A DQN-Based Approach for Online Service Placement in Mobile Edge Computing

Xiaogan Jie[1], Tong Liu[1,2,3,4(✉)], Honghao Gao[1], Chenhong Cao[1,3], Peng Wang[1], and Weiqin Tong[1,3,4]

[1] School of Computer Engineering and Science, Shanghai University, Shanghai, China
{jiegan,tong_liu,gaohonghao,caoch,pengwang,wqtong}@shu.edu.cn
[2] Shanghai Key Laboratory of Data Science, Shanghai, China
[3] Shanghai Institute for Advanced Communication and Data Science, Shanghai University, Shanghai, China
[4] Shanghai Engineering Research Center of Intelligent Computing System, Shanghai, China

Abstract. Due to the development of 5G networks, computation intensive applications on mobile devices have emerged, such as augmented reality and video stream analysis. Mobile edge computing is put forward as a new computing paradigm, to meet the low-latency requirements of applications, by moving services from the cloud to the network edge like base stations. Due to the limited storage space and computing capacity of an edge server, service placement is an important issue, determining which services are deployed at edge to serve corresponding tasks. The problem becomes particularly complicated, with considering the stochastic arrivals of tasks, the additional latency incurred by service migration, and the time spent for waiting in queues for processing at edge. Benefiting from reinforcement learning, we propose a deep Q network based approach, by formulating service placement as a Markov decision process. Real-time service placement strategies are output, to minimize the total latency of arrived tasks in a long term. Extensive simulation results demonstrate that our approach works effectively.

Keywords: Mobile edge computing · Service placement · Deep reinforcement learning

1 Introduction

With the emergence of 5G communication technology and the proliferation of mobile smart devices, many new computation and data intensive applications with low latency requirements have come forth to mobile devices, such as online interactive games, augmented reality and video stream analysis [2,9,19]. Unfortunately, traditional cloud computing cannot meet the requirements of the applications, due to the long data transmission time. To address the issue, mobile

H. Gao et al. (Eds.): CollaborateCom 2020, LNICST 350, pp. 169–183, 2021.
https://doi.org/10.1007/978-3-030-67540-0_10

edge computing [7,10] is put forward to provide services to the applications, by deploying computing resources at the edge of Internet. A mobile edge computing system enabled by 5G networks is shown in Fig. 1, in which base stations endowed with servers act as edge nodes.

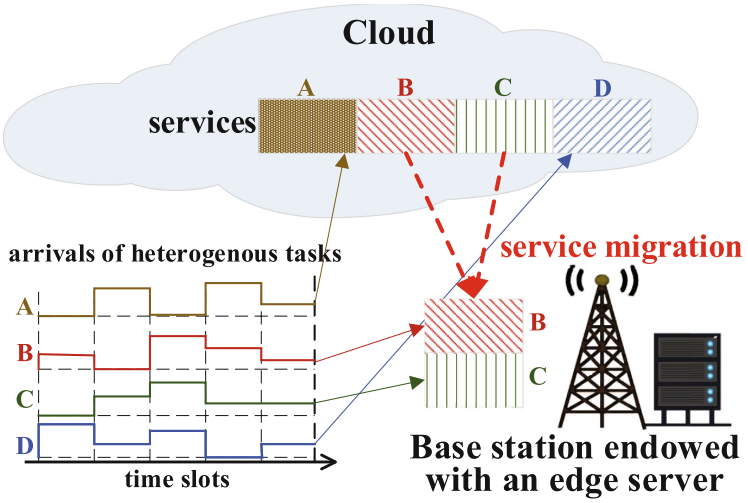

Fig. 1. An illustration of the 5G-enabled mobile edge computing system, in which base stations endowed with servers act as edge nodes.

To reduce the response time, computation tasks generated by different applications on mobile devices should be offloaded to an edge server for processing, instead of offloaded to the cloud. However, to serve the tasks of a particular application, the corresponding service should be placed on the edge server in advance, including installing required softwares and caching related databases/libraries. Unlike the cloud, both storage space and computing capacity of an edge server are extremely limited. Thus, service placement is an important issue in mobile edge computing, i.e., determining which services should be deployed at the edge. Particularly, the problem becomes complicated with the consideration of heterogenous tasks randomly arriving to the base station in real time. An online optimal service placement strategy is desired, to achieve the minimal response time of all tasks.

