to find the best action with the maximum reward. Focusing on edge server e_l only, iSMA is detailed below. For easy presentation, the observed state o_τ^l, action set A_τ^l and hidden state s_τ^l obtained by edge server e_l are simplified to o_τ, A_τ and s_τ below.

A. Latent Space Model

The latent space model consists of four components, transition model, observation model, reward model and encoder model, where the observation model is used for model training. Due to the dynamic environment in MEC, we propose to establish the latent space based on a recurrent transition model, as shown in Fig. 2. In our model, the hidden state consists of two parts, a stochastic state s_τ and a deterministic state h_τ, where s_τ is modelled as a Gaussian variable, and h_τ is obtained based on the LSTM network [12]. Our recurrent latent space model (RLSM) is then established as follows.

- Deterministic state transition model: $h_\tau = f(h_{\tau-1}, s_{\tau-1}, A_{\tau-1})$
- Stochastic state transition model: $s_\tau \sim p(s_\tau|h_\tau)$
- Observation model: $o_\tau \sim p(o_\tau|h_\tau, s_\tau)$
- Reward model: $r_\tau \sim p(r_r|h_\tau, s_\tau)$
- Encoder: $s_\tau \sim q(s_\tau|h_\tau, o_\tau)$

In RLSM, the stochastic state transition model is a fully-connected neural network with mean and variance as outputs. The observation model is a fully-connected neural network with mean and identity covariance as outputs. The reward model is a fully-connected neural network with mean and unit variance as outputs. And the encoder model is a fully-connected neural network with mean and variance as outputs.

The objective of RLSM is to jointly maximize the log-likelihood of the observed and reward sequences:.

$$\theta = \arg\max_\theta (\ln p(o_{1:K}|A_{1:K}) + \ln p(r_{1:K}|A_{1:K})), \quad (8)$$

where $\ln p(o_{1:K}|A_{1:K})$ is the log-likelihood of the observed sequences, and $\ln p(r_{1:K}|A_{1:K})$ is the log-likelihood of the reward sequences.

B. Algorithm iSMA

Based on the latent space model, the cross-entropy planning algorithm is used to generate a set of action sequences $\{A_{\tau:\tau+K}\}$ and output the best one with the maximum reward, where $A_{\tau:\tau+K}$ denotes the actions from $t = \tau$ to $t = \tau + K$. To be more specific, with the observed state o_τ as input, iSMA consists of the following 9 steps:

1) Initialize the distribution of the action sequences with normal distribution, i.e., $A_{\tau:\tau+K} \sim Normal(0, \mathbb{I})$.

2) With $A_{\tau:\tau+K} \sim Normal(0, \mathbb{I})$, a possible action set A_τ is obtained.

3) With the observed state o_τ and action set A_τ, a hidden state and reward r_τ are inferred based on the RLSM model.

4) Repeat steps $2) - 3)$ K times by varying time t from $t = \tau + 1$ to $t = \tau + K$. One action sequence $A_{\tau:\tau+K}$ with the corresponding reward sequence $r_{\tau:\tau+K}$ is then obtained. We evaluate the action sequence $A_{\tau:\tau+K}$ based on (9):

$$R^i = \sum_{k=\tau}^{\tau+K} \gamma^{k-\tau} r_k \quad (9)$$

5) Repeat step 4) I times to get a set of action sequences $\{A^i\}_{i=1}^I$, where each A^i is an action sequence $A_{\tau:\tau+K}$. Similarly, a set of rewards $\{R^i\}_{i=1}^I$ is obtained, where each R_i is calculated based on (9).

