
There may be differences between this version and the published version. You are advised to consult the publisher’s version if you wish to cite from it.

https://eprints.gla.ac.uk/270704/

Deposited on: 10 May 2022

Enlighten – Research publications by members of the University of Glasgow
https://eprints.gla.ac.uk
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
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 \( b_i \) 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 computa-
tional 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 gen-
erated 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 \( \tau \).
Since the task request is generated based on the data size \( c_i \),
\( T_i(\tau) \) in a time slot \( \tau \) can be NULL if the data collected is
less than \( c_i \). Otherwise, \( T_i(\tau) = \{c_i, f_i^L, f_i^r, g_i, \delta_i^{\text{max}}\} \), 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 \( \delta_i^{\text{max}} \) 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.) Mean-
while, 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 \( a_i = \{a_i(IP), a_i(\eta), a_i(f^r)\} \), where
\( a_i(IP) \) is the address of the edge server serving \( u_i \), \( a_i(\eta) \)
is the computation offloading ratio and \( a_i(f^r) \) 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 strat-
tegy 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 serv-
er currently associated with. Each edge server \( e_l \) stores the set
of its current serving users \( U_l^t \), the set of task requests received
\( T_i^t \) and the set of remaining computing resources at all edge
servers \( F_r \). In other words, \( U_l^t = \{u_i | u_i \text{ served by } e_l \text{ at } t = \tau\} \), \( T_i^t = \{T_i(\tau) | u_i \in U_l^t\} \).

Suppose user \( u_i \) sends its first task request \( T_i(\tau) \) to edge
server \( e_l \) at time slot \( \tau \). Server \( e_l \) then recognizes that \( u_i \) is
a newcomer, the user set \( U_l^t \) and request set \( T_i^t \) are then up-
dated, and the decision-making algorithm will be immediately
executed at server \( e_l \) with \( U_l^t \), \( T_i^t \) and \( F_r \) as input.

When the algorithm terminates, action \( a_i = \{a_i(IP), a_i(\eta), a_i(f^r)\} \) is obtained and sent to user \( u_i \).
No service migration if \( a_i(IP) \) is still server \( e_l \) itself; other-
wise, 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
dedge server and process the remaining \((1 - a_i(\eta))\) portion
of data locally. Once the \( a_i(\eta) \) portion of data is received,
the target edge server will use \( a_i(f^r) \) \((a_i(f^r) \leq 1)\) 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 \(e_i\) also updates its decisions every \(D_t\) time slots for all the requests received. The new service migration strategy (determined by \(a_i(\eta)\)) and computation offloading strategy (determined by \(a_i(\eta)\)) obtained will then be sent to the corresponding user. We call \(D_t\) a decision period of edge server \(e_i\). During the decision period of \(e_i\), the same service migration and computation offloading strategies will be applied to the component tasks generated by \(e_i\)’s current serving user. Obviously, the value of \(D_t\) 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

\[ t^L_i = \frac{f_i^L(1 - a_i(\eta))}{f_i^L}, \]

where \(f_i^L\) is the CPU clock frequency of user \(u_i\), and \(f_i^L\) 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 \(e_i\). With signal power \(p_i\) and channel gain \(g_i\), between \(u_i\) and \(b_j\), the channel capacity is

\[ R^w_i = B \log(1 + \frac{p_i g_i}{N_0}), \]

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

\[ t_i^{cw} = \frac{c_i a_i(\eta)}{R^w_i}, \]

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 \(e_i\) is \(R^f_i\). Then the transmission time from \(b_j\) to \(e_i\) is mainly determined by

\[ t_i^{cf} = \frac{c_i a_i(\eta)}{R^f_i} d_{i,j}, \]

where \(d_{i,j}\) is the number of hops needed from \(b_j\) to \(e_i\) via the Internet. The queuing delay is ignored by assuming the network is not heavily loaded.

Finally, the data processing time at server \(e_i\) is

\[ t_i^P = \frac{a_i(\eta) f_i^P}{f_i^L a_i(f^P)}, \]

where \(f_i^P\) is the CPU clock frequency of server \(e_i\) and \(a_i(f^P)\) is the computing resource ratio allocated for the task.