Some existing works have paid attention to the service placement in mobile edge computing. Most of the works focus on designing service placement policies to improve the quality of service. In [3,6,15,18], the authors consider a stationary environment and propose several offline service placement algorithms to determine which services are selected and where they are placed. And these works achieve provable close-to-optimal performance. In addition, some works [1,4,12,14,21] consider the varying demands of computation requests arriving to a mobile edge computing system in real time. In [1,8,12,17], the authors propose

online service placement schemes, assuming the future dynamics like user mobility can be predicted in advance or make a milder assumption that user mobility follows a Markov chain. However, it is hard to obtain the future dynamics as prior knowledge in practice. Some other works [4,5,13,14,21] solving the service placement problem without a priori knowledge of future dynamics.

In this work, to make optimal service placement decisions in real time is non-trivial due to the following challenges. *Firstly*, the tasks generated by various applications request differential computing resources, and the arrivals of tasks randomly vary over time. Dynamic service placement strategy is desired, adapting to the changing demands of tasks, since only the tasks served by placed services can be executed at the edge. *Secondly*, different services request distinguishing storage for caching the corresponding softwares and databases. Thus, the placements of different services are interactive. Moreover, the set of services to be placed on the edge server is extremely constrained by its storage space. *Thirdly*, additional latency is incurred for migrating a service from the cloud to the edge server, including the time spent for transmitting and operating. This latency can be omitted if the service has been placed in the previous time slot. Thus, the service placement strategies in two successive time slots are coupled with each other. It is hard to derive the optimal strategy without any future information.

To tackle the difficulties, we propose an online learning-based approach for the base station to make real-time service placement decisions, aiming to minimize the total latency of arrived tasks in a long term. Specially, we first analyze the switching latency, serving latency, and offloading latency incurred by a service placement decision respectively, and formulate the service placement problem as a non-linear optimization with coupled constraints. Due to the superiority of reinforcement learning in making controls in dynamic and random environments, we model service placement as a Markov decision process and adopt the deep Q network (DQN) method, in which the reward achieved by a service placement decision is approximated by a deep neural network. The main contributions of this work are summarized as follows:

- We formulate the online service placement in a 5G-enabled mobile edge computing system as a Markov decision process. We consider stochastic arrivals of heterogenous tasks which are served by different services, and the limited storage space and computing capacity of the edge server.
- We propose a deep reinforcement learning based approach to obtain the optimal service placement strategy, with the objective of minimizing the total latency of tasks in a long term.
- Extensive simulations are conducted to evaluate the performance of our proposed approach, compared with several baselines. The simulation results confirm that our approach performs better than baselines.

The rest of this paper is organized as follows. Section 2 reviews related works. In Sect. 3, we describe the system model and problem formulation. Then, we propose our approach in Sect. 4. The results of extensive simulations are shown in Sect. 5, followed by the conclusion in Sect. 6.

2 Related Work

Considering the limited resources of the edge server, some exiting works [1,3–6,8,12–15,17,18,21] have studied on service placement in mobile edge computing. We roughly classify related works into two categories, according to whether the dynamic nature of an edge computing system is considered.

With considering a stationary environment, several offline service placement algorithms are proposed, to determine which services are selected and where they are placed. Specially, both [15] and [6] focus on the joint service placement and request scheduling problem, aiming to maximize the number of computing requests processed by edge servers. A near-optimal algorithm with a constant approximation factor is provided by each of them, respectively. Ascigil *et al.* [3] propose a set of uncoordinated strategies for solving the service placement problem, with considering multiple services with different latency deadlines. However, the above works do not take the operation cost incurred by service migration into account. Differently, Wang *et al.* [18] jointly optimize the service activation cost and service placement cost of an edge cloud provider, which provisions a social VR application based on edge clouds. The authors convert the cost optimization into a graph cut problem and employ a max-flow algorithm.

Considering the varying demands of computation requests arriving to a mobile edge computing system, services placed on each edge server should be adapted over time. To tackle the challenge, several works [1,12] propose online service placement schemes, assuming the future dynamics like user mobility can be predicted in advance. Some other works [8,17] make a milder assumption that user mobility follows a Markov chain. However, all the works consider future dynamics as prior knowledge, which is hard to obtain in practice.