6) Sort set $\{A^i\}_{i=1}^I$ from the largest reward R_i to the smallest. The first $Q(Q < I)$ action sequences are then added to a new set $\mathbb{A}$, i.e.,

$$\mathbb{A} \longleftarrow \arg sort(\{A^i|R^i\}_{i=1}^I)_{1:Q} \quad (10)$$

7) The mean and variance of set $\mathbb{A}$ are calculated based on (11) and (12). The distribution of the action sequences is then updated with the new mean and variance obtained, i.e., $A_{\tau:\tau+K} \sim Normal(\mu_{\tau:\tau+K}, \sigma_{\tau:\tau+K}\mathbb{I})$.

$$\mu_{\tau:\tau+K} = \frac{1}{Q} \sum_{A^i \in \mathbb{A}} A^i_{\tau:\tau+K} \quad (11)$$

$$\sigma_{\tau:\tau+K} = \sqrt{\frac{1}{Q-1} \sum_{A^i \in \mathbb{A}} (A^i_{\tau:\tau+K} - \mu_{\tau:\tau+K})^2} \quad (12)$$

8) Repeat steps $2) - 7)$ J times with the new distribution $A_{\tau:\tau+K} \sim Normal(\mu_{\tau:\tau+K}, \sigma_{\tau:\tau+K}\mathbb{I})$.

9) The final mean calculated based on (11) is selected as the best action sequence, i.e., $A^* = A_{\tau:\tau+K} = \mu_{\tau:\tau+K}$. The action set A_τ in A^* is then selected as output, and algorithm iSMA stops.

From the steps above, we can see that the time complexity of iSMA is $O(I \cdot J \cdot K \cdot T^{RLSM})$, where T^{RLSM} is the time complexity of our proposed RLSM. As the five models used in RLSM are based on LSTM or fully-connected neural networks, T^{RLSM} is then determined by the dimensions of inputs, the number of layers and the number of cells used in the neural network.

TABLE I
PARAMETERS IN RLSM

Network	Layer	Hidden units	Activation
Encoder model	Input	$H^e_{in} = H^L_h + 246$	/
	μ_{output}	$H^e_{out1} = 60$	tanh
	σ_{output}	$H^e_{out2} = 60$	sigmoid
Stochastic transition model	Input	$H^s_{in} = 200$	/
	Hidden layer1	$H^s_1 = 512$	tanh
	μ_{output}	$H^s_{out1} = 60$	tanh
	σ_{output}	$H^s_{out2} = 60$	sigmoid
Reward model	Input	$H^R_{in} = 60 + 200$	/
	Hidden layer1	$H^s_1 = 512$	relu
	Hidden layer2	$H^s_{out1} = 60$	relu
	output	$H^s_{out2} = 60$	relu
Observation model	Input	$H^o_{in} = 60 + 200$	/
	Hidden layer1	$H^o_1 = 512$	tanh
	Hidden layer2	$H^o_2 = 256$	tanh
	output	$H^o_{out2} = 166$	sigmoid

IV. SIMULATION RESULTS

A. System Parameters

We assume that there are 6 edge servers and 40 mobile users in our simulations. The coverage of each edge server is 1km. Each user u_i moves at a fixed speed v_i, ranging from 20km/h to 120km/h. The CPU clock frequency of server e_l or f^e_l ranges from 3.19GHz to 19.14GHz [11]. The CPU clock frequency of user u_i or f^L_i ranges from 100MHz to 350MHz [11]. The duration of each time slot is set to 1ms. We randomly choose a slot to generate the first component task of each user. Once the first component task is generated, the subsequent component tasks are generated with the same time interval, where the interval is randomly chosen from [50ms, 200ms]. The data size of a component task is randomly picked between 200KB and 3MB, and the number of CPU cycles needed to complete this task is randomly picked between $6 * 10^9$ and $9 * 10^{10}$. The decision period D_l at e_l ranges from 2s to 12s.

We next detail the parameters used in algorithm iSMA. We set $I = 1000$, $J = 200$, $K = 10$ and $Q = 100$. The deterministic transition model in RLSM is implemented based on the LSTM network [12], where the number of cells used is $H^L_{cell} = 5$, and the dimension of the memory state is $H^L_h = 200$. A one-dimensional vector with size $(N + N * 5 + L) = 246$ is used in our simulations to represent the observed state $o^l_\tau = \{U^l_\tau, T^l_\tau, F_\tau\}$ at edge serer e_l. With such a one-dimensional vector as input, the parameters used in RSTM are given in Table I.