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

\[ t_i = \max(t_i^L, t_i^{cw} + t_i^{cf} + t_i^P). \]

Under the constraint that \(t_i\) is no longer than a given threshold \(\delta^t_{max}\), 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 \(A_r = \{A^r_l, l = 1, 2, ..., L\}\) denote the actions for the \(N\) users at time slot \(\tau\), where \(A_r = \{a_l u_i \in U^l_i\}\) is the action set obtained by server \(e_i\) at time slot \(\tau\). Let \(o^r_{\tau} = \{U^l_{\tau}, T^l_{\tau}, F^l_{\tau}\}\) be the observed state of \(e_i\) at time slot \(\tau\), which is a partial state of the environment. Then we formulate the problem as a POMDP:

- Transition model: \(s_{\tau+1}^l \sim p(s_{\tau+1}^l | s^l_{\tau-1}, A^l_{\tau-1})\)
- Observation model: \(o_{\tau}^l \sim p(o_{\tau}^l | s^l_{\tau})\)
- Reward model: \(r_{\tau}^l \sim p(r_{\tau}^l | s^l_{\tau}, A^l_{\tau})\)
- Action: \(A^l_{\tau} \sim \pi^l(A^l_{\tau} | o^l_{\tau}, A^l_{<\tau})\)
- Encoder: \(s_{\tau}^l \sim p(s_{\tau}^l | o^l_{\tau}, A^l_{<\tau})\)

where \(s_{\tau}^l\) denotes the hidden state (or unobserved state) of the environment inferred by server \(e_i\) at time slot \(\tau\), and \(r_{\tau}^l\) is a reward obtained based on action set \(A^l_{\tau}\) and hidden state \(s_{\tau}^l\). Based on the partial observed states \(o^l_{\tau}\) and previous actions \(A^l_{<\tau}\), server \(e_i\) 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 \(A^l_{0,1}, A^l_{2,3,4}, \ldots, A^l_{n-1,n}\) 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:

\[
P : A_0, \ldots, A_\tau, \ldots, A_{\infty} \rightarrow \arg\min_{A_0, \ldots, A_\tau, \ldots, A_{\infty}} \left\{ \sum_{\tau=0}^{\infty} \left( \frac{1}{N} \sum_{i=1}^{N} t_i^t \right) \right\}
\]

\[
= \arg\min_{A_0, \ldots, A_\tau, \ldots, A_{\infty}} \left\{ \sum_{\tau=0}^{\infty} \frac{1}{N} \sum_{i=1}^{N} \max(t_i^L, t_i^{cw} + t_i^{cf} + t_i^P) \right\}
\]

\[
= \arg\min_{A_0, \ldots, A_\tau, \ldots, A_{\infty}} \left\{ \sum_{\tau=0}^{\infty} \frac{1}{N} \sum_{i=1}^{N} \max\left( (1 - a_i(\eta)) f_i^P, c_i a_i(\eta) R_i^{cw} + c_i a_i(\eta) d_{i,j} + \frac{f_i^P a_i(\eta) f_i^L}{f_i^L a_i(f^P)} \right) \right\}
\]
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 used 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 $o_\tau$, action set $A_\tau$ and hidden state $s_\tau$ obtained by edge server $e_l$ are simplified to $o_\tau$, $A_\tau$ and $s_\tau$ 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 $s_\tau$ and a deterministic state $h_\tau$, where $s_\tau$ is modelled as a Gaussian variable, and $h_\tau$ 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:
  \[ r_\tau \sim p(r_\tau|h_\tau, s_\tau) \]
- 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:

\[ \theta = \arg \max_\theta \{ \ln p(o_{1:K}|A_{1:K}) + \ln p(r_{1:K}|A_{1:K}) \} \tag{8} \]

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 \( \{A_{\tau:t+K}\} \) and output the best one with the maximum reward, where \( A_{\tau:t+K} \) denotes the actions from \( t = \tau \) to \( t = \tau + K \). To be more specific, with the observed state $o_\tau$ as input, isMA consists of the following 9 steps:

1) Initialize the distribution of the action sequences with normal distribution, i.e., $A_{\tau:t+K} \sim \text{Normal}(0, I)$.
2) With $A_{\tau:t+K} \sim \text{Normal}(0, I)$, a possible action set $A_\tau$ is obtained.
3) With the observed state $o_\tau$ and action set $A_\tau$, a hidden state $h_\tau$ and reward $r_\tau$ 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 $A_{\tau:t+K}$ with the corresponding reward sequence $r_{\tau:t+K}$ is then obtained. We evaluate the action sequence $A_{\tau:t+K}$ based on (9):

\[ R^i = \sum_{k=\tau}^{\tau+K} \gamma^{k-\tau}r_k \tag{9} \]

5) Repeat step 4) $I$ times to get a set of action sequences \( \{A^i_{\tau:t+K}\}_{i=1}^I \), where each $A^i$ is an action sequence $A_{\tau:t+K}$. Similarly, a set of rewards \( \{R^i_{\tau:t+K}\}_{i=1}^I \) is obtained, where each $R^i_{\tau:t+K}$ is calculated based on (9).
6) Sort set \( \{A^i_{\tau:t+K}\}_{i=1}^I \) from the largest reward $R^i_{\tau:t+K}$ to the smallest. The first $Q (Q < I)$ action sequences are then added to a new set $\mathcal{A}$, i.e.,

\[ \mathcal{A} \leftarrow \arg \text{sort}(\{A^i_{\tau:t+K}\}_{i=1}^I) \tag{10} \]