Besides, there are several existing works solving the service placement problem without a priori knowledge of future dynamics. Xu *et al.* [21] adopt the Lyapunov optimization to solve the service placement problem as a queue control problem, which lacks taking service migration cost into consideration. Differently, Farhadi *et al.* [4] takes operation cost incurred by service placement into consideration. To balance the cost and the performance of serving requests, a two-time-scale framework for joint service placement and request scheduling is proposed. By taking advantage of set function optimization, a greedy-based service placement algorithm based on shadow request scheduling is proposed, which achieved constant-factor approximation. Similarly, performance-cost tradeoff is pursued in [14] with in view of the operational cost of service migration. Considering the long-term cost budget constraint, the authors design an online service placement algorithm based on Markov approximation and Lyapunov optimization. In addition, a time-efficiency distributed scheme is also provided based on the best response update technique. In [5], Gao *et al.* jointly study the access network selection and service placement, with the objective to balance the access, switching and communication delay. An efficient online framework is proposed for decomposing the long-term optimization and an iteration-based algorithm is designed to derive a computation-efficient solution. Ouyang *et al.* [13] propose a multi-armed bandit and Thompson-sampling based online learning algorithm to

make optimal service placement strategy, which aims to optimize the completion latency of tasks and overhead caused by service migration.

Different from the above works, we jointly consider the switching latency incurred by service migration and the waiting time spent by tasks for processing at the edge. As the power of reinforcement learning has been witnessed in complex control under stochastic and dynamic environments, we propose a DQN-based online service placement approach without knowing future dynamics of the mobile edge computing system.

3 System Model and Problem Formulation

3.1 System Model

In this work, we consider a 5G-enabled mobile edge computing system, in which base stations endowed with servers act as edge nodes. Numerous heterogenous computation-intensive tasks arrive to the base stations in real time. For the convenience of modeling an online system, we discretize time into a set of equal-interval time slots, denoted by $\mathcal{T} = \{1, 2, \cdots, T\}$. The interval of each time slot is represented as Δt.

To execute the tasks of a certain type on an edge server, the corresponding service should be placed firstly. Specially, a service is an abstraction of an application, such as video stream analysis, interactive gaming, and augmented reality. To run a particular service, an amount of associated data should be cached by the edge server, including the software and database required by the application. Only the services cached on the edge server can be employed to serve the corresponding tasks.

We consider a library of L services are provided by the mobile edge computing system, denoted by $\mathcal{L} = \{1, 2, \cdots, L\}$. We assume different services have various amounts of associated data and require different CPU frequencies to process tasks. Specially, we use s_l and f_l to denoted the associated data size and the required CPU frequency of service $l \in \mathcal{L}$, respectively. Initially, all the services are stored on a remote cloud, as shown in Fig. 1. Each edge server can manually migrate one or more services from the cloud to its locality in an online manner. However, which services can be placed is restricted by the limited storage space and computing ability of an edge server. Generally, we denote the maximal storage space and CPU frequency of the edge server on a base station as s_{max} and f_{max}, respectively.

In each time slot $t \in \mathcal{T}$, we assume that the arrival of tasks served by service l follows a Poisson process at rate n_l^t. It means the average number of tasks served by l arriving in time slot t equals to n_l^t. The overall computation demand arriving to the base station in time slot t is noted as $\mathbf{n}^t = [n_1^t, n_2^t, \cdots, n_L^t]$. Besides, we assume the required CPU cycles of each task served by service l follows an exponential distribution with mean c_l. Thus, the computing time of such a task also follows an exponential distribution with mean c_l/f_l, if it is processed on the edge server.

According to the real-time arrivals of heterogenous tasks and their computation requirements, the base station needs to make the service placement decision online. In other words, the base station should determine which services to be placed on the edge server in the current time slot. To be specific, the services have been placed in the last time slot can be kept or removed according to the current decision. On the other hand, the unplaced services need to be migrated from the cloud.

We employ a binary variable $I_l^t \in \{0, 1\}$ to indicate whether service l is determined to be placed on the edge server or not in time slot t. The service placement decision in t can be represented by vector $\mathbf{I}^t = [I_1^t, I_2^t, \cdots, I_L^t]$. Specially, if service l is placed on the edge server in time slot t, then $I_l^t = 1$; Otherwise, $I_l^t = 0$. Due to the limited storage space and computing capacity, there exist

$$\sum\nolimits_{l=1}^{L} I_l^t \cdot s_l \leq s_{max}, \forall t \tag{1}$$

$$\sum\nolimits_{l=1}^{L} I_l^t \cdot f_l \leq f_{max}, \forall t \tag{2}$$

3.2 Service Placement

In this work, we consider the *switching latency*, the *serving latency*, and the *offloading latency* incurred by different service placement decisions, which are described in detail as follows.