B. Performance Comparisons

We first compare iSMA with the SLSM-based approach, DLSM-based approach and a well-known algorithm DQN [6]. The SLSM-based or DLSM-based approach is similar to iSMA, but only using stochastic state s_τ or deterministic state h_τ to establish the latent space (see Section III.A). The existing algorithm DQN only considers a single user. For fair comparison, we extend DQN to support multiple users, where each edge server processes the requests received sequentially. Since the number of output layers in DQN exponentially increases with the number of inputs, the decision in DQN can only be made sequentially with one request at a time.

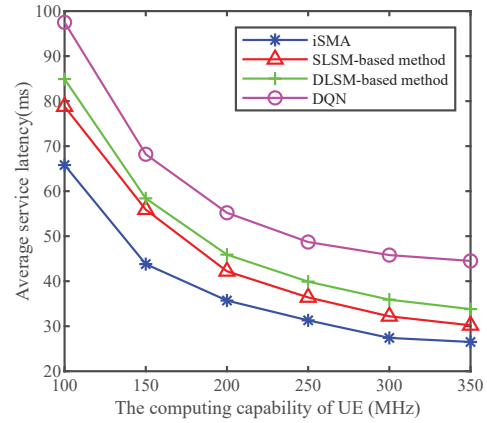

Fig. 3. Service delay under different local computing capability.

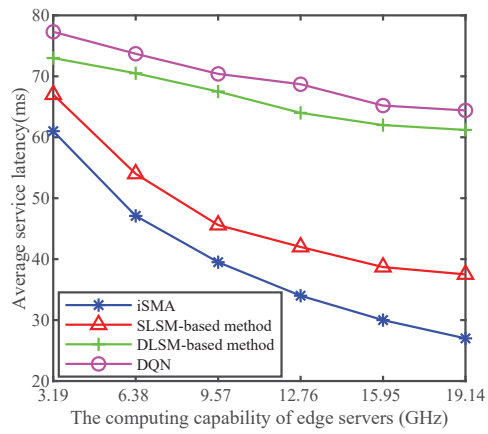

Fig. 4. Service delay under different computing capability of servers

Fig. 3 compares the long-term average service delay obtained under different computing capability of users (i.e., the local computing capability). In Fig. 3, we set D_l = 2s, v_i = 120km/h, and f^e_l = 15.95GHz. We can see that as the local computing capability increases, the average service delay is reduced for all mechanisms. We also find that the proposed iSMA provides the smallest service delay, and reduces the service delay by about 14.9%, 23.7% and 40.2% as compared to the SLSM-based approach, DLSM-based approach and DQN, which implies the efficiency of the proposed iSMA and RLSM model. DQN provides the worst performance because the decisions cannot be jointly made based on all the requests received at each edge server.

We next compare the long-term average service delay obtained under different computing capability of servers in Fig. 4. We set D_l = 2s, v_i = 120km/h and f^L_i = 150MHz in Fig. 4. Again, we can see that the service delay obtained using iSMA is the smallest. In particular, iSMA improves the service delay by about 28.1%, 55.9% and 58.1% as compared to the SLSM-based approach, DLSM-based and DQN, respectively.