7) The mean and variance of set $\mathcal{A}$ 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., $A_{\tau:t+K} \sim \text{Normal}(\mu_{\tau:t+K}, \sigma_{\tau:t+K} I)$.

\[ \mu_{\tau:t+K} = \frac{1}{Q} \sum_{A^i \in \mathcal{A}} A^i_{\tau:t+K} \tag{11} \]

\[ \sigma_{\tau:t+K} = \sqrt{\frac{1}{Q-1} \sum_{A^i \in \mathcal{A}} (A^i_{\tau:t+K} - \mu_{\tau:t+K})^2} \tag{12} \]

8) Repeat steps 2) – 7) $J$ times with the new distribution $A_{\tau:t+K} \sim \text{Normal}(\mu_{\tau:t+K}, \sigma_{\tau:t+K} I)$.
9) The final mean calculated based on (11) is selected as the best action sequence, i.e., $A^* = A_{\tau:t+K} = \mu_{\tau:t+K}$.

The action set $A_\tau$ 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.
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 1 km. Each user \( u \) moves at a fixed speed \( v_l \), ranging from 20 km/h to 120 km/h. The CPU clock frequency of server \( e_i \) or \( f^e_i \) ranges from 3.19 GHz to 19.14 GHz [11]. The CPU clock frequency of user \( u_i \) or \( f^u_i \) ranges from 100 MHz to 350 MHz [11]. The duration of each time slot is set to 1 ms. 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 [50 ms, 200 ms]. The data size of a component task is randomly picked between 200 KB and 3 MB, and the number of CPU cycles needed to complete this task is randomly picked between \( 2 \times 10^5 \) and \( 9 \times 10^{10} \). The decision period \( D_l \) at \( e_i \) ranges from 2 s to 12 s.

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 \( H^L_{mem} = 200 \). A one-dimensional vector with size \( (N + N \times 5 + L) = 246 \) is used in our simulations to represent the observed state \( o^l_e = \{ \mu^l_e, \sigma^l_e \} \) at edge server \( e_i \). With such a one-dimensional vector as input, the parameters used in RSTM are given in Table I.


<table>
<thead>
<tr>
<th>Network</th>
<th>Layer</th>
<th>Hidden units</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder model</td>
<td>Input</td>
<td>( H^L_{in} = H^L_{in} + 246 )</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>( \mu^L_{output} )</td>
<td>( H^L_{out1} = 60 )</td>
<td>tanh</td>
</tr>
<tr>
<td></td>
<td>( \sigma^L_{output} )</td>
<td>( H^L_{out2} = 60 )</td>
<td>sigmoid</td>
</tr>
<tr>
<td>Stochastic transition model</td>
<td>Input</td>
<td>( H^L_{in} = 200 )</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>( \mu^L_{output} )</td>
<td>( H^L_{out1} = 512 )</td>
<td>tanh</td>
</tr>
<tr>
<td></td>
<td>( \sigma^L_{output} )</td>
<td>( H^L_{out2} = 60 )</td>
<td>sigmoid</td>
</tr>
<tr>
<td>Reward model</td>
<td>Input</td>
<td>( H^L_{in} = 60 + 200 )</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>( \mu^L_{output} )</td>
<td>( H^L_{out1} = 512 )</td>
<td>relu</td>
</tr>
<tr>
<td></td>
<td>( \sigma^L_{output} )</td>
<td>( H^L_{out2} = 60 )</td>
<td>relu</td>
</tr>
<tr>
<td>Observation model</td>
<td>Input</td>
<td>( H^L_{in} = 60 + 200 )</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>( \mu^L_{output} )</td>
<td>( H^L_{out1} = 512 )</td>
<td>tanh</td>
</tr>
<tr>
<td></td>
<td>( \sigma^L_{output} )</td>
<td>( H^L_{out2} = 256 )</td>
<td>tanh</td>
</tr>
</tbody>
</table>

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 \( s_r \) or deterministic state \( h_r \) 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 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 \( D_l = 2 s \), \( v_l = 120 \) km/h, and \( f^u_i = 15.95 \) GHz. 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 \( D_l = 2 s \), \( v_l = 120 \) km/h and \( f^u_i = 150 \) MHz 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.
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 \(c_t\) \((D_t)\) 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 \(D_t\). We set \(f_{L}^{t} = 150\text{MHz}, f_{E}^{t} = 3.19\text{GHz}\) and \(v_i = 120\text{km/h}\) in Fig. 5. We can see that \(D_t\) increases, the service delay is increased for all mechanisms. This is because if iSMA is not executed frequently at the edge server (or \(D_t\) is large), the decisions cannot be updated timely, which in turn increases the service delay. We further notice that as \(D_t\) 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 \(f_{L}^{t} = 150\text{MHz}, f_{E}^{t} = 3.19\text{GHz}\) and \(D_t = 10s\) 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