Switching Latency. To migrate a particular service from the remote cloud to the edge server, a certain amount of time is spent for transmitting its associated data, which is referred as the switching latency in our work. Specially, for service l, we assume the switching latency in time slot t is ψ_l^t, which depends on the data size of service l and the network condition between the base station and the cloud in time slot t.

According to the service placement decision in the last time slot $\mathbf{I}^{t-1}$, we can derive the overall switching latency incurred by a current service placement decision $\mathbf{I}^t$ as

$$\Psi^t = \sum\nolimits_{l=1}^{L} \psi_l^t \cdot \mathbb{1}_{\{I_l^t - I_l^{t-1} = 1\}}, \tag{3}$$

where $\mathbb{1}_{\{\cdot\}}$ is an indicator function. If the condition in the brace is satisfied, then the value of the indicator function equals to one; otherwise, it equals to zero. Note that only the services unplaced in the last time slot but needed in the current time slot will incur the switching latency.

Serving Latency. Note that only the tasks served by the services placed on the edge server can be processed on the base station, while the other tasks have to be offloaded to the remote cloud for execution. We firstly analyze the latency incurred for completing a task on the edge server, which is referred as the serving latency. The serving latency of a task is defined as the period from when the task arrives to the base station to when the task is completed, which consists of

the waiting time and the computing time. We employ a specific M/M/1 queue to model the execution process of a task served by service l on the edge server, as both the arriving interval and the computing time of the task obey exponential distributions. And hence, the average serving time of the task (known as sojourn time in queuing theory) is

$$\omega_l^t = \frac{1}{f_l/c_l - n_l^t/\Delta t} = \frac{c_l \Delta t}{f_l \Delta t - n_l^t c_l}. \tag{4}$$

To make Eq. (4) sensible, we guarantee that serving rate f_l/c_l is greater than arriving rate $n_l^t/\Delta t$, by offloading extra tasks served by service l to the cloud.

Therefore, the overall serving latency incurred by a service placement decision $\mathbf{I}^t$ in time slot t can be obtained, i.e.,

$$\Omega^t = \sum\nolimits_{l=1}^{L} I_l^t \cdot n_l^t \cdot \omega_l^t. \tag{5}$$

Offloading Latency. We consider the cloud has all services and adequate computing resource, and hence the executing time of a task offloaded to the cloud can be ignored. For a task whose required service l is not placed on the edge server, we assume the expected time spent for transmitting its input data to the cloud (referred as the offloading latency) is ϕ_l^t. Obviously, it depends on the input data size and the network condition between the base station and the cloud in t. Therefore, the overall offloading latency incurred by a service placement decision $\mathbf{I}^t$ in time slot t can be obtained as

$$\Phi^t = \sum\nolimits_{l=1}^{L} (1 - I_l^t) \cdot n_l^t \cdot \phi_l^t. \tag{6}$$

3.3 Problem Formulation

In this work, we focus on the online service placement problem in a mobile edge computing system, where heterogenous computation-intensive tasks stochastically arrive in real time. We try to find an optimal service placement strategy for the base station, aiming to minimize the total latency of all tasks over time, i.e.,

$$\begin{aligned} \min_{\mathbf{I}^t} \ & \sum\nolimits_{t=1}^{T} \Psi^t + \Omega^t + \Phi^t \\ \text{s.t.} \ & \sum\nolimits_{l=1}^{L} I_l^t s_l \leq s_{max}, \forall t, \\ & \sum\nolimits_{l=1}^{L} I_l^t f_l \leq f_{max}, \forall t. \end{aligned}$$

Without complete future information, e.g., stochastic task arrivals and dynamic network conditions, it is difficult to obtain the optimal service placement strategy. Moreover, as the switching latency not only depends on the current service placement decision but also the previous service placement, there

exists a trade-off between migrating unplaced services or offloading tasks to the cloud, in terms of the total latency incurred. Specially, service migration introduces certain switching latency, while the execution latency of tasks can be reduced.

To this end, we propose an online learning-based approach for the problem via adopting reinforcement learning [22], which is applicable for decision making in stochastic and dynamic environments without any future information in prior.

4 Approach Design

To adopt reinforcement learning, we firstly reformulate the online service placement problem as a Markov Decision Process (MDP). Then, we propose a DQN-based online service placement approach.

4.1 Reformulated MDP Problem

A typical MDP model mainly consists of state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function R. In what follows, we formally model the online service placement problem as a MDP, by defining each component mentioned above.