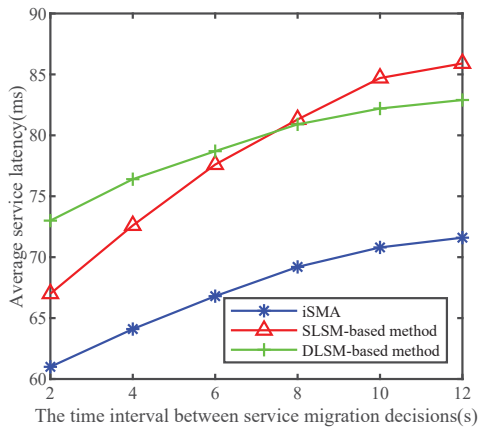

Fig. 5. The impact of decision period D_l

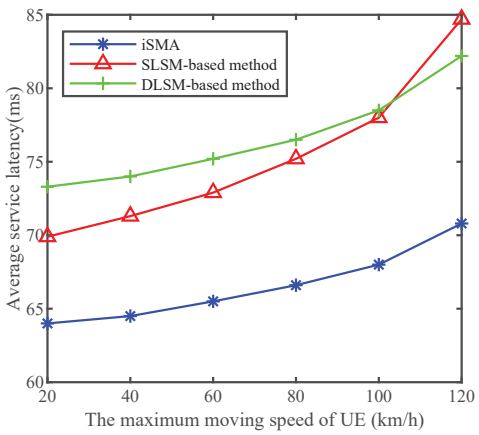

Fig. 6. The impact of user's moving speed v_i

We also find that as the server's computing capability increases, the improvement of service delay is limited when DLSM-based approach is used. This is because DLSM adopts a deterministic transition model to infer the hidden state, which can easily result in poor performance.

C. Impact of Decision Period and Moving Speed

When the moving speed of users (v_i) increases, or the decision period at server e_l (D_l) increases, the performance will be adversely affected. In this section, we examine the impact incurred. In Fig. 5, the long-term average service delays obtained using different algorithms are plotted versus the value of D_l. We set f_i^L = 150MHz, f_l^e = 3.19GHz and v_i = 120km/h in Fig. 5. We can see that as D_l increases, the service delay is increased for all mechanisms. This is because if iSMA is not executed frequently at the edge server (or D_l is large), the decisions cannot be updated timely, which in turn increases the service delay. We further notice that as D_l increases, RLSM provides the smallest performance degradation. That further verifies the efficiency of the proposed RLSM model in iSMA.

In Fig. 6, the average service delays obtained using different algorithms are plotted versus the moving speed of users. We set f_i^L = 150MHz, f_l^e = 3.19GHz and D_l = 10s in Fig. 6. We can see that the service delay increases with the user's moving speed (as expected). We also find that when the moving speed is 120km/h, the DLSM-based approach can outperform the SLSM-based approach. This is because more historical information is considered in the deterministic transition model (as can be seen from Section III.A), which can result in a more accurate prediction on user mobility.

V. Conclusion

In this paper, we investigated the joint problem of service migration and computation offloading in MEC. We formulated the problem as a POMDP based on the partially observable information, and proposed an intelligent algorithm iSMA to minimize the long-term average service delay of all users. Numerical results show that iSMA decreases the service delay by about 28.1%, 55.9% and 58.1% as compared to the SLSM-based approach, DLSM-based approach and the existing DQN.

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, Mobile edge computing: A survey, *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450465, 2018.

[2] S. Wang, J. Xu, N. Zhang, and Y. Liu, A survey on service migration in mobile edge computing, *IEEE Access*, vol. 6, pp. 2351123528, 2018.

[3] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, Dynamic service migration in mobile edge computing based on markov decision process, *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 12721288, 2019.

[4] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, Dynamic service migration in mobile edge clouds, *in 2015 IFIP Networking Conference (IFIP Networking)*, 2015, pp. 19.

[5] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, Migration modeling and learning algorithms for containers in fog computing, *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712725, 2019.

[6] Z. Gao, Q. Jiao, K. Xiao, Q. Wang, Z. Mo, and Y. Yang, Deep reinforcement learning based service migration strategy for edge computing, *in 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2019, pp. 1161165.

[7] J. Li, Q. Liu, P. Wu, F. Shu, and S. Jin, Task offloading for uav-based mobile edge computing via deep reinforcement learning, *in 2018 IEEE/CIC International Conference on Communications in China (ICCC)*, 2018, pp. 798802.

[8] T. Ouyang, Z. Zhou, and X. Chen, Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing, *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 23332345, 2018.

[9] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, Software-defined networking security: pros and cons, *IEEE Communications Magazine*, vol. 53, no. 6, pp. 7379, 2015.

[10] K. Guo, M. Sheng, J. Tang, T. Q. S. Quek, and Z. Qiu, Hierarchical offloading for delay-constrained applications in fog ran, *IEEE Transactions on Vehicular Technology*, vol. 69, no. 4, pp. 42574270, 2020.

[11] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, iRAF: A deep reinforcement learning approach for collaborative mobile edge computing iot networks, *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 70117024, 2019.

[12] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Computation*, vol. 9, no. 8, pp. 17351780, 1997.