State Space. Each state $s^t \in \mathcal{S}$ represents the information observed from the mobile edge computing system in time slot t. Specially, state s^t contains the service placement in the last time slot, the arrivals of heterogenous tasks in the current time slot, and the current network condition, i.e., $s^t \triangleq (\mathbf{I}^{t-1}, \mathbf{n}^t, B^t)$. Here, B^t represents the real-time bandwidth of the channel between the base station and the cloud.

Action Space. In each time slot t, the base station needs to make a service placement decision according to the current state, determining which services placed on the edge server, i.e., $a^t \triangleq \mathbf{I}^t$, and hence $\mathcal{A} = \{0, 1\}^{\times_L}$.

Reward Function. After taking action a^t under state s^t, the base station will receive a reward r^t defined as the total latency incurred in time slot t, i.e.,

$$r^t = -(\Psi^t + \Omega^t + \Phi^t). \tag{7}$$

Obviously, minimizing the overall latency is equivalent to maximizing the reward.

4.2 DQN-based Approach Design

Given the MDP model defined above, we can adopt reinforcement learning to obtain the optimal service placement policy. A feasible model-free reinforcement learning solution is Q-learning [20], which is widely used for making controls in a dynamic environment without any prior knowledge. In Q-learning, Q-function

$Q(s^t, a^t)$ is introduced to represent the estimated cumulative reward achieved by taking action a^t under state s^t.

$$Q(s^t, a^t) \triangleq \mathbb{E}\left[\sum_{\tau=t}^{T} \gamma^{\tau-t} \cdot r^{\tau}\right], \tag{8}$$

where $\gamma \in [0, 1]$ is a discount factor to indicate the degree of future rewards we look into. An optimal action a^t can be obtained by maximizing the Q-function.

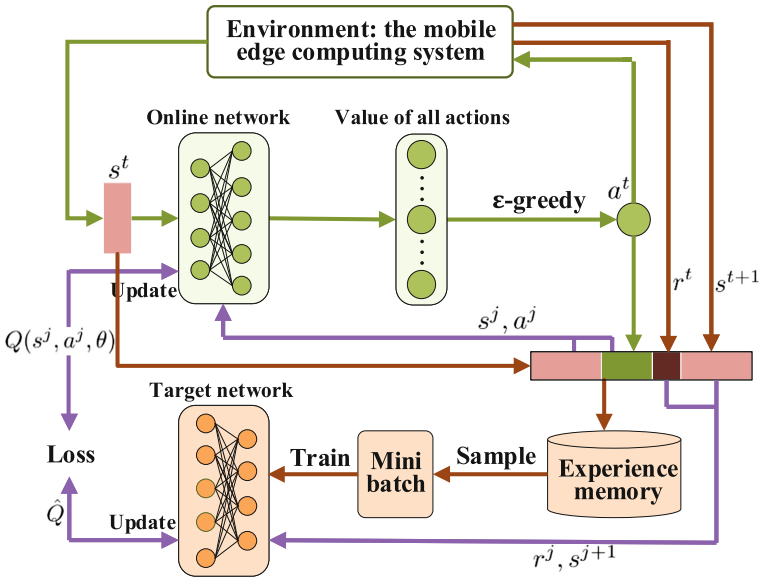

Fig. 2. The workflow and training process of our proposed DQN-based approach.

However, Q-learning maintains the Q-function value of each state-action pair in a Q-table, which will enlarge explosively with the increase of the state and action space. In our MDP model, the number of state-action pairs is infinite, as the number of arrived tasks and the network condition vary continuously. Therefore, Q-learning cannot be used to solve the online service placement problem. To tackle this challenge, the DQN solution [11] has been developed, in which the Q-function of each state-action pair is estimated by a deep neural network (DNN). In this work, we design a DQN-based approach for the online service placement problem.

Approach Workflow. The workflow of our proposed approach is presented in Fig. 2. In each time slot t, the base station firstly observes the current state s^t of the mobile edge computing system, and input it to a DNN with parameters θ which is called the *online network*. Then, the network outputs the Q-function

value of each action. At last, the base station selects an action a^t according to the ϵ-greedy algorithm [16]. To be specific, the action with the maximal value $Q(s^t, a^t; \theta)$ is picked by the edge server with probability $1 - \epsilon$, which is referred as *exploitation*. On the other hand, the edge server also has the chance to randomly chooses an arbitrary action with probability ϵ to prevent being trapped in the local optimum. This way is referred as *exploitation*. Parameter $\epsilon \in (0, 1)$ is tunable to balance the tradeoff between exploitation and exploration. After taking action a^t, a reward r^t is returned by the environment and the state transforms to s^{t+1}. The process is recorded by a tuple (s^t, a^t, r^t, s^{t+1}), referred as an *experience*, which is stored in the memory of the edge server.

Network Training. We train the parameters of the online network based on historical experiences. Firstly, we randomly sample a mini batch of experiences from the memory, and input them to the online network and another DNN which has the same structure with the online network but different parameters θ', referred as the *target network*, as shown in Fig. 2. The target network is used to estimate the ground truth of Q-function values, to help train the parameters of the online network.

We take one sampled experience (s^j, a^j, r^j, s^{j+1}) as an example to illustrate the training process. Specially, we can obtain the Q-function value estimated by the online network given s^j and a^j, i.e., $Q(s^j, a^j; \theta)$. Then, s^{j+1} is input to the target network and the target Q-function value $\hat{Q}$ can be obtained according to the Bellman equation [22], i.e.,

$$\hat{Q} = r^t + \gamma \cdot \max_a Q(s^{j+1}, a; \theta'). \tag{9}$$

In order to reduce the difference between $Q(s^j, a^j; \theta)$ and $\hat{Q}$, we define the loss function to train the online network as follows,

$$Loss = (\hat{Q} - Q(s^j, a^j; \theta))^2. \tag{10}$$

We leverage the gradient descent algorithm to minimize the loss function. And hence, the parameters θ can be updated as

$$\theta \leftarrow \theta + \eta(Q(s^j, a^j; \theta) - \hat{Q})\nabla Q(s^j, a^j; \theta), \tag{11}$$

where η is the learning rate.

To maintain the stability of the training process, we update the parameters of the online network and the target network asynchronously. To be specific, the parameters of the online network are updated immediately after each training process. Whereas, the target network copy the parameters of the online network after a certain time slots.

5 Performance Evaluation

5.1 Simulation Setup

We conduct extensive simulations to evaluate the performance achieved by our proposed approach, compared with four baselines listed as follows.

- *Cloud Processing Only Approach*: All tasks are offloaded to the remote cloud directly, which has all services and adequate computing resources.
- *Stochastic Approach*: Each service is randomly determined to be placed on the edge server or not, with considering the storage space and CPU frequency constraints.
- *Service-prior Greedy Approach*: As many as possible services are placed on the edge server, under the storage space and CPU frequency limitations.
- *Task-prior Greedy Approach*: Services are placed on the edge server one by one. The service which can serve the most tasks in the current time slot is selected each time, unless the resource constrains cannot be satisfied.

We evaluate the performance of different approaches according to the their achieved rewards, i.e., the total latency of all tasks.

The default values of parameters in our simulations are set as follows. There are four types of services. We set the associated data size s_l and the required CPU frequency f_l of each service are within [30,40] GB and [2,3] GHz, respectively. The maximal storage space and CPU frequency of the edge server are set as s_{max} =100 GB and f_{max} =5 GHz, respectively. Besides, the required CPU cycles of each task served by service l follows an exponential distribution with mean $c_l \in [0.02, 0.03]$ GHz. The switching latency of migrating service l from the cloud to the base station is $\psi_l^t \in [0.1, 0.4]$ seconds, and the latency for offloading a task served by service l to the cloud is $\phi_l^t \in [0.5, 1]$ seconds. We set the number of tasks served by l arriving in time slot t, i.e., n_l^t, is uniformly distributed in [50,100]. The interval of each time slot is set as 1 s. In addition, we set each episode in the training phase of our approach equals to 250 time slots, in which a whole interacting process between the base station and the environment is completed. In default, the parameters of the target network are updated once when the parameters of the online network have been updated 40 times (referred as *update frequency* in our evaluation results). Besides, the batch size sampled for training is set as 64. Discount factor γ to 0.9, and learning rate η is 0.005.

5.2 Evaluation Results

We firstly study the impact of the batch size sampled for training and the update frequency of the target network to the convergence speed of the training process in our approach, as shown in Fig. 3 and Fig. 4 respectively. Figure 3 plots the rewards achieved by our approach varying with the episode, when the batch sizes are 64, 128, and 256, respectively. We can find that when the batch size equals to 64, the training process of our approach can converge faster than when the batch size is 128 or 256. This is because the smaller batch size, the steeper direction of the gradient descent. In other words, the parameters of the online network can be updated more quickly, according to Eq. (11). Moreover, a larger batch size consumes more time for training. Hence, we set the default batch size as 64 in the training phase. Figure 4 presents the rewards obtained by our approach changing with the episode, when the update frequencies are 40, 80, and 120, respectively. Apparently, the larger update frequency, the slower convergence

speed of the training process. In order to make a tradeoff between maintaining the stability of the training process and achieving faster convergence, we set the update frequency as 40.

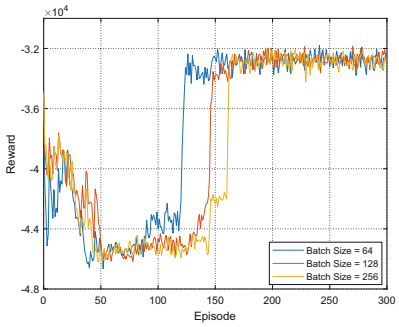

Fig. 3. Convergence v.s. batch size.

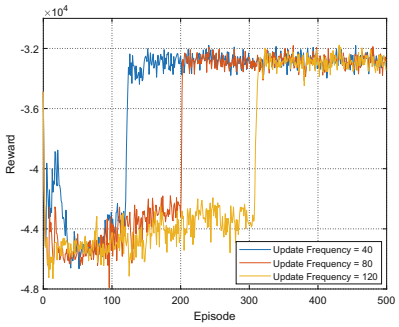

Fig. 4. Convergence v.s. update frequency.

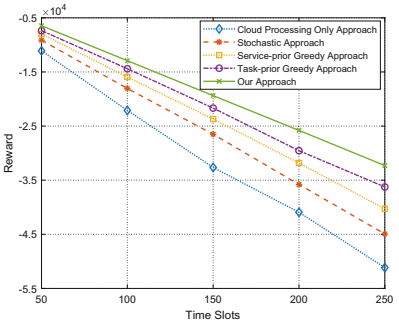

Fig. 5. Reward v.s. number of time slots.

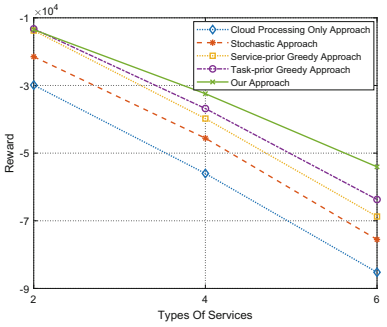

Fig. 6. Reward v.s. number of services.

Next, we evaluate the performance of our proposed approach, compared with the four baselines, under different parameter settings. As shown in Fig. 5, we plot the rewards achieved by different approaches when the number of time slots varies from 50 to 250. The more time slots, the larger total latency achieved by each approach, since tasks successively arrive to the base station over time. We can find that the cloud processing only approach achieves the worst performance, due to the long data transmission delay to the cloud. Therefore, it is highly recommended to extend some services to the network edge. Furthermore, it can be found that the task-prior greedy approach achieves better performance than the other baselines, as more tasks can be served on the edge server. However, it obtains larger total latency than our approach, due to the increase of

the switching latency introduced by frequent service migration. Obviously, our approach achieves the best performance under different settings of the number of time slots. Specially, when there are 250 time slots, the total latency achieved by our approach is 10.91%, 19.82%, 28.11%, 36.86% lower than the four baselines, respectively. Figure 6 presents the rewards achieved by the five approaches, when the number of service belongs to $\{2, 4, 6\}$, respectively. When there are only two services, the performance of our approach is as same as the service-prior greedy approach and the task-prior greedy approach, because the two services can be placed on the edge server at the same time. With the increase of the number of services, our approach significantly outperforms the baselines, which indicates our approach is suitable to complex service placement scenarios. Specially, when there are six services, the total latency obtained by our approach is 14.96%, 21.17%, 28.48%, 36.47% lower than the four baselines, respectively.

6 Conclusion

In this work, we focus on the online service placement problem in a 5G-enabled mobile edge computing system, in which numerous tasks randomly arrive to the edge (i.e., base station) in real time. The base station should make service placement decisions in real time, it is necessary to find an optimal service placement policy to minimize the total completion latency. However, which services are placed on the base station should be dynamically adjusted according to the computation demands. Moreover, the additional latency incurred by service migration and the time spent for waiting in queues should be considered. To overcome the challenges, we formulate the service placement problem by an MDP model and propose an DQN-based online service placement approach to obtain the optimal service placement policy. Extensive simulations are conducted to validate the performance of our proposed approach. The simulation results show that our approach achieves higher reward compared with baseline methods, no matter how the number of time slots and different types of services.

Acknowledgement. This research is supported by NSFC (No. 61802245), the Shanghai Sailing Program (No. 18YF1408200), and STSCM (No. 19511121000). This work is also supported by the Open Project Program of Shanghai Key Laboratory of Data Science (No. 2020090600002).

References

1. Aissioui, A., Ksentini, A., Gueroui, A.M., Taleb, T.: On enabling 5G automotive systems using follow me edge-cloud concept. IEEE Trans. Veh. Technol. **67**(6), 5302–5316 (2018)
2. Al-Shuwaili, A., Simeone, O.: Energy-efficient resource allocation for mobile edge computing-based augmented reality applications. IEEE Wirel. Commun. Lett. **6**(3), 398–401 (2017)

3. Ascigil, O., Phan, T.K., Tasiopoulos, A.G., Sourlas, V., Psaras, I., Pavlou, G.: On uncoordinated service placement in edge-clouds. In: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 41–48. IEEE (2017)
4. Farhadi, V., et al.: Service placement and request scheduling for data-intensive applications in edge clouds. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1279–1287. IEEE (2019)
5. Gao, B., Zhou, Z., Liu, F., Xu, F.: Winning at the starting line: joint network selection and service placement for mobile edge computing. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1459–1467. IEEE (2019)
6. He, T., Khamfroush, H., Wang, S., La Porta, T., Stein, S.: It's hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 365–375. IEEE (2018)
7. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing-a key technology towards 5G. ETSI White Paper **11**(11), 1–16 (2015)
8. Ksentini, A., Taleb, T., Chen, M.: A Markov decision process-based service migration procedure for follow me cloud. In: 2014 IEEE International Conference on Communications (ICC), pp. 1350–1354. IEEE (2014)
9. Liu, J., Zhong, L., Wickramasuriya, J., Vasudevan, V.: uWave: accelerometer-based personalized gesture recognition and its applications. Pervasive Mobile Comput. **5**(6), 657–675 (2009)
10. Mao, Y., You, C., Zhang, J., Huang, K., Letaief, K.B.: A survey on mobile edge computing: the communication perspective. IEEE Commun. Surv. Tutor. **19**(4), 2322–2358 (2017)
11. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
12. Nadembega, A., Hafid, A.S., Brisebois, R.: Mobility prediction model-based service migration procedure for follow me cloud to support QoS and QoE. In: 2016 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2016)
13. Ouyang, T., Li, R., Chen, X., Zhou, Z., Tang, X.: Adaptive user-managed service placement for mobile edge computing: an online learning approach. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 1468–1476. IEEE (2019)
14. Ouyang, T., Zhou, Z., Chen, X.: Follow me at the edge: mobility-aware dynamic service placement for mobile edge computing. IEEE J. Sel. Areas Commun. **36**(10), 2333–2345 (2018)
15. Poularakis, K., Llorca, J., Tulino, A.M., Taylor, I., Tassiulas, L.: Joint service placement and request routing in multi-cell mobile edge computing networks. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, pp. 10–18. IEEE (2019)
16. Sutton, R., Barto, A.: Introduction to Reinforcement Learning. MIT Press, Cambridge (1998)
17. Taleb, T., Ksentini, A., Frangoudis, P.: Follow-me cloud: when cloud services follow mobile users. IEEE Trans. Cloud Comput. **PP**, 1 (2016)
18. Wang, L., Jiao, L., He, T., Li, J., Mühlhäuser, M.: Service entity placement for social virtual reality applications in edge computing. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 468–476. IEEE (2018)
19. Wang, S., Dey, S.: Adaptive mobile cloud computing to enable rich mobile multimedia applications. IEEE Trans. Multimed. **15**(4), 870–883 (2013)

20. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992). https://doi.org/10.1007/BF00992698
21. Xu, J., Chen, L., Zhou, P.: Joint service caching and task offloading for mobile edge computing in dense networks. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 207–215. IEEE (2018)
22. Zeng, D., Gu, L., Pan, S., Cai, J., Guo, S.: Resource management at the network edge: a deep reinforcement learning approach. IEEE Netw. **33**(3), 26–33 (2